Universidad de las Ciencias Informáticas
Facultad 3

Diseño e Implementación de un sistema informático para la gestión del proceso de Diligencias Previas en los Tribunales Provinciales Cubanos.

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

Autor: Jorge Alberto Marrero Terga.

Tutor: Ing. Sándor Rodríguez Prieto.

Junio 2011

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste se firma la presente a los ____ días del mes de junio del año 2011.

_____	_____
Jorge Alberto Marrero Terga	Ing. Sándor Rodríguez Prieto
Autor	Tutor

Datos de Contacto

Autor: Jorge Alberto Marrero Terga.

Correo Electrónico: jamarrero@estudiantes.uci.cu

Tutor: Ing. Sándor Rodríguez Prieto.

Correo Electrónico: sprieto@uci.cu

Agradecimientos

En primer lugar agradecer a mi familia en general, por ser constante veladora de mis largos años de estudios.

En especial a mi abuela Adamelis.

A mi madre, por todo su amor y por su apoyo incondicional. Por su fuerza ante la vida.

A mi novia Sache por su amor y comprensión de todos estos meses.

A todos mis amigos en la universidad que han contribuido a mi formación.

A mi tutor Sándor por el tiempo dedicado, la experiencia y los consejos.

Dedicatoria

Dedico este trabajo mi familia porque es mi inspiración y siempre los llevo presente, y porque un logro mío también es un logro de ellos. En especial a mi abuelo Bernardo que aunque ya no está entre nosotros sigue en nuestros corazones. A mi abuela Adamelis y mi madre Beatriz.

Resumen

Los Tribunales Populares Cubanos iniciaron una serie de transformaciones con el objetivo de agilizar los trámites judiciales. Como parte de estas transformaciones se concibió el proyecto Tribunales Populares del País (TPC), el cual está constituido por varios módulos, entre los que se encuentra el Económico. El presente trabajo incluye las etapas de Diseño e Implementación sobre la base del análisis de la arquitectura que soporta el sistema. Al finalizar se muestran los resultados obtenidos al realizar pruebas al código desarrollado durante la etapa de Implementación del sistema de gestión para el proceso de Diligencias Previas. Todo esto para contribuir a la prevención de hechos ilícitos y manifestaciones de corrupción.

Índice de contenido

Introducción

En la actualidad existe la convicción generalizada del profundo impacto que la informática está produciendo en la sociedad actual, generando transformaciones en los ámbitos de la vida social y por supuesto en la ciencia del Derecho. Este avance tecnológico está alterando las condiciones en que se desenvolvían diferentes ciencias (Pelaez, 2002).

El avance de la tecnología informática en el campo del Derecho representa hoy un importante y necesario campo de estudio que trate de delimitar los alcances y contenidos que derivan de esa relación. Por lo que en la Universidad, específicamente en el centro CEGEL adscrito a la Facultad 3 un grupo de trabajo, integrado por profesores y estudiantes están desarrollando un sistema de gestión informático para los Tribunales Populares Cubanos (TPC) (Figueras Rodríguez, 2010).

Los tribunales constituyen un sistema de órganos estatales, con la función de impartir justicia del pueblo y es ejercida en su nombre por (ALARCÓN DE QUESADA, 1998):

- El Tribunal Supremo Popular.
- Los Tribunales Provinciales Populares.
- Los Tribunales Municipales Populares.

Los Tribunales cuentan con diferentes procesos distribuidos en cinco materias que son tratadas a todos los niveles. Las materias anteriormente mencionadas son:

- Materia Administrativa.
- Materia Civil.
- Materia Económica.
- Materia Laboral.
- Materia Penal.

La solución informática planteada para los TPC está conformada por siete subsistemas o módulos como comúnmente se le conocen. Entre estos se encuentra el subsistema Económico que es el encargado de resolver conflictos económicos entre empresas o entre empresas y personas (CEGEL, 2009).

Actualmente en la sala de lo económico del Tribunal Provincial el proceso de Diligencias Previas se lleva a cabo de forma manual. Lo que provoca que el trabajo se demore mucho más y en muchas ocasiones no se cumpla el término de radicación de un escrito, además que trae otras consecuencias como la introducción de errores de repetición en el número de los asientos de los escritos, tachaduras, saltos en los espacios de las anotaciones, borraduras, entre otros. Muchos documentos son generados varias veces a causa del proceso manual, pues impide la tenencia de registros actualizados por la demora del llenado de datos y de errores que se puedan cometer durante la presentación de un documento o la radicación del mismo (CEGEL, 2009).

Es de señalar que el proceso Diligencias Previas se caracteriza por la celeridad en cuanto a los términos con que cuenta para cada momento procesal. Al ser muy cortos conlleva a que el expediente se mantenga en tramitación constantemente que sumado al volumen de expedientes tanto jueces como el personal de secretaría se mantenga en un estado de presión de trabajo constante. Multiplicándose el diligenciamiento de notificaciones por parte del alguacil (Martínez Reyes, 2008).

La búsqueda de un determinado expediente se torna complicada por la gran cantidad de archivos guardados, a pesar que en los estantes se guardan de manera organizada por año. Al pasar el tiempo las carátulas de los expedientes se deterioran y cuando hay que volverlos a usar en muchas ocasiones hay que cambiarlas (CEGEL, 2009).

Teniendo en cuenta las deficiencias antes mencionadas surge la necesidad de crear un sistema capaz de gestionar los procesos que se llevan a cabo en los TPC, planteando el siguiente **problema:** ¿Cómo contribuir a la gestión del proceso Diligencias Previas en los Tribunales Provinciales Cubanos que permita elevar la calidad y la celeridad en su tramitación?

Asumiendo como **objeto de estudio:** proceso de desarrollo de software en la creación de un sistema para la gestión judicial, con el **objetivo:** realizar el Diseño e Implementación de un sistema informático para la gestión del proceso Diligencias Previas de los Tribunales Provinciales Cubanos, que permita elevar la calidad y la celeridad en su tramitación.

La presente investigación se desarrolla teniendo en cuenta como **campo de acción:** el Diseño e Implementación como fases específicas dentro del proceso de desarrollo de software en la gestión del proceso Diligencias Previas de los Tribunales Provinciales Cubanos.

La investigación está basada en la **Idea a defender:** realizando el Diseño e Implementación de un sistema informático para la gestión del proceso Diligencias Previas de los Tribunales Provinciales Cubanos se logrará elevar la calidad y la celeridad en su tramitación.

Para dar cumplimiento al objetivo propuesto se han derivado un conjunto de **objetivos específicos** orientados fundamentalmente a proveer los elementos necesarios para el diseño e implementación de la solución, los cuales se listan a continuación:

- Realizar el marco teórico de la investigación.
- Analizar la arquitectura definida para la implementación de la solución informática.
- Definir una solución de diseño para el proceso de Diligencias Previas de la sala de lo económico de los Tribunales Populares Cubanos.
- Diseñar la solución, diagramas de clases y los diagramas de interacción, para dicha solución informática.
- Implementar la solución diseñada.
- Analizar y evaluar los resultados obtenidos, luego de aplicar diferentes métricas para el control y aseguramiento de la calidad en diseños de sistemas de software.

Las **tareas de investigación:** trazadas quedaron estructuradas de la siguiente forma:

- Estudio del proceso Diligencias Previas de la materia Económico en los Tribunales Populares Cubanos.
- Estudio de la plataforma de desarrollo y las herramientas asociadas a ésta.
- Estudio de la Metodología de Desarrollo de Software.
- Estudio de los Paradigmas de Programación.
- Estudio y selección de los Patrones de Diseño más factibles para esta propuesta de solución.
- Realización de los Diagramas de Clases del Diseño.
- Realización de los Diagramas de Secuencia.
- Realización del Diagrama de Componentes.
- Implementación de los componentes.
- Validación de la solución propuesta.

Para dar respuesta a las tareas planteadas con anterioridad se proponen **Métodos Científicos** entre los que se encuentran los siguientes:

Teóricos

Método histórico - lógico: Este método permite hacer un estudio de los antecedentes de cómo se realiza el proceso de Diligencias Previas en la sala de lo económico de los Tribunales Populares Cubanos para un mejor entendimiento y compresión de las características de esta entidad.

Método analítico- sintético: El estudio y análisis de la bibliografía permite una correcta formulación del problema y los objetivos, la elaboración de las conclusiones y recomendaciones logrando resultados satisfactorios en la investigación.

Sistémico: Para determinar los elementos del diseño que componen el sistema y sus relaciones.

Modelación: Consiente la creación de modelos que representan abstracciones con el objetivo de comprender el funcionamiento del proceso Diligencias Previas asociado a la sala de lo Económico de los Tribunales Provinciales Cubanos.

Empíricos

Entrevista: Tuvo lugar una conversación planificada a las 10:00 horas del miércoles 8 de Diciembre de 2010 en la Universidad de las Ciencias Informáticas, Cuba, por interés del

autor. El entrevistado es el líder del módulo Económico del proyecto Tribunales Populares Cubanos Sándor Rodríguez Prieto. La finalidad de la entrevista fue obtener información relacionada con el sistema informático que se propone.
El presente documento se estructura en 3 capítulos y anexos que incluyen los aspectos relacionados con el diseño e implementación del sistema de gestión para el proceso de Diligencias Previas.

El **Capítulo 1** se enmarca en la Fundamentación Teórica de la investigación. Para ello se aborda el tema del proceso de Diligencias Previas en el subsistema económico de los Tribunales Populares Cubanos mediante todo un estudio del arte de todos los procesos existentes en el subsistema económico. Además se describen en este capítulo las principales herramientas, plataforma y metodologías usadas.

El **Capítulo 2** se analiza la arquitectura propuesta por el equipo de arquitectura del proyecto Tribunales Populares Cubanos, las diferentes capas que lo componen, asi como los patrones de diseño utilizados. Además se exponen los aspectos técnicos del proceso de Diligencias Previas, expresados a través del modelo de diseño conformado por los diagramas de clase y de secuencia y además el modelo de implementación compuesto por el diagrama de componentes y el diagrama de despliegue previamente diseñados.

El **Capítulo 3** expone toda la validación hecha a la solución obtenida. Se presentan los resultados obtenidos de la aplicación de métricas al diseño como TOC y RC, además de pruebas de caja negra con casos de prueba previamente diseñados.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se aborda el tema del proceso de Diligencias Previas de la sala de lo económico de los Tribunales Populares Cubanos a través de un análisis de la realización del proceso en los ámbitos internacional y nacional. Además se describen en este capítulo las principales herramientas, plataforma y metodología propuestas para el desarrollo.

1.1. Proceso Diligencias Previas

Con el propósito de solicitar Diligencias Previas al módulo Ejecutivo acuden las personas a las Salas de lo Económico de los Tribunales Provinciales. La tramitación de las Diligencias Previas antes de la promulgación de la instrucción 172/2003, del Consejo de Gobierno del Tribunal Supremo Popular (TSP), no era uniforme en todas las salas, las que empleaban el procedimiento y término que entendían pertinente acorde a la interpretación que cada uno hacía de la ley (Martínez Reyes, 2008).

Con la instrucción 172/2003 del Consejo de Gobierno del TSP se estableció que las Diligencias Previas al proceso Ejecutivo debían ser llevadas como simple trámite judicial independiente y anterior al proceso Ejecutivo. En la propia instrucción se concede a la parte actora la facultad de presentar demanda en proceso Ejecutivo en base al título reconocido dentro de los términos de prescripción establecidos en el ordenamiento civil común (Martínez Reyes, 2008).

Se define como Diligencias Previas aquel acto judicial preparatorio, que sucede antes del módulo Ejecutivo y cuyo fin es obtener la confesión de la deuda y/o el reconocimiento de documento o el de su firma, los que a partir de este momento podrán adquirir fuerza ejecutiva como títulos de crédito, líquido, vencidos y exigibles, y de esta forma declarar preparada la acción ejecutiva. Se asegura, por tanto, el pago de una deuda reconocida por la entidad deudora (Bello, 2006).

La Ley de Procedimiento Civil Administrativo y Laboral, en su artículo 487 prevé el procedimiento a seguir cuando para preparar la acción ejecutiva se solicitare el reconocimiento de un documento y/o la confesión de la deuda; trámite que no constituía

competencia de las Salas de lo Económico hasta la entrada en vigor del Decreto Ley 223/2001 de la jurisdicción y competencia de estas, dictado por el Consejo de Estado (Martínez Reyes, 2008).

Teniendo en cuenta la situación creada y las omisiones existentes en la regulación legal que hasta el momento existía se hizo necesario dictar la Instrucción172/2003 de veintiocho de marzo del 2003, del Consejo de Gobierno del TSP donde se estableció que la realización de las Diligencias Previas debía hacerse de manera independiente y anterior al proceso Ejecutivo sin darse paso de inmediato y obligatoriamente al proceso Ejecutivo, pudiéndose iniciar por la parte actora en cualquier momento posterior dentro de los términos de prescripción establecidos en el ordenamiento civil común (Martínez Reyes, 2008).

Una vez que la parte actora presenta el escrito de Diligencias Previas se detectan dificultades que atentan contra el buen desarrollo del proceso y en la que no sólo los jueces desempeñan un papel fundamental, sino también los directivos de empresas y asesores jurídicos de estas entidades (Martínez Reyes, 2008).

Los documentos que se presentan a la sala en ocasiones no cumplen con el requisito de ser privados, para atribuirle fuerza ejecutiva como título de crédito líquido, vencido y exigible. En otros casos se presentan documentos firmados por personas que no tienen la representatividad requerida para reclamar la existencia de la deuda, el hecho trae consigo que el documento suscrito pierda fuerza. Así mismo no contienen la declaración de voluntad de ambas partes que demuestre la existencia de un compromiso de pago con límite en el tiempo, a fin de atribuirle la mencionada fuerza (Martínez Reyes, 2008).

La intervención del director de la entidad en estos casos es de vital importancia, por cuanto es él el que ostenta la verdadera representación de la misma, hecho que hace posible que pueda admitir la existencia o no de la deuda contraída por la entidad, dando fe de su actualidad y vigencia. En otros casos no comparecen al acto de reconocimiento de documentos y/o confesión de la deuda; en el mejor de los casos porque están conscientes de la existencia de ésta y en otros por falta de cultura jurídica y/o sentido de

pertenencia, que los lleva a no proteger adecuadamente los intereses económicos de la persona jurídica que representan (Martínez Reyes, 2008).

Otra situación que se manifiesta en las Diligencias Previas es que se utiliza habilidosamente para evitar llevar el caso en un proceso Ordinario y por lo tanto no tener que presentar pruebas relacionadas con el cumplimiento de la obligación, lo que redunda en la excesiva utilización de esta vía y en el aumento innecesario de la radicación de las Salas de lo Económico (Martínez Reyes, 2008).

Finalmente se declara preparada la ejecución y la parte actora tiene hasta un año para presentar demanda al proceso Ejecutivo por lo que con el propósito de ingresar a su cuenta de forma ágil y eficaz la cuantía adeudada, utiliza la vía judicial en el menor tiempo posible aconteciendo con frecuencia que en ese momento es que despierta la presunta ejecutada e intenta alegar por lo general la existencia de la deuda; resultando meritorio destacar que en virtud de lo anteriormente expuesto, a nuestro criterio, es excesivo el término concedido para presentar demanda en el módulo ejecutivo proveniente de Diligencias Previas, prevista en el artículo 486, apartado 2 y 3 de la Ley de Procedimiento Civil, Administrativo y Laboral (Martínez Reyes, 2008).

La sencillez en la tramitación de las Diligencias Previas, no puede incidir en que no se cumpla con el procedimiento establecido para ellas. Tampoco pueden dar lugar a que se encubran a través de ellas operaciones realizadas con mala fe por alguna de las partes y que luego no sea posible resolverlas o pongan en peligro no solo la economía de las entidades implicadas sino también la economía nacional (Núñez, 2006).

1.2. Soluciones informáticas existentes

Madrid: En un proceso de renovación de los sistemas informáticos a partir de marzo del 2010 se implanta el sistema Minerva. El nuevo programa informático de gestión de expedientes judiciales Minerva (Sistema de Gestión Integral de la Información Bibliográfica). Ideado para potenciar la comunicación entre los órganos judiciales y tener un mayor control sobre las ejecutorias penales (sentencias pendientes de ejecutar). Evita

fallos judiciales y favorece el intercambio de información entre los juzgados (Muñoz, 2010).

Islas Canarias: El sistema informático ATLANTE desarrollado registra todas las diligencias realizadas por las distintas instituciones. El sistema remite la información adecuada al siguiente paso, aunque los funcionarios esperan que lleguen los documentos en papel para procesarlos. La versión que está en explotación está basada en Lotus Notes y bases distribuidas interconectadas por mensajería (EUROPA, 2009).

Vasco: Se implementa un sistema llamado EJ, el cual está desarrollado en lenguaje de programación C, utilizando bases de datos Oracle y desplegado sobre sistema operativo UNIX. Al igual que la aplicación de Canarias, este registra todas las diligencias realizadas por las distintas instituciones. El sistema remite la información adecuada al siguiente paso, aunque los funcionarios esperan que lleguen los documentos en papel para procesarlos. Están actualmente desarrollando una segunda versión del sistema realizada en ambiente WEB y SQL Server como sistema gestor de base de datos (País, 2008).

Es necesario destacar que estos tres sistemas no pueden ser asumidos económicamente por nuestro país debido a sus altos costos y mantenimiento de sus licencias.

Villa Clara, Cuba: El sistema SisProP fue propuesto para abarcar las instancias Supremo y Provincial en el módulo penal, pero únicamente se desarrolló la tramitación de los procesos y apelaciones competencia del tribunal provincial. El sistema fue mal concebido desde su inicio y presenta un grupo de limitaciones que se resumirán a continuación, que tienen como causa fundamental ese error de concepción (Castro Morell, 2008):

- No obtiene datos de la fase judicial del proceso.
- No aporta estadística, ni información alguna y al no haberse programado la introducción de los datos de la fase judicial señalados en el punto uno no es solucionable.
- El nivel de informatización que supone el sistema es mínimo y en el caso de las apelaciones es prácticamente nulo.
- El sistema no valida casi ningún dato.

- El sistema está programado en Delphi y corre sobre SQL Server, por lo que es incompatible con el software libre en el que se están programando los sistemas generales de cada materia judicial, fue dictaminado así por el equipo de especialistas de la UCI y fue aceptado por el propio especialista informático que hizo este sistema, Msc. Daniel E. Castro Morel, quien expresó además no estar en posibilidades de reprogramarlo si se decidiera hacerlo.

Ciudad de La Habana, Cuba: En la sala económica provincial de Ciudad de la Habana cuentan con un software hecho en el 2002 para el área de la estadística llamado SisEco, sencillo. En él se insertan los documentos radicados manualmente y las salvas diariamente se guardan en disquetes. La secretaria de estadística recoge el libro de radicación de escritos (LRE) y en un período de un día inserta los datos en la aplicación. Esto con el inconveniente, de que tiene que esperar a que el libro esté disponible en la mesa de radicación para poderlo usar. El sistema cada cinco años borra los datos de los años anteriores, quedando solamente la información asentada en los libros (Universidad de la Habana, 2002).

A partir del estudio realizado concluimos que no es factible el uso de ninguno de los sistemas anteriormente analizados.

1.3. Desarrollo de Software.

1.3.1. Metodologías de Desarrollo de Software.

Durante el ciclo de vida del desarrollo de software se deben completar una serie de tareas para obtener un producto. A menudo se dice que los distintos componentes de software deben pasar por distintas fases o etapas durante el ciclo de vida.
Se habla de detalles organizativos, de un "estilo" de hacer las cosas. Pero siguiendo un poco más allá de un simple estilo, formalizando ese "estilo" añadiendo algo de rigurosidad y normas se obtiene una metodología (RUP, 2010).
Existen dos grupos en los cuales se dividen estas metodologías: tradicionales y las ágiles, con marcadas diferencias.
A continuación se listan algunas:

Programación Extrema –XP: es el más destacado de los procesos de desarrollo de software ágil. Surge ideada por Kent Beck, como proceso de creación de software diferente del convencional. En palabras de Beck: XP es una metodología ligera, eficiente, con bajo riesgo, flexible, predecible y divertida para desarrollar software. [1]

Los objetivos de XP son muy simples: lograr la satisfacción del cliente y potenciar al máximo el trabajo en equipo. Esta metodología trata de dar al cliente el software que él necesita y cuando lo necesita. Por tanto, se debe responder muy rápido a las necesidades del cliente, incluso cuando los cambios sean al final del ciclo de la programación. XP propone valores que deben poseer los miembros del equipo de desarrollo para desarrollar el trabajo y conseguir los planteamientos iniciales. Estos valores son: comunicación, sencillez, retroalimentación y valentía. Además, define variables para proyectos de software: coste, tiempo, calidad y ámbito; y un conjunto de prácticas básicas tales como: el juego de la planificación, pequeñas versiones, metáfora, diseño sencillo, hacer pruebas, recodificación, programación por parejas, propiedad colectiva, integración continúa, cuarenta Horas semanales, cliente In-situ, estándares de codificación.

Scrum: tiene su primera aparición en 1995, pero no es hasta el 2001 que sus creadores describen completamente la metodología[2]. Scrum es un proceso en el que se aplican de manera regular un conjunto de mejores prácticas para trabajar en equipo y obtener el mejor resultado posible de un proyecto. Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, los requisitos son cambiantes o poco definidos y la innovación, la competitividad y la productividad son fundamentales. Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, específicamente cuando: Se necesita capacidad de reacción ante la competencia.

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos (iteraciones de un mes natural y hasta de dos semanas, si así se necesita). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite. Las actividades

[1] Extreme Programming Explained: Embrace Change. Kent Beck, (1999).
[2] Agile Software Development with Scrum. Ken Schwaber y Mike Beedle. (2001)

fundamentales del proceso son la planificación y la revisión. Para ello Scrum plantea la ejecución de una serie de reuniones, estas son (Scrum, 2009):

- Scrum Diario (Daily Scrum)
- Scrum de Scrum
- Reunión de Planificación del Sprint (Sprint Planning Meeting)
- Reunión de Revisión del Sprint (Sprint Review Meeting)
- Retrospectiva del Sprint (Sprint Retrospective)

El Proceso Unificado Racional (Rational Unified Process - RUP) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML[3], constituye una metodología robusta, la más estándar y utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Es una metodología para la ingeniería de software que va más allá del mero análisis y diseño orientado a objetos para proporcionar una familia de técnicas que soportan el ciclo completo de desarrollo de software. El resultado es un proceso basado en componentes, dirigido por casos de uso, centrado en la arquitectura, iterativo e incremental (Guerra, 2010).

En RUP se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo principales. Los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo. Figura 1 tomada de (RUP, 2010)

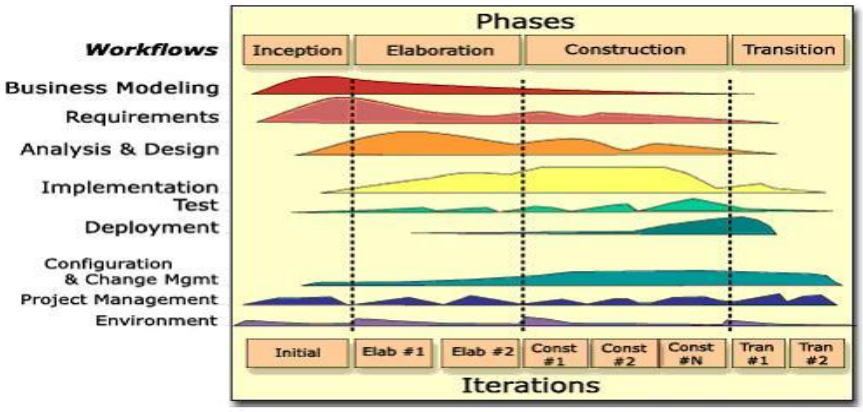


Fig. 1 RUP en dos dimensiones

[3] Unified Modelling Language por sus siglas en inglés. Versión original en 1997. Se ha convertido en el estándar para notaciones de modelado por idea de tres expertos en modelado: Booch, Rumbaugh y Jacobson.

Características principales de RUP:

- Guiado por los Casos de Uso: los Casos de Uso son el instrumento para validar la arquitectura del software y extraer los casos de prueba.
- Centrado en la arquitectura: los modelos son proyecciones del análisis y el diseño constituye la arquitectura del producto a desarrollar.
- Iterativo e incremental: durante todo el proceso de desarrollo se producen versiones incrementales (que se acercan al producto terminado) del producto en desarrollo.

Por parte del equipo de arquitectura del proyecto se decide utilizar para el desarrollo RUP, debido a que Tribunales Populares Cubanos se caracteriza por ser un proyecto grande en lo que se refiere a recursos humanos y a tiempo de desarrollo, con gran número de iteraciones, y por consiguiente de artefactos generados en cada una de sus fases. Teniendo en cuenta las dimensiones que posee el proyecto, y a causa de su característica de ser a distancia, añadiendo a esto el gran volumen de documentación que se precisa. Se considera además, que resulta más adaptable para los proyectos a largo plazo que necesiten un desarrollo iterativo capaz de mantener un buen control sobre los cambios y que facilite sobre todo el trabajo a distancia con los clientes, aplicando los criterios de flexibilidad que ofrece, robustez en la gestión del proyecto y los cambios, la cantidad de artefactos que se generan en forma de entregables al cliente y la adaptabilidad ante un sistema de gran tamaño. En base a esta decisión, se define como lenguaje de modelado UML (Montané Izaguirre, 2010).

1.3.2. Herramientas CASE.

De acuerdo con Kendall la Ingeniería de Sistemas Asistida Por Ordenador es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo. Su objetivo es acelerar el proceso para el que han sido diseñadas, en el caso de CASE para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas.

Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, generación automática de parte del código con el diseño dado, compilación automática, documentación o detección de errores entre otras (case-tools, 2011).

Se clasifican en disímiles tipos, según las funciones específicas que brindan, fundamentalmente las referidas a la fase que dan apoyo dentro del desarrollo de software, ya sea la captura de requisitos y el análisis y diseño o todo el ciclo de vida hasta la implementación y pruebas. A continuación se muestran algunas de ellas:

Rational Rose Enterprise: brinda soporte para la generación de código a partir de modelos en Ada, ANSI C++, C++, CORBA, Java™/J2EE™, Visual C++® y Visual Basic®. Como todos los demás productos Rational Rose, proporciona un lenguaje común de modelado para el equipo que facilita la creación de software de calidad rápidamente (IBM, 2008).

Características generales:

- Crea modelos independientemente del lenguaje de arquitectura de software, necesidades empresariales, activos reutilizables y comunicación a nivel de gestión.
- Posee un único entorno de diseño.
- Presenta un desarrollo basado en modelos con soporte para UML.
- Brinda soporte para múltiples modelos en la Arquitectura basada en Modelos (MDA).
- Se ejecuta de forma independiente o integrada con Microsoft Visual Studio .NET.
- Crea arquitecturas independientes de la plataforma que se pueden implementar en plataformas Java y .NET.
- Uso de patrones definibles por el usuario para crear, personalizar y aprovechar patrones de diseño arquitectónico.

- Usa referencias cruzadas entre modelos y el control de versiones a nivel de clase y diagrama para la estructuración ajustable a cualquier proyecto.
- Conserva la capacidad de rastreo entre los modelos de análisis, diseño e implementación. Modela de formas libres. Soporta publicación web y generación de informes.

Visual Paradigm: es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software, análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML (Visual Parading , 2004). A continuación se listan sus características:

- Soporte de UML versión 2.1.
- Diagramas de Procesos de Negocio.
- Modelado colaborativo con CVS y Subversion (nueva característica).
- Interoperabilidad con modelos UML2 (meta modelos UML 2.x para plataforma Eclipse) a través de XMI (nueva característica).
- Ingeniería de ida y vuelta. Ingeniería inversa - Código a modelo, código a diagrama. Ingeniería inversa Java, C++, Esquemas XML, XML, .NET exe/dll, CORBA IDL.
- Generación de código - Modelo a código, diagrama a código.
- Editor de Detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Diagramas de flujo de datos.
- Soporte ORM - Generación de objetos Java desde la base de datos.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.

- Generador de informes para generación de documentación.
- Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.

Enterprise Architect: es una herramienta de modelado basada en UML 2.1 flexible, completa y poderosa para plataformas Windows. Es orientada a objetos y provee el límite competitivo para el desarrollo de sistemas, administración de proyectos y análisis de negocios. Ayuda al desarrollador de software a trazar especificaciones de alto nivel a modelos de análisis, diseño, implementación, pruebas y mantenimiento, usando UML, SysML, BPMN y otros estándares abiertos para modelado (Enterprise Architect , 2009). Entre sus principales características se pueden destacar:

- Con un repositorio de alto desempeño, facilita que los grandes equipos compartan la misma visión del desarrollo del producto.
- Provee trazabilidad completa desde los modelos de requisitos, análisis y diseño, hasta la implementación y despliegue.
- Generación potente de documentación.
- Generación e ingeniería inversa de código fuente. Soporta la generación e ingeniería inversa de código fuente para muchos lenguajes populares. Las integraciones para Eclipse y Visual Studio .NET, les proveen a los desarrolladores un acceso directo a los diseños y capacidades de modelado justo dentro del IDE. Las plantillas de generación de código le permiten personalizar el código fuente generado de acuerdo con las especificaciones de su compañía.
- Permite que se construya, pruebe, depure, active y ejecuten scripts de despliegue, todo dentro del entorno de desarrollo de Enterprise Architect. Con la capacidad de generar clases de prueba NUnit y JUnit desde clases origen usando transformaciones MDA y la integración del proceso de pruebas directamente en el IDE de Enterprise Architect, ahora puede integrar el modelado con UML y el proceso de compilación/prueba/ejecución/despliegue.

Enterprise Architect es utilizado por varios tipos de desarrolladores de programas de sistemas para una amplia gama de industrias. Enterprise Architect tiene una fuerte base

de usuarios en más de 60 países del mundo. Es también utilizado efectivamente para propósitos de entrenamiento en UML y arquitecturas de negocios en muchos prominentes colegios, compañías de entrenamiento y universidades alrededor del mundo.

ER/Studio Enterprise: cuando se hace la planificación de la base de datos, la primera etapa del ciclo de vida de las aplicaciones de bases de datos, también se puede escoger una herramienta CASE. Su uso puede mejorar considerablemente la productividad en el desarrollo de una aplicación de bases de datos. ER / Studio Enterprise dentro de la industria de software es una herramienta de modelado para el análisis, visualización y comunicación de base de datos, aplicaciones de diseños de datos y arquitectura de la información. La combinación de procesos, datos, modelado UML, y presentación de reportes en un potente entorno de diseño en multi-niveles son algunas de sus principales características (ER/Studio Enterprise , 2008).

El uso del ER / Studio Enterprise trae como beneficios:

- Promoción de la reutilización de datos y la colaboración en tiempo real a través de una gestión eficaz del modelo de la empresa y la capacidad de publicación de metadatos.
- Mejora de la visibilidad y la calidad de información con la ingeniería inversa y el soporte al ciclo de vida completo de base de datos.
- Comunicación efectiva de los modelos de toda la empresa con informes de metadatos en tiempo real y herramientas de publicación.

Por parte del equipo de arquitectura del proyecto se decide utilizar como herramienta CASE Visual Paradigm en su versión 6.4 ya que es como una herramienta de análisis evaluada muy potente y orientada a proyectos grandes donde el trabajo en equipo es esencial. Esta herramienta CASE describe la metodología seleccionada, RUP en este caso, en muy buena medida. Visual Paradigm se integra bien con UML en sus últimas versiones y proporciona generación de código en varios lenguajes. Además ya ha sido utilizada por los roles del proyecto implicados y resulta fácil de usar, rápida de entender y muy manejable. Además se selecciona por parte del equipo de arquitectura del proyecto para el modelado de los datos el Embarcadero ERStudio, una herramienta fácil de usar y multinivel, para el diseño y construcción de bases de datos tanto a nivel físico como

lógico, direccionando las necesidades diarias de los desarrolladores que construyen y mantienen aplicaciones de bases de datos grandes y de gran complejidad (Navarro, 2009).

1.3.3. Lenguajes de modelado.

El lenguaje de modelado es un conjunto estandarizado de símbolos y de modos de disponerlos para modelar un software. En la mayoría de los casos son utilizados en combinación con una metodología de desarrollo de software para realizar la especificación del desarrollo de un software y de este modo hacerlo extensivo a todo el equipo de desarrollo. El uso de un lenguaje de modelado es más sencillo que la auténtica programación. Por su robustez y fiabilidad el equipo de arquitectura del proyecto decidió que se utilizaría el lenguaje UML (Unified Modeling Lenguage, lenguaje unificado de modelado) para especificar los artefactos que se deben generar.

UML: Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios (Rumbaugh, et al., 2000).

1.3.4. Entorno de desarrollo integrado Netbeans.

Un entorno de desarrollo integrado (*integrated development environment*, IDE en inglés) proporciona un marco de trabajo amigable para gran cantidad de lenguajes de programación. Además es posible que un mismo entorno de desarrollo tenga la posibilidad de utilizar varios lenguajes de programación. Eclipse, ZendStudio y Netbeans son entornos de desarrollo integrados que pueden utilizar PHP como lenguaje de programación. Por parte del equipo de arquitectura se decidió usar este último en su versión 6.9 debido a que es un es un proyecto de código abierto, soporta lenguajes dinámicos como PHP y Java Script tiene integración con el subversión, es multi-plataforma, tiene una interfaz muy amigable e intuitiva, tiene todas las herramientas para crear aplicaciones profesionales ya sean de escritorio, empresariales, web, móviles y

aplicaciones SOA, no solo en Java sino también en C/C++ y Ruby, puedes hacer diagramas UML y es de múltiples lenguajes. Para la visualización de las interfaces se utiliza el Mozilla Firefox con el plugin Firebug para los entornos de desarrollo y prueba (Navarro, 2009).

Netbeans es una plataforma de programación utilizada para crear entornos de desarrollo. Sus creadores lo definen como un IDE para todo y nada en particular. La arquitectura de plugins de Netbeans permite, además de integrar diversos lenguajes sobre un mismo IDE, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces y ayuda en línea para librerías (netbeans.org, 2008).

1.3.5. Framework de Desarrollo.

En general, un framework es una estructura real o conceptual que pretende servir como soporte o guía para la construcción de algo que expande su estructura en algo usable. En sistemas informáticos, un framework es con frecuencia una estructura que indica cuales tipos de programas pueden o deben ser construidos y cómo estos se interrelacionarán. Algunos pueden además incluir programas existentes, interfaces de programación específicas u ofrecer herramientas de programación para usar el framework. Un framework puede ser para un conjunto de funciones dentro de un sistema y como ellos se interrelacionan, las capas de un sistema operativo, de un subsistema de aplicación, cómo la comunicación debe ser estandarizada a un cierto nivel de la red de trabajo, y así sucesivamente. Es generalmente más comprensivo que un protocolo y más prescriptivo que una estructura. Básicamente un framework evita construir una aplicación desde cero, los frameworks orientados a objetos están estructurados en librerías de clases y ahorran de cierta forma el trabajo largo y a veces tedioso de la programación, se debe tener en cuenta que para comenzar a desarrollar en un framework se deben conocer sus especificaciones. A continuación se exponen los que fueron utilizados para desarrollar el framework seleccionado (Saavedra López, 2008).

Zend Framework: Es un framework de código abierto para desarrollar aplicaciones y servicios Web con PHP 5. Se ejecuta utilizando un 100% de código orientado a objetos. Su estructura por componentes es algo único, cada componente ha sido diseñado con unas dependencias de otros componentes. Esta flexibilidad permite a los desarrolladores de la arquitectura utilizar los componentes individualmente. Ofrece una alta capacidad y robustez para la implementación del patrón Modelo Vista Controlador (Zend Company, 2008).

Zend_Ext Framework 1.0: Es una extensión del framework de Zend desarrollada por el Centro de Informatización de Gestión Empresarial y el centro de Desarrollo y Asimilación de Tecnologías de la UCID (UCI-Defensa) con el objetivo de crear un framework extensible y configurable, centrando el desarrollo de las aplicaciones en la lógica del negocio y en las interfaces de usuario y alejando a los programadores de los detalles arquitectónicos, con soporte para entornos multi entidad y para una arquitectura de sistema orientada a componentes (Gómez Baryolo, 2010).

Doctrine Framework: El framework Doctrine es una potente, multiplataforma y completo sistema para el mapeo de objeto-relacional (ORM: Object Relational Mapper, mapeador relacional de objetos) para PHP 5.2 ó superior (doctrine-project, 2008).

ExtJS Framework: Es una biblioteca de Java Script para el desarrollo de aplicaciones web interactivas usando tecnologías como AJAX, DHTML y DOM. Incluye un alto rendimiento, interfaces de usuario personalizables, bien diseñado y extensible modelo de componentes, una interfaz intuitiva y fácil de utilizar con licencias de código abierto y comercial (ExtJS, 2008).

A continuación se muestra el framework seleccionado:

Framework Sauxe: integrado al Generador de reportes dinámicos desarrollado por el Centro de Datos, el primero tiene como framework base Zend Framework y el ORM Doctrine que este trae por defecto lográndose independencia del gestor de base de datos. Además este framework es multi-entidad, gestiona los permisos a través de los roles asignados a cada usuario teniendo en cuenta rol, dominio y entidad, es configurable en la

administración de las conexiones, en cuanto a la configuración de perfil es multi-lenguaje, multi-escritorio y multi-tema. Puede integrarse con otros sistemas a nivel de servicios por ejemplo SOAP es utilizado por varios sistemas como son (Gómez Baryolo, 2010):

- Cedrux
- SIMEM
- LiberGIS
- Sistema de supervisión y control de los PSI (Hoyo).
- Sistema General PATDSI.
- Sistema de Gestión Estadística (CENTALD).
- Aduana, Transoft, Desoft, FAR, MININT Fuerza de Trabajo Calificada (MEP).
- Minería de datos (CENTALD)
- Generador de reportes (CENTALD).
- Sistema informático para la gestión de auditoría y control (SIGAC).
- Sistema para la gestión de las transportaciones militares en las FAR (DITRANS).

1.3.6. Lenguajes utilizados

1.3.6.1. Lenguajes del lado del cliente

Java Script

Java Script es un lenguaje que permite a los desarrolladores crear acciones en sus páginas web. Puede ser utilizado por profesionales y para quienes se inician en el desarrollo y diseño de sitios web. No requiere de compilación ya que el lenguaje funciona del lado del cliente, los navegadores son los encargados de interpretar estos códigos. Entre los diferentes servicios que se encuentran realizados con Java Script en Internet se encuentran: Correo, Chat, Buscadores de Información (Flanagan, 2002).

1.3.6.2. Lenguajes del lado del servidor

Se les clasifica como lenguajes del lado del servidor a los lenguajes de programación en la tecnología cliente servidor que se ejecutan del lado del servidor y de los que los cuales los usuarios solo obtienen el beneficio del procesamiento de la información (Desarrolloweb, 2002).

PHP

PHP es un acrónimo recursivo que significa Hypertext Pre-processor (inicialmente PHP Tools, o, Personal Home Page Tools) es un lenguaje "open source" interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor, que fue escrito originalmente por Rasmus Lerdorf. Ventajas de PHP:

- **Rendimiento:** El motor de PHP 5 fue rediseñado completamente con un administrador de memoria optimizado para mejorar el rendimiento.
- **Portabilidad:** PHP está disponible para sistemas operativos como UNIX, Microsoft Windows, Mac OS y OS/2. Los programas escritos en este lenguaje son portables entre plataformas.
- **Facilidad de uso:** Su sintaxis es clara y consistente, cuenta además con documentación exhaustiva para todas las funciones de su núcleo.
- **Código Abierto:** El lenguaje es desarrollado por una comunidad de programadores voluntarios que publican su código libremente en la Web y puede ser usado sin el pago de licencias o inversiones en hardware costoso. Esto reduce el costo de la producción del software sin afectar la flexibilidad o confiabilidad.
- **Soporte comunitario:** Cuenta con amplio soporte gracias a la numerosa comunidad de programadores que lo usan en todo el mundo.

Selección del lenguaje a utilizar:

Se utilizan 2 lenguajes interpretados, del lado del servidor PHP(PHP 5.2.5 con las extensiones php-pgsql, php-xsl, php-soap, php-gdi) cuyo intérprete es el motor de Zend y Java script del lado del cliente usando como intérprete Mozilla Firefox aunque se posibilita la portabilidad para otros navegadores web que cumplan con el estándar ECMA 262 que norma al ECMAScript[4] (Navarro, 2009).

1.3.7. Sistemas Gestores de Base de Datos

[4] ECMAScript es una especificación de lenguaje de programación publicada por ECMA International. El desarrollo empezó en 1996 y estuvo basado en el popular lenguaje JavaScript propuesto como estándar por Netscape Communications Corporation. Actualmente está aceptado como el estándar ISO 16262.

Para que los usuarios puedan procesar, describir, administrar y recuperar los datos almacenados en dicha base de datos se utiliza un sistema gestor de bases de datos o SGBD. En estos sistemas se proporciona un conjunto coordinado de programas, procedimientos y lenguajes que permiten a los distintos usuarios realizar sus tareas habituales con los datos, garantizando además la seguridad de los mismos. Lógicamente tiene que proporcionar herramientas a los distintos usuarios. Dentro de este concepto hoy son más populares debido a sus funcionalidades los Sistemas de Gestión de Bases de Datos Relacionales (SGBDR). Bajo este concepto se oculta toda la complejidad informática necesaria para gestionar (Desarrollo Web, 2007):

- El acceso controlado de los procesos de los usuarios a los datos
- La gestión del almacenamiento de los datos
- La gestión de las comunicaciones entre procesos clientes y servidores
- Interconexión con distintos protocolos y sistemas operativos
- Acceso a los recursos conectados a la red

A continuación se menciona el Sistema Gestor de Base de Datos que se decidió utilizar por parte de la dirección del proyecto:

PostgreSQL: es un poderoso Sistema de Base de Datos Relacional de código abierto. Cuenta con más de 15 años de desarrollo activo y una arquitectura probada que se ha ganado una sólida reputación de confiabilidad, integridad de datos y corrección. Funciona en todos los principales sistemas operativos, incluyendo Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), y Windows. Es totalmente compatible con ACID, tiene soporte completo para claves foráneas, uniones, vistas, disparadores y procedimientos almacenados (en varios idiomas). También es compatible con el almacenamiento de objetos binarios, incluyendo imágenes, sonidos o vídeo. Tiene interfaces de programación nativa de C / C + +, Java, NET, Perl, Python, Ruby, Tcl, ODBC, entre otros, y la documentación en varios idiomas.

PostgreSQL cuenta con sofisticadas funciones como el Control de Concurrencia Multi-Versión (MVCC), punto en el tiempo de recuperación, tablespaces, replicación asincrónica, transacciones anidadas (puntos de retorno), backups en línea, un planificador optimizador de consultas sofisticadas, y escribir por delante de registro para la tolerancia

a fallos. Es compatible con conjuntos de caracteres internacionales, codificación de caracteres multi-byte, Unicode, y es consciente de la configuración regional para la clasificación, caso-sensibilidad, y el formato. Es altamente escalable, tanto en la enorme cantidad de datos que puede manejar y en el número de usuarios concurrentes que puede acomodar. No hay sistemas activos de PostgreSQL en entornos de producción que manejan en exceso de 4 terabytes de datos.

Una falencia importante que gira en torno a PostgreSQL es la replicación[5]. Es un problema que algunas herramientas externas abordan sólo en parte, las formas más complicadas no están hechas todavía. Además, podría tornarse en desventaja el hecho de lo lento de importar datos desde archivos de texto o que hay un sobrecosto por tupla que en algunos casos puede ser muy grande (Lawebdelprogramador, 2011).

1.3.8. Arquitectura de Software.

La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de estos componentes según se les percibe del resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema (Aquitectos de Software, 2009). La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones. La definición oficial de Arquitectura del Software es la IEEE Std 1471-2000 que define así: “La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución” (Microsoft, 2008).

1.3.8.1. Estilos arquitectonicos.

Los estilos de arquitectura favorece atributos de calidad, y la decisión de implementar alguno de los existentes depende de los requerimientos de calidad del sistema. No es objetivo de este epígrafe hacer un análisis de los distintos estilos arquitectónicos en su totalidad, sino sólo aquellos que podrían considerarse para la construcción del sistema,

[5] Copiar datos de una base de datos a otra.

analizando las ventajas y desventajas que estos poseen para determinar si debe ser o no aceptado para el desarrollo del software. Algunos de estilos son:

Estilos de Flujo de Datos

La familia de los estilos de Flujos de Datos, enfatiza la reutilización y la modificabilidad, es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos, ejemplos de estas se pueden encontrar las arquitecturas de proceso secuencial por lotes, red de flujos de datos, y tuberías y filtros. De las arquitecturas que ofrece esta familia de estilos no se recomienda ninguna de ellas para el desarrollo de un sistema con las características que se necesita. A pesar de que es simple de entender, implementar, al igual que enfatiza la reutilización y la modificabilidad, en los procesos que complementan al proceso de inscripción el trámite pasa por determinados pasos, no se escoge pues no se hacen modificaciones sino que se le agregan metadatos a lo largo de todo el flujo y en otros casos lo único que se le hace es transmitir esos datos a diferentes departamentos pero sin modificaciones a este documento (Microsoft, 2008).

Estilos Centrados en Datos

Otro de los estilos arquitectónicos es el Centrado en Datos, esta familia de estilos enfatiza la integridad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Entre los subestilos característicos están los Repositorios, Base de Datos y Arquitectura de Pizarra. No se considera dicha arquitectura apropiada para la realización del sistema, pues está más bien enfatiza los datos y no el modelo, además no son objetivos de este tipo de estilo la reutilización y la flexibilidad del código (Microsoft, 2008).

Estilos Peer-to-Peer

Los estilos Peer-to-Peer o también llamado de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Algunos de los miembros de esta familia de estilos son las Arquitecturas Basadas en Eventos, Arquitectura Basadas en Servicios y la Arquitectura Basada en Recursos. El estilo basado en eventos más bien se utiliza en

ambientes de integración de herramientas, interfaces de usuario o cuando no se quiere que exista un acoplamiento entre el emisor y el receptor. No se selecciona los estilos Peer-to-Peer, porque el negocio de los procesos que se desean automatizar, tienen sus propias reglas, donde es muy importante el rendimiento del sistema, destacando la flexibilidad y la reusabilidad, además de que existen ciertas condiciones en las que el sistema no podría funcionar, rompiendo así con los esquemas propuestos por dichos estilos de arquitectura (Microsoft, 2008).

Estilos de Llamada y Retorno

Los estilos de Llamada y Retorno enfatizan la modificabilidad y la escalabilidad, son los estilos más utilizados o más generalizados por los sistemas a gran escala. Dentro de esta familia se encuentran la Arquitectura Orientada a Objetos y la Arquitectura en Capas (Microsoft, 2008).

- **Arquitecturas Orientadas a Objetos**: nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. Los objetos representan una clase de componentes que en algunas ocasiones son denominados managers, debido a que son responsables de preservar la integridad de su propia representación. El principal problema de esta arquitectura es que es necesario tener un nivel de abstracción superior que agrupe estos objetos de modo tal que sean más fáciles de entender y desarrollar en paralelos por diferentes grupos de desarrolladores; en sistemas de gran tamaño, las interacciones entre los objetos y su volumen se pueden volver inmanejables.
- **La Arquitectura en Capas:** podría decirse que su finalidad es abstraer las funcionalidades de una capa, de manera tal que esta pueda ser totalmente reemplazada. La Arquitectura de Capas más común es la que está compuesta por tres: Presentación, Modelo o Reglas del Negocio de la Empresa y Acceso a Datos. De esta forma, se podrá cambiar cualquiera de estas sin afectar a las restantes. Aunque tres capas es lo más común, a medida que aumenta la complejidad de los sistemas, las capas crecen. A su vez cada capa puede estar compuesta por

subcapas y una capa o subcapa puede estar compuesta por una o más clases del diseño.

Para la solución se elige por la dirección del Proyecto la Arquitectura en Capas. Primero, el estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. En segundo lugar, el estilo admite muy naturalmente optimizaciones y refinamientos. En tercer lugar, proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas. Para la aplicación se hará uso de una arquitectura en capas que proveerá alto nivel de modificabilidad y escalabilidad, permitiéndole al sistema cambios ya sea de requisitos funcionales, agregaciones o mejoras en las funcionalidades (Navarro, 2009).

1.3.9. Métricas para la medición del diseño del software.

Cuando se construye un sistema informático es de gran importancia tener en cuenta todos los aspectos necesarios para obtener un producto de software con la mayor calidad posible.

Según la terminología de la IEEE, la calidad de un sistema, componente o proceso de desarrollo de software, se obtiene en función del cumplimiento de los requerimientos iniciales especificados por el cliente o el usuario final (IEEE, 1993).

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

Métricas para Sistemas Orientados a Objetos

Los sistemas de software orientados a objetos, como el propuesto en este trabajo, poseen características especiales que lo diferencian de otros sistemas de software construidos

utilizando métodos convencionales, es por ello que las métricas aplicadas a sistemas orientados a objetos deben ajustarse a estas características.

Métricas orientadas a clases.

Se incluyen dentro de las métricas de diseño arquitectónico. Teniendo en cuenta que la clase es la unidad fundamental de un sistema orientados a objetos, es que las métricas aplicadas a las clases, sus jerarquías y colaboraciones entre ellas son un recurso de suma importancia para la estimación de la calidad del sistema. Entre algunas de las métricas que se han definido a nivel de clases se pueden mencionar:

- **Métricas de Lorenz y Kidd**

Estas métricas se dividen en cuatro categorías: tamaño, herencia, valores internos, valores externos. Las métricas orientadas a tamaño se centran en el cálculo de atributos y operaciones de una clase de manera individual; las basadas en herencia se enfocan en la reutilización de las operaciones en una jerarquía de clases; las basadas en valores internos examinan la cohesión y otros aspectos relacionados con el código; y las métricas orientadas a valores externos se centran en el acoplamiento y la reutilización.

En este grupo de métricas se encuentran: Tamaño de Clase (TC), Número de Operaciones Redefinidas por una clase (NOR) y Número de Operaciones Añadidas por una clase (NOA).

Existen además otras familias de métricas que se pueden aplicar para determinar la calidad de un diseño en un sistema, entre estos grupos se encuentran: Métricas de Li y Henry que son una modificación de las métricas de CK más cuatro nuevas métricas; y las Métricas de Hendersson-Sellers que están relacionadas fundamentalmente con el acoplamiento y la cohesión del diseño.

1.3.10. Métricas de Prueba.

Las métricas de prueba que existen se concentran en el proceso de prueba y no en las características técnicas de la prueba. Los responsables de la prueba se guían por las métricas de análisis, diseño y código. Entre ellas vale destacar la métrica de Puntos de Función, Bang, Halstead y la Complejidad Ciclomática.

Otras a destacar dentro de la mencionada clasificación son las **Métricas de Pruebas de Unidad**. Estas se incluyen dentro de las Métricas de Prueba. El término prueba de unidad se refiere a la prueba individual de unidades separadas de un sistema de software. En sistemas orientados a objetos, estas unidades son, típicamente, clases y métodos. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error. Se dice que tiene éxito si descubre un error no detectado hasta entonces. Para llevarlas a cabo se utilizan los casos de prueba, los cuales se consideran más efectivos cuando tienen una alta probabilidad de mostrar un error no descubierto hasta entonces.

Las **pruebas de caja blanca** permiten examinar la estructura interna del programa realizando un seguimiento del código fuente según va ejecutando los casos de prueba, de manera que se determinan concretamente las instrucciones, bloques en los que existen errores. Para la realización de las pruebas de unidad se diseñan casos de prueba que se encargan de examinar la lógica del sistema. Los casos de prueba garantizan que se ejerciten todos los caminos independientes de cada módulo o clase y todas las decisiones lógicas así como que se ejecuten todos los bucles y las estructuras de datos internas.

Las **pruebas de caja negra** comprueban que cada función es operativa, en la prueba de caja negra, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta. Permiten detectar funcionamiento incorrecto o incompleto, errores de interfaz, errores de acceso a estructuras de datos externas, problemas de rendimiento y/o errores de inicio y terminación.

Conclusiones Parciales.

Al finalizar este capítulo se arriba a las siguientes conclusiones:

Los conceptos estudiados resultaron de gran importancia para una mejor comprensión del objetivo de este trabajo.

Las metodologías, herramientas y lenguajes de programación que serán utilizados son los idóneos para el desarrollo del sistema y se resumen en las siguientes:

- RUP, como metodología de desarrollo.
- UML, como lenguaje de modelado.
- Visual Paradigm para UML, como herramienta de modelado CASE.
- PHP, como lenguaje de programación.

- Netbeans, como entorno de desarrollo.
- SAUXE, como frameworks de desarrollo.
- PostgreSQL y Apache, como servidores de BD y Web, respectivamente.

Además de las métricas a utilizar para validar el diseño propuesto como son Tamaño Operacional de Clase (TOC) y Relaciones Entre Clases (RC).

CAPÍTULO 2: ARQUITECTURA, DISEÑO E IMPLEMENTACIÓN

Propone la solución técnica que se propone en la investigación. Se analiza la arquitectura del sistema a desarrollar, las diferentes capas que los componen y sus componentes. Presenta el modelo de diseño conformado por los diagramas de clases y además el modelo de implementación con los diagramas de componentes y el diagrama de despliegue previamente diseñados.

2.1. Descripción de la arquitectura del sistema.

El desarrollo del sistema informático responde a una arquitectura basada en capas, compuesto por la **Capa de Presentación; Capa de Control o Negocio; Capa Acceso a Datos; Capa de Datos.** (Figura 2).

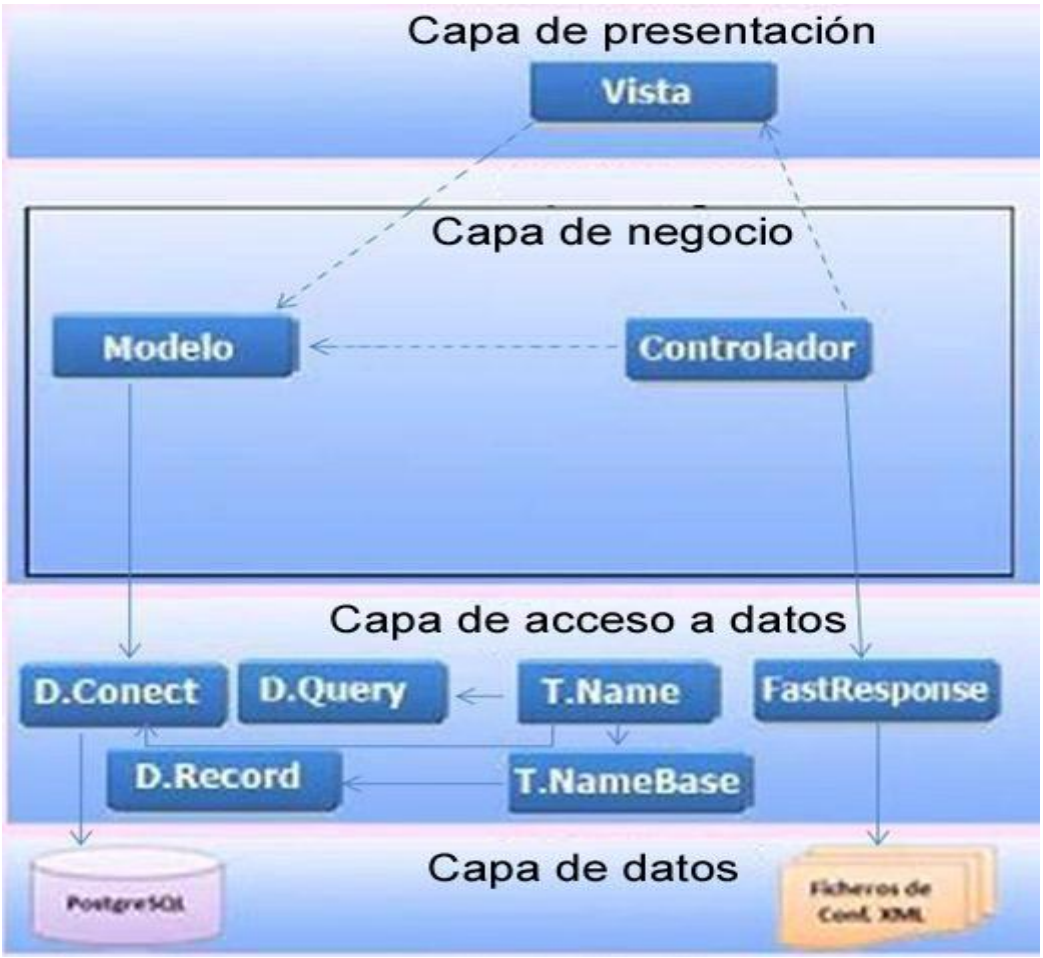


Fig. 2 Arquitectura en capas

Capa de Presentación

Es la capa superior, contiene los componentes con las que va a interactuar el usuario. Desde esta capa se pueden alcanzar las funcionalidades que brinda el negocio y mostrar o capturar la información a través de los diferentes elementos que comprende. En esta capa se emplea las facilidades que brinda el Framework ExtJS para la construcción de interfaces amigables a la vista de los usuarios. ExtJS centra su desarrollo en tres componentes fundamentales JS-File, CSS-File y Client-page y Server-Page (Gómez Baryolo, 2010).

Capa de Control o Negocio

Su diseño depende directamente del negocio específico al que se refiera cada subsistema. Esta capa recibe una petición del nivel superior, gestiona o procesa la misma, de ser necesario solicitándola a la capa de Acceso a Datos y finalmente envía una respuesta continuando el proceso en el punto donde se inició dicha petición (Gómez Baryolo, 2010).

Capa de Acceso a Datos

En esta capa estará presente el ORM (Object Relational Mapping) Doctrine, como Framework de persistencia para la comunicación con el servidor de datos mediante el protocolo PDO, también estará un Persistidor de Configuración que es el encargado de comunicarse vía XML con los Ficheros de Configuración del sistema denominado FastResponse (Gómez Baryolo, 2010).

Capa de Datos

En esta capa estará ubicado como servidor de base de datos PostgreSQL y un conjunto de Ficheros de Configuración de la arquitectura tecnológica (Gómez Baryolo, 2010).

2.2. Estándar de Codificación.

En el caso de Tribunales Populares Cubanos, el estándar de codificación fue definido por la dirección del proyecto en sus inicios, se define el formato para los siguientes puntos (Navarro, 2009):

Identación

- En el contenido siempre se identará este con tabs, nunca utilizando espacios en blanco.

Cabecera del archivo

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

A continuación se muestra un ejemplo de cómo debe quedar la cabecera de los archivos:

```
/*** Description of RegistrarEscritoPromocional

 *@versión: 5.4.2      @modificado: 10 de mayo del 2011

 * @author Jorge

 */
```

Comentarios en las funciones

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

Clases

- Las clases serán colocadas en un archivo .php aparte, donde sólo se colocará el código de la clase. El nombre del archivo será el mismo del de la clase y siempre empezará en mayúscula. En lo posible, procurar que los nombres de clase tengan una sola palabra. Las clases siguen las mismas reglas de las

funciones, por tanto, debe colocarse un comentario antes de la declaración de la clase explicando su utilidad.

Rutinas y Métodos

- Los métodos comenzarán su nombre con minúscula y si es más de una palabra la segunda comenzará con mayúscula.
- Un buen nombre para una función o método es aquel que describe todo lo que la rutina hace.
- Es recomendable que los nombres de los métodos comiencen con un verbo, seguido del objeto al que afecta. Por ejemplo: incluirDiligencia, verDiligencia, obtenerInforme.
- Cuando existan grupos de funciones que realicen operaciones similares con pequeñas diferencias, se deberá establecer un sistema de creación de nombres coherente.

Números de métodos y grados de responsabilidad

- El número de métodos de una clase debe ser inferior a 40. Esta regla debe aplicarse a todas las clases del proyecto (Rosenberg, Stapko, & Gallo, 1999).

Nombre de variables

- Excepto las constantes, todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas (lowerCamelCase)[6].
- No se utilizarán nombres de variables que puedan ser ambiguos. Por ejemplo Col es un nombre ambiguo ya que puede ser columna o color.
- Se procurará evitar dar nombres sin sentido a variables temporales. Por ejemplo: temp, i, tmp.

[6] lowerCamelCase: cuando la primera letra de cada una de las palabras es minúscula.

- Las variables booleanas deben tener nombres que sugieran respuestas o contenidos de tipo S/N, por ejemplo: Éxito, Correcto, Realizado.
- Los nombres de las variables booleanas deben ser positivos, por ejemplo: encontrado en lugar de noEncontrado.
- Se introducirán variables booleanas temporales cuando en una estructura de control (if, case, while) la expresión sea excesivamente compleja.

2.3. Patrones de arquitectura y diseño

Patrón Arquitectónico MVC (Modelo – Vista – Controlador)

Durante el desarrollo del Capítulo 1 se explica que para el desarrollo de la aplicación propuesta se utilizará el framework SAUXE, que está basado en un patrón clásico del diseño web conocido como arquitectura MVC, formado por 3 niveles: Modelo, Vista y Controlador.

A continuación se muestra la siguiente figura donde se muestra la utilización del mismo:

Fig. 3 Aplicación del Patrón MVC (Model-Controller)

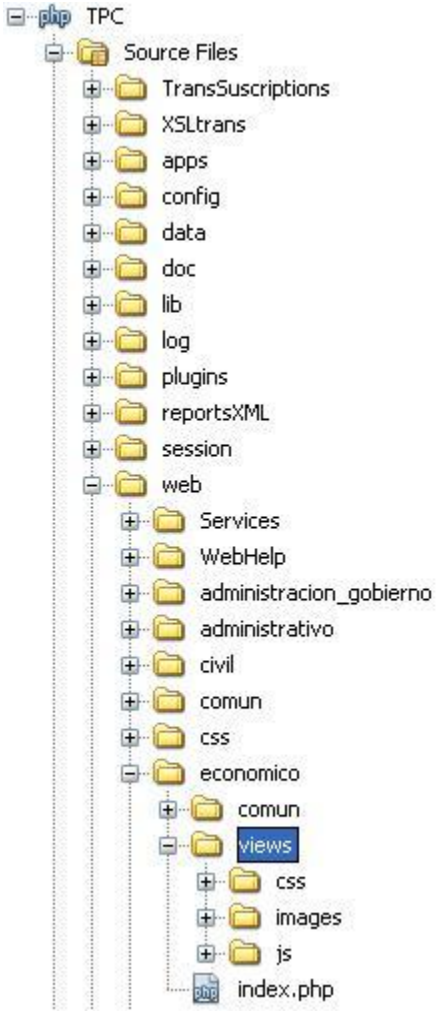


Fig. 4 Aplicación del Patrón MVC (Views)

Patrón Bajo Acoplamiento: Las clases que implementan la lógica de negocio y de acceso a datos se encuentran en el Modelo, estas clases no tienen asociaciones con las de la Vista o el Controlador por lo que la dependencia en este caso es baja, demostrándose de esta manera el uso de este patrón.

Patrón Inversión de Control (IoC): La Inversión de Control es un patrón de diseño pensado para permitir un menor acoplamiento entre componentes de una aplicación y fomentar así el reuso de los mismos. Este patrón es utilizado por el framework para crear y brindar servicios. Cada módulo del proyecto cuenta con un esquema propio en la base

de datos. Cuando un módulo necesita acceder al esquema de otro, lo que se hace es crear y brindar un servicio para que el que lo necesite lo consuma.

El framework a través del patrón IoC, define la creación de clases, para de ellas brindar los servicios necesarios. En la siguiente figura se muestra la clase ProcedimientosService ubicada en el paquete apps/económico/services donde se crea la función ObtenerProcedimientos. (Figura 5)

```php
<?php

/*
 * @package TPC
 * @copyright TPC
 * @author Jorge Alberto Marrero Terga
 * @version 1.0-0
 */
class ProcedimientosService {

        function ObtenerProcedimientos($idinstancia){
         return NprocedimientoExt::cargarprocedimiento($limit, $start);
        }


}
?>
```

Fig. 5 Representación de la clase ProcedimientosService

Cada módulo cuenta con un fichero ioc-general.xml en el cual se brindan los servicios deseados, para que otros lo consuman. La siguiente figura muestra el fichero ioc-general.xml que se encuentra en el paquete apps/económico/común/recursos/xml.

```
<economico src="economico">
        <ObtenerProcedimientos reference="">
                <inyector clase="ProcedimientosService" metodo="ObtenerProcedimientos" />
                <prototipo>
                        <parametro nombre="idinstancia" tipo="integer" />
                        <resultado tipo="array" />
                </prototipo>
        </ObtenerProcedimientos>
        <ListarPaises reference="">
                <inyector clase="DpaService" metodo="ListarPaises" />
                <prototipo>
                        <resultado tipo="array" />
                </prototipo>
        </ListarPaises>
        <ListarDpa reference="">
                <inyector clase="DpaService" metodo="ListarDpa" />
                <prototipo>
                        <parametro nombre="idpadre" tipo="string" />
                        <resultado tipo="array" />
                </prototipo>
        </ListarDpa>

</economico>
```

Fig. 6 Fichero ioc-general.xml

La siguiente figura muestra la función cargarComboProcesosAction en la clase InicioController.php del módulo Común, la cual consume el servicio ObtenerProcedimientos brindado por el módulo Económico.

```
    function cargarComboProcesosAction() {
        $idninstancia = $this->_request->getPost('idinstancia');
                $materia = $this->_request->getPost('materia');
                if($materia == 'Penal')
                        $procedimientos = $this->integrator->penal->ObtenerProcedimientos($idninstancia);
                else if($materia == 'Civil')
                        (print_r("Llamar a servicio de civil");die();)
                else if($materia == 'Laboral')
                        (print_r("Llamar a servicio de laboral");die();)
                else if($materia == 'Administrativo')
                        (print_r("Llamar a servicio de administrativo");die();)
                else if($materia == 'Economico')
                        $procedimientos = $this->integrator->economico->ObtenerProcedimientos($idninstancia);
        echo json_encode($procedimientos);
    }
```

Fig. 7 Función cargarComboProcesosAction

2.4. Diseño e Implementación.

2.4.1. Estructura de paquetes

Es importante conocer que para el desarrollo del proyecto se propone la realización de los siguientes modelos:

- Modelo de Negocio.
- Modelo de Casos de Uso.
- Modelo de Diseño.
- Modelo de Implementación.
- Modelo de Despliegue.
- Modelo de Datos.
- Modelo de Prueba.

Es por ello que existe un paquete para cada modelo donde internamente cada modelo de estos tiene un paquete para cada módulo.

Dentro del paquete Modelo de Diseño existe un paquete para cada módulo, y dentro de cada módulo se encontrará un paquete para cada caso de uso y dentro de este paquete un diagrama de clases de ese caso de uso (Figura 8).

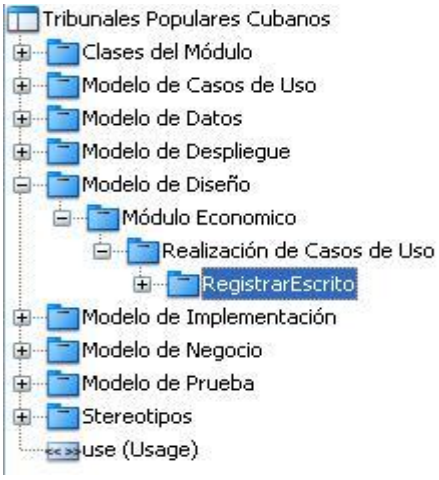


Fig. 8 Estructura interna del módulo dentro del paquete Modelo de Diseño

Además hay un paquete donde están todas las clases del proyecto, donde su organización está orientada a la arquitectura que propone el framework SAUXE. Dentro

de este paquete Clases del proyecto existen dos paquetes más, apps y web donde se encuentran principalmente las clases necesarias para el desarrollo de la aplicación. Dentro de cada uno de estos paquetes existe un paquete para cada módulo para organizar las clases correspondientes a cada módulo, ya sean clases de presentación, de negocio o clase de datos. Dentro del paquete web están todas las clases de presentación o sea las clases del tipo js, y dentro del paquete apps están todas las clases del negocio y las clases de datos. Dentro del paquete apps, en el paquete Controllers están las clases controladoras. Luego paralelo al paquete Controllers está el paquete Models quien tiene todas las clases que se generan por cada tabla de la base de datos. Se puede encontrar otro paquete llamado Estereotipos donde tendremos todos los estereotipos usados para el modelado (Rodríguez Urrutia, 2010).

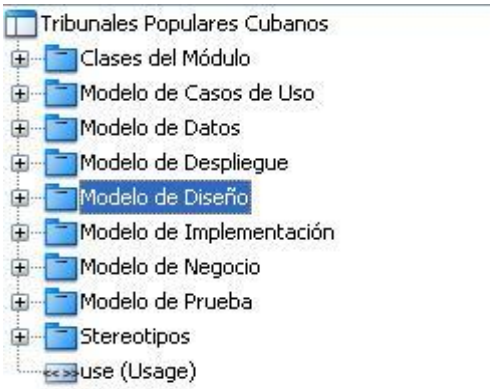


Fig. 9 Estructura organizativa general en Visual Parading

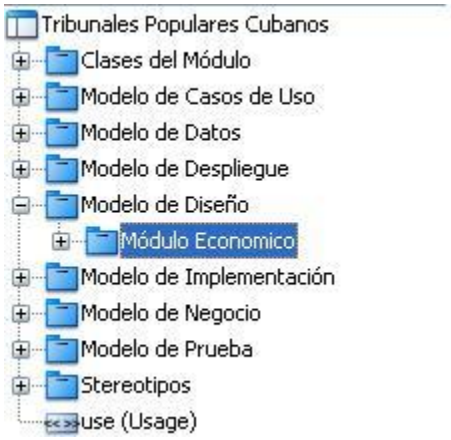


Fig. 10 Estructura del Modelo de Diseño

2.4.1.1. Clases del Módulo

En esta carpeta se encontrarán todas las clases del módulo. Dentro del paquete Clases del Módulo estarán los paquetes apps y web (Rodríguez Urrutia, 2010).

2.4.1.2. Paquete web

Dentro de cada módulo habrá un paquete llamado “js” donde estarán todas las clases de tipo js (Rodríguez Urrutia, 2010) (Figura 11).

Fig. 11 Estructura de las clases del proyecto en Visual Parading

2.4.1.3. Paquete apps

Por otra parte el paquete apps tiene internamente dos paquetes, Controllers y Models. En el paquete Controllers están todas las clases controladoras y en el paquete Models se encuentran las clases que se generaran por cada una de las tablas de la base de datos. Están además los paquetes Views donde se encuentran todas las vistas del subsistema, o sea las clases del tipo phtml. Se puede encontrar además otro paquete llamado Validators donde están todas las clases que llevan a cabo las funciones de validación del subsistema (Rodríguez Urrutia, 2010) (Figura 12).

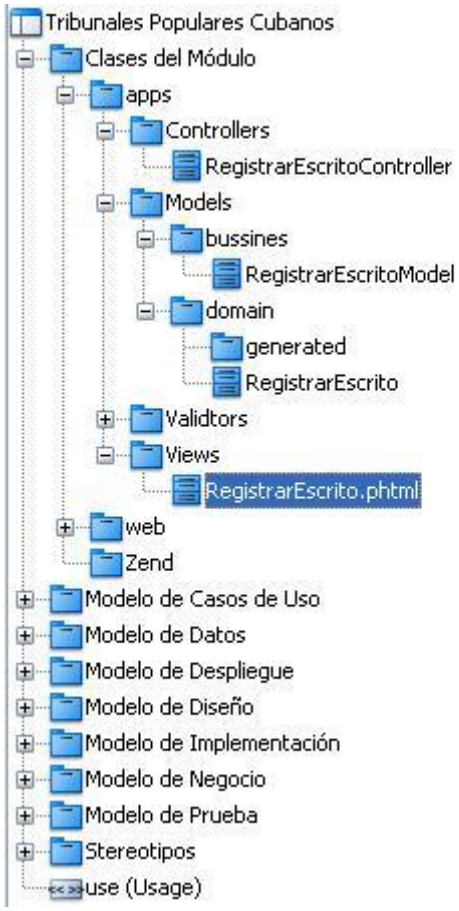


Fig. 12 Estructura de las clases incluidas dentro del paquete apps

Las clases de presentación, o sea las clases del tipo js se crean en Visual Paradigm dentro del módulo que le corresponda. Estas clases no se pueden exportar ni importar. Las clases controladoras también se crean en Visual Paradigm dentro del módulo que le pertenece. Es importante aclarar que se tiene una clase controladora por cada caso de uso. Estas clases pueden ser exportadas generando gran parte del código a utilizar. Las clases que se encuentran dentro del paquete Models se importan desde el framework Sauxe. Mediante el ORM Doctrine por cada tabla de la base de datos se generarán tres clases en el directorio \apps\Nombre_Módulo\models. Dentro de este directorio en bussines se encuentran las clases con sufijo model en las que se realizan las operaciones de Crear, Eliminar y Modificar. Dentro del mismo directorio en la carpeta domain se

encuentran las clases sobre las que se realizarán las consultas a la base de datos. Las dos clases mencionadas anteriormente heredan de una clase con el prefijo base que se encuentra en domain en una carpeta denominada generated.

2.4.2. Modelo de Diseño

El modelo de diseño, es un modelo de objetos que describe la realización física de los casos de uso, centrándose en como los requisitos funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Además el modelo de diseño sirve de abstracción de la implementación del sistema y es de ese modo utilizado como una entrada fundamental de las actividades de implementación. Los casos de uso son realizados por las clases de diseño y sus objetos. Esto se representa por colaboraciones en el modelo de diseño y denota realización de caso de uso-diseño (Jacobson, et al., 2010).

2.4.2.1. Estrategia trazada para el diseño

Para la realización de la solución propuesta en el proyecto TPC implementó una estrategia para el diseño, basada en los patrones arquitectónicos y de diseño propuestos para el desarrollo. A continuación se exponen los pasos a seguir en dicha estrategia (Rodríguez Urrutia, 2010).

- Confección del modelo de datos para la solución informática para el subsistema económico.
- Realización de los diagramas de clases correspondientes a cada caso de uso que se implementa, unido a los diagramas de secuencia correspondientes para cada uno de los escenarios que se necesiten.
- Generación de código de las clases del dominio, y los gestores del negocio y de acceso a datos.
- Capacitación al programador de los diagramas realizados para cada caso de uso.

2.4.2.2. Diagrama de Clases

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

Caso de uso Registrar Escrito Promocional

Caso de Uso:	Registrar escrito promocional
Actores:	Registrador de escrito (inicia)
Resumen:	El caso de uso se inicia cuando el registrador de escrito accede al sistema para registrar los datos del escrito promocional. Para ello introduce en el sistema los datos asociados al escrito promocional, el sistema registra el escrito, lo radica, lo turna e informa al ponente del nuevo escrito presentado; de este modo termina el caso de uso.
Precondiciones:	El usuario debe estar autenticado correctamente en el sistema.
Referencias	RF.04, RF.5, RF.6, RF.08
Reglas del negocio	
Prioridad	Crítico

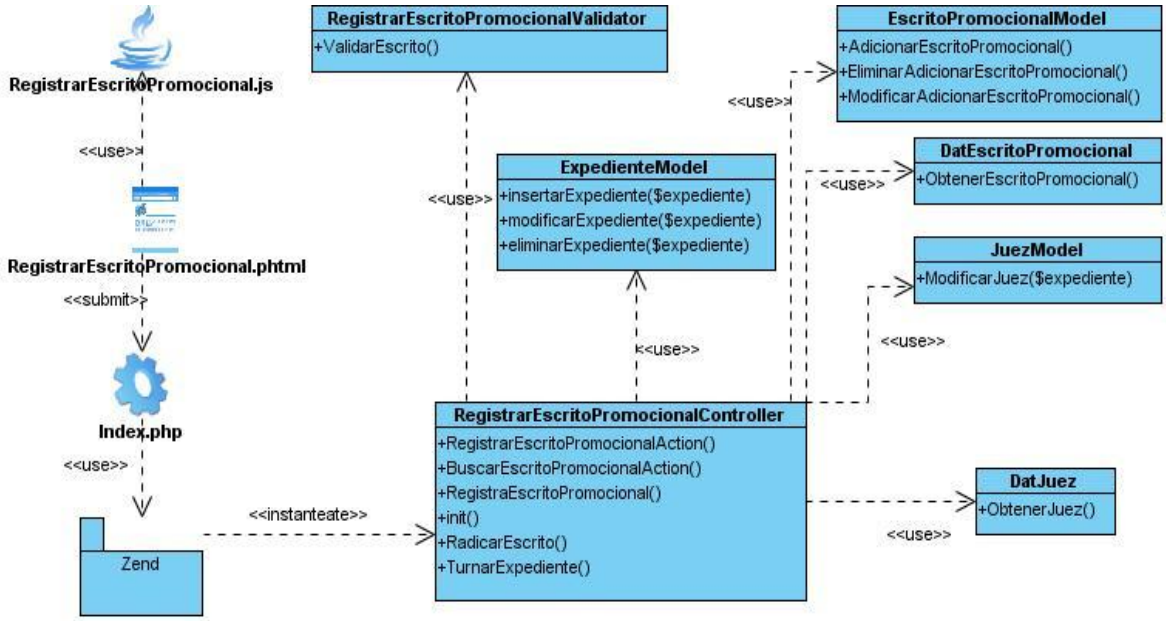


Fig. 13 Diagrama de clases del caso de uso Registrar Escrito Promocional

2.4.2.3. Diagrama de Interacción

Estos diagramas se pueden expresar en diagramas de secuencia y en diagramas de colaboración. Los primeros muestran las interacciones entre objetos, ordenadas en secuencia temporal durante un escenario concreto y son los que encontrará en el Modelo de Diseño, los segundos destacan la organización de los objetos que participan en una interacción (Rodríguez Urrutia, 2010).

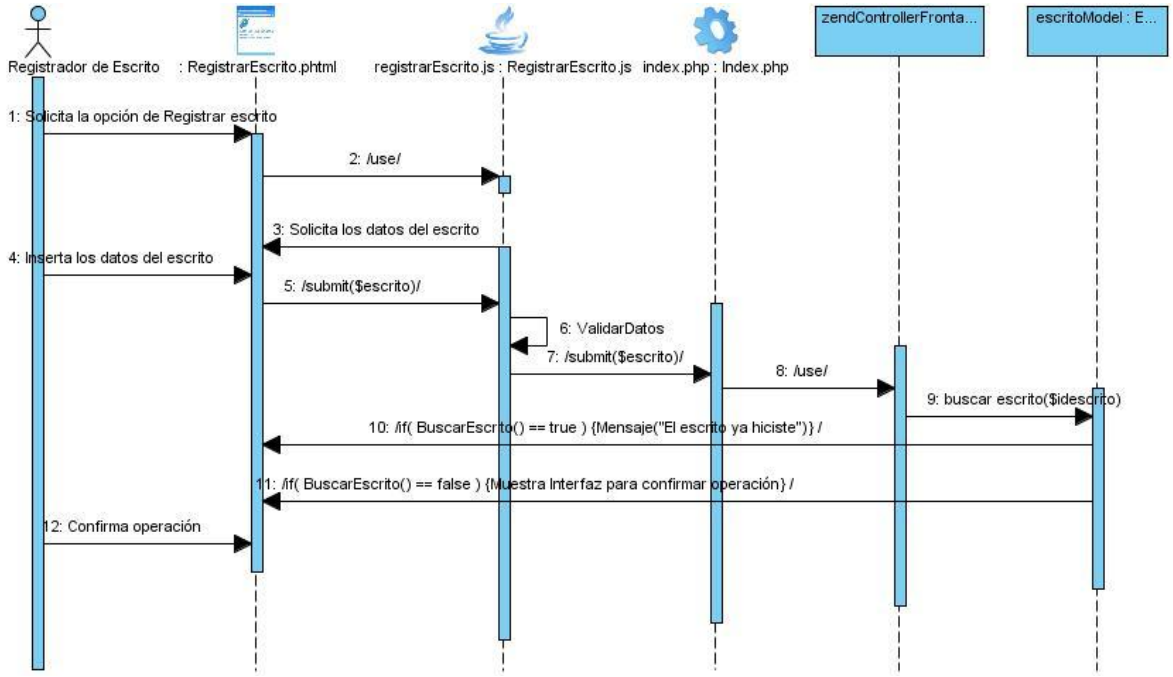


Fig. 14 Diagrama de secuencia del caso de uso Registrar Escrito

2.4.3. Modelo de Implementación.

El modelo de implementación está conformado por los **diagramas de despliegue** y **componentes,** que son artefactos generados durante el flujo de trabajo de implementación ya que es en este flujo de trabajo donde se describe cómo los elementos del modelo del diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue.

2.4.3.1. Diagrama de componentes

Los **diagramas de componentes** son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre sus elementos. El uso más importante de estos diagramas es mostrar la estructura de alto nivel del modelo, especificando los subsistemas de implementación y sus dependencias.

A continuación se muestra el Diagrama de Componentes para el sistema. (Figura 15).

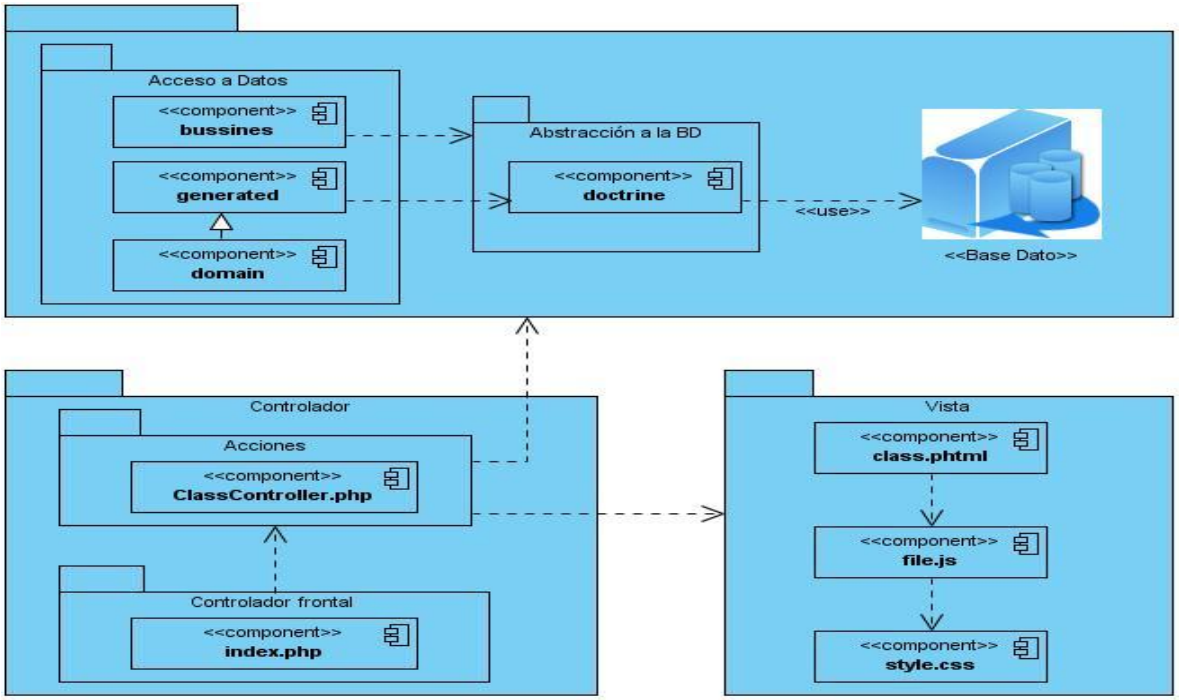


Fig. 15 Diagrama de Componentes para el sistema

2.4.3.2. 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.

Podemos observar lo siguiente sobre el modelo de despliegue:

- Cada nodo representa un recurso de cómputo, normalmente un procesador o un dispositivo de hardware similar.
- Los nodos poseen relaciones que representan medios de comunicación entre ellos, tales como Internet, intranet, bus y similares.
- "El modelo de despliegue puede describir diferentes configuraciones de red, incluidas las configuraciones para pruebas y para simulación" (Payá, 2008).

El propósito del modelo de despliegue es capturar la configuración de los elementos de procesamiento y las conexiones entre estos elementos en el sistema. Permite el mapeo de procesos dentro de los nodos, asegurando la distribución del comportamiento a través de aquellos nodos que son representados.

A continuación se muestra el Diagrama de Despliegue que se propone para el sistema. (Figura 16).

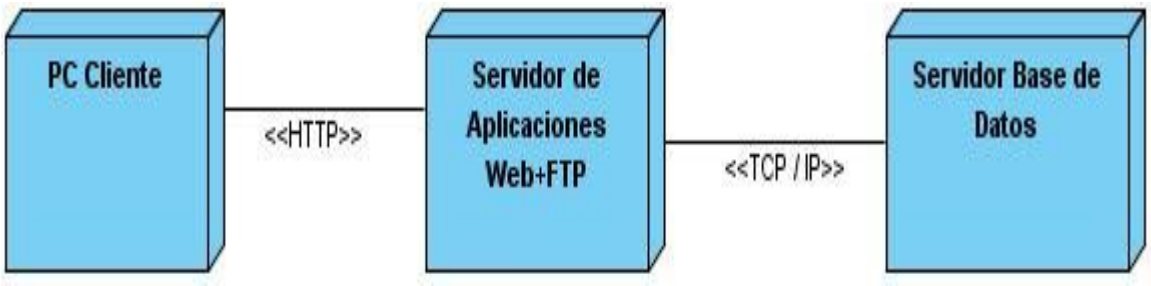


Fig. 16 Diagrama de Despliegue del sistema

2.4.3.3. Descripción de los nodos físicos

Un nodo es "un elemento físico que existe en tiempo de ejecución y que representa un recurso computacional, que en general tiene al menos una memoria y a menudo capacidad de procesamiento" (Payá, 2008).
Los nodos que componen el modelo de despliegue correspondiente a la aplicación son los siguientes:

- **Nodo PC – Cliente:** se encontrará el sistema operativo Windows o Linux y los navegadores web Mozilla Firefox o Internet Explorer mediante los cuales los clientes tendrán acceso al sistema.
- **Nodo Servidor Web Apache:** se encontrará todo lo relacionado con la aplicación. Estarán agrupados los archivos a través de los cuales el usuario logra acceder al sistema, además se encuentra contenida toda la información específica de cada registro, sus clases; así como almacena además la configuración general del sistema, las clases y librerías externas y los plugins de instalación de la aplicación.
- **Nodo Servidor BD PostgreSQL:** serán almacenados los datos de la aplicación, se guardará el modelo de objetos del sistema.

2.4.4. Interfaces del sistema

Para la realización de la aplicación se desarrollaron interfaces amigables y seguras, acorde con el personal que trabajará con la misma. A continuación se presenta la interfaz principal de de entrada al sistema. En esta se encuentran los vínculos a todas las funciones con que debe cumplir la misma. (Figura 17).

Fig. 17 Pantalla inicial del Juez

Conclusiones Parciales

Los artefactos obtenidos en este capítulo son muy necesarios y de vital importancia para la construcción del sistema ya que posibilitan definirlo con suficiente detalle como para permitir su interpretación y realización física. En particular:

- El Modelo del diseño permitió materializar con precisión los requerimientos del cliente y sirvió como guía para las actividades de implementación, al producir una representación técnica del software que será desarrollado.
- El Modelo de implementación permitió obtener una visión general de los componentes y subsistemas a implementar, facilitando que el proceso de implementación se realizara de forma más simple y con una mayor calidad.

CAPÍTULO 3: ANÁLISIS DE RESULTADOS

En este capítulo se hace un análisis de los resultados obtenidos durante el desarrollo de la solución propuesta con la utilización de métricas de diseño TOC y RC, y pruebas de implementación de caja negra.

3.1. Métricas de software

Las métricas aplicadas al diseño para la evaluación de la solución propuesta son las siguientes:

Tamaño operacional de clase (TOC): Está dado por el número de métodos asignados a una clase.

Atributo que afecta	Modo en que lo afecta
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución del grado de reutilización de la clase.

Tabla 1. Tamaño operacional de la clase (TOC)

Relaciones entre clases (RC): Está dado por el número de relaciones de uso de una clase con otras.

Atributo que afecta	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del Acoplamiento de la clase.
Complejidad de mantenimiento	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.

Reutilización	Un aumento del RC implica una disminución en el grado de reutilización de la clase.

Tabla 2. Relaciones entre clases (RC)

3.1.1. Resultado del instrumento de evaluación de la métrica Tamaño operacional de la clase (TOC)

La gráfica siguiente muestra los resultados de la métrica Tamaño operacional de clase (TOC), donde en la gráfica cantidad de operaciones, se mide el porciento de cantidad de procedimientos que contiene una clase. Como puede apreciar existe casi un 63% de clases que contienen menos de 6 procedimientos y casi un 32% que tienen menos de 10 procedimientos lo que significa un valor aceptable y satisfactorio para la métrica Instrumentos y tabla de resultados en (Anexo 1: Instrumento de medición de la métrica Tamaño operacional de clase (TOC)).

Promedio de procedimientos por clase= 5.
Responsabilidad: Baja <=5, 5 < Media < 10, Alta > 10.
Complejidad: Baja <=5, 5 < Media < 10, Alta > 10.
Reutilización: Baja > 10, 5 < Media < 10, Alta <= 5.

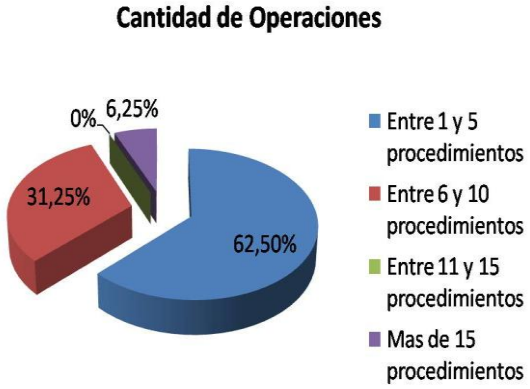


Fig. 18 Representación de los resultados obtenidos en el instrumento, agrupados en los intervalos definidos

A continuación se muestra la gráfica que mide el atributo Responsabilidad de la métrica, obteniéndose resultados satisfactorios mostrando una responsabilidad baja con valor casi del 63%.

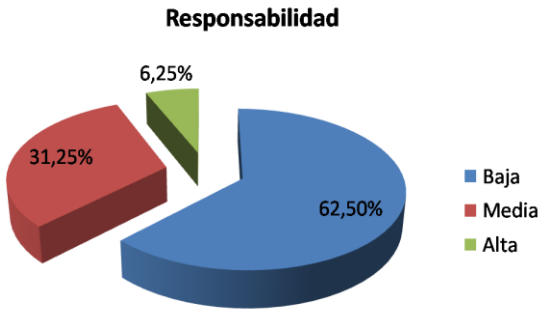


Fig. 19 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad

El atributo que se mide en la gráfica que se muestra a continuación, después de haberse realizado la medición de la métrica, arrojó también resultados positivos ya que la complejidad de las clases es baja en un 62.5%.

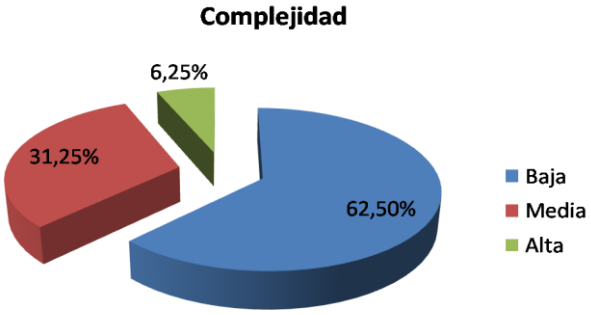


Fig. 20 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad

Al igual que el atributo complejidad la reutilización del código presenta resultados satisfactorios al brindar valores por encima del 60% en una reutilización alta.

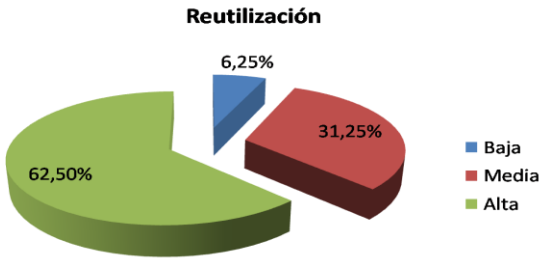


Fig. 21 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización

3.1.2. Resultado del instrumento de evaluación de la métrica Relaciones entre Clases (RC)

La gráfica siguiente muestra el resultado de la métrica Relación entre Clases (RC), donde se mide el porciento de cantidad de dependencias que contiene una clase. Como se puede apreciar existe un 50% de clases que no contienen dependencias y no alcanza el 40 % del total de las clases que presentan solo una dependencia, lo que significa un valor aceptable y satisfactorio para esta métrica. Instrumentos y tabla de resultados en (Anexo 2: Instrumento de medición de la métrica Relaciones entre Clases (RC)).

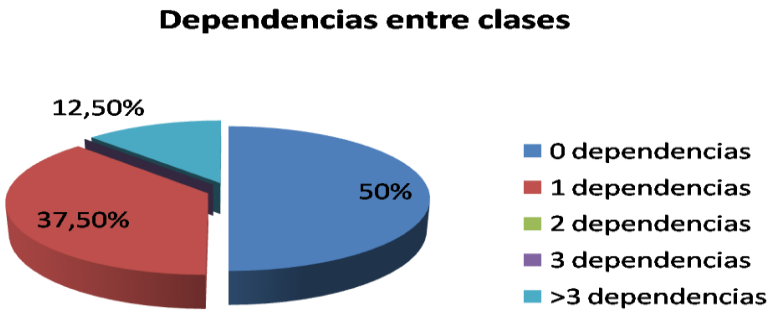
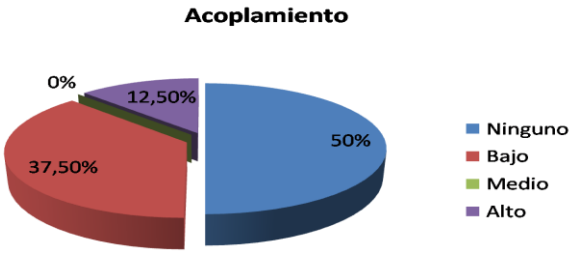


Fig. 22 Representación de los resultados obtenidos en el instrumento, agrupados en los intervalos definidos

El siguiente gráfico muestra el resultado del atributo acoplamiento entre las clases del sistema el cual presenta un 50% con valor ninguno, lo que quiere decir que las clases presentan un bajo acoplamiento, lo que significa que la medición de la métrica está dando frutos satisfactorios.



.

Fig. 23 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento

Por otra parte los resultados del atributo complejidad de mantenimiento son bastante satisfactorios, con una baja complejidad de 87.5%, lo que significa fácil soporte.

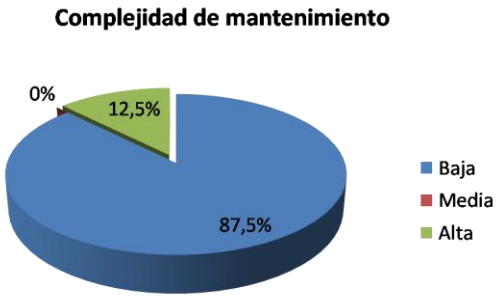


Fig. 24 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento

Luego de haber analizado varios parámetros, se llevo a cabo la medición de la variable cantidad de pruebas, el que brindo resultados satisfactorios al presentar un mas del 85% de bajo esfuerzo a la hora de realizar pruebas al diseño.

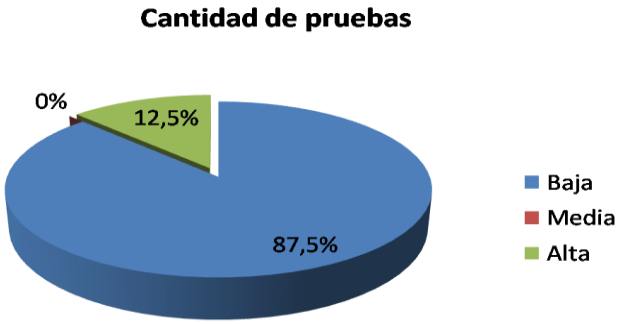


Fig. 25 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas

Finalmente se realizó la medición del atributo reutilización, para comprobar el nivel de reutilización del diseño de clases, la métrica generó resultados positivos al informar que el diseño de la solución posee más del 85% de reutilización de sus clases.

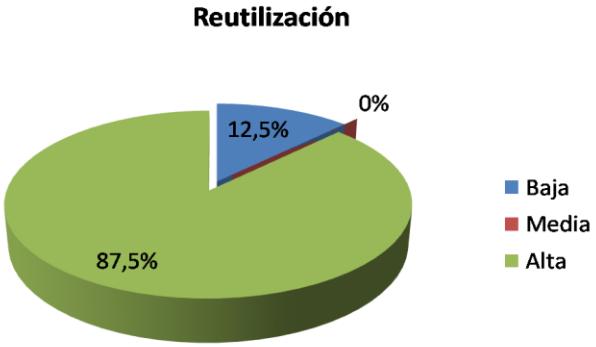


Fig. 26 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización

En conclusión, la evaluación de los resultados obtenidos durante la aplicación del instrumento de medición de la métrica Relación entre Clases y Tamaño Operacional de la Clase, demuestran el uso de un buen diseño de software.

3.2. Pruebas

Las pruebas verifican que el producto funcione como se diseñó y que los requerimientos son satisfechos cabalmente, además de brindar soporte para encontrar, documentar y solucionar defectos del sistema.

El principal objetivo de las pruebas es evaluar la calidad del producto que se está desarrollando, pues está presente durante todo el ciclo de desarrollo del sistema lo que posibilita que se vaya refinando constantemente y no al final. El papel de las pruebas no es asegurar la calidad, pero sí evaluarla, y proporcionar una realimentación a tiempo, de forma que los aspectos de calidad puedan resolverse de manera efectiva en tiempo y costo.

3.2.1. Pruebas de Caja Negra

Las pruebas de Caja Negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, o sea los casos de pruebas pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto.

Permite obtener conjuntos de condiciones de entrada que preparen completamente todos los requisitos funcionales de un programa. Intenta encontrar errores de las siguientes categorías: funciones incorrectas o ausentes; errores de interfaz, en estructuras de datos o en acceso a bases de datos externas; errores de rendimiento, de inicialización y de terminación.

A continuación se muestran resultados obtenidos durante la realización de las pruebas de caja negra. Estas arrojaron resultados satisfactorios a los efectos de la implementación del sistema.

Caso de prueba de Caja Negra para validar el requisito Registrar Escrito Promocional.

Descripción General

El caso de uso Registrar Escrito Promocional se inicia cuando el registrador de escrito accede al sistema para registrar los datos del escrito promocional. Para ello introduce en el sistema los datos asociados al escrito promocional, el sistema registra el escrito, lo

radica, lo turna e informa al ponente del nuevo escrito presentado; de este modo termina el caso de uso.

Condiciones de Ejecución

El usuario debe estar autenticado correctamente en el sistema.

Secciones a probar en el Caso de Uso.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC1:Registrar Escrito Promocional	EC1.1 Registrar Escrito Promocional	Valida que los datos introducidos son correctos, que los campos obligatorios no están vacíos y guarda los datos.
	EC 1.2: Insertar datos incorrectos y/o campos vacíos.	Informa que los datos son incorrectos y no guarda los cambios
	EC 1.3: Cancelar operación.	Cierra la interfaz y no guarda los cambios.
	EC 1.4: Visualizar escrito	Muestra el contenido del escrito
	EC 1.5: Enviar	1. Radica el escrito. 2. Turna el expediente. 3. Informa al ponente del nuevo escrito registrado.
SC2:Registrar Escrito Promocional por secretaria	EC 1.1 Registrar Escrito Promocional por secretaria	Valida que los datos introducidos son correctos, que los campos obligatorios no están vacíos y guarda los datos.

	EC 1.2: Cancelar operación.	Cierra la interfaz y no guarda los cambios.
	EC 1.5: Aceptar	1. Radica el escrito. 2. Turna el expediente. 3. Informa al ponente del nuevo escrito registrado.

Tabla 3. Secciones de prueba en el caso de uso.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Grupo empresarial	Combo Box	No	Se selecciona el grupo empresarial al que pertenece
2	Causal	Combo Box	No	Se selecciona el causal
3	Tipo de persona(Promovente)	Combo Box	No	Se selecciona el tipo de persona
4	Tipo de persona(Presunto deudor)	Combo Box	No	Se selecciona el tipo de persona
5	Datos de la deuda	Gridpanel	No	Se adicionan los datos de la deuda

6	Hechos	Campo de texto	No	Se describen como ocurrieron los hechos
7	Fundamentos de Derecho	Campo de texto	No	Se describen los fundamentos de derecho
8	Documentos que acompañan (Formato Duro)	Adjuntar	Si	Se escribe el nombre del documento en formato duro
9	Documentos que acompañan(Formato Digital)	Adjuntar	Si	Se adjunta un documento

Tabla 4. Descripción de variable

Conclusiones Parciales

En este capítulo se aplicaron métricas dirigidas a evaluar la calidad del diseño del software, basadas en diferentes atributos de calidad que avalan sus resultados. Con la métrica Tamaño Operacional de Clase se evidencia que la implementación no es complicada, que la mayoría de las clases tienen una baja responsabilidad, además de tener una reutilización alta. Por su parte la de Relaciones entre Clases demuestra que la implementación del sistema tiene una calidad aceptable donde la mayor parte de las

clases tienen una o ninguna dependencia, en cuanto al nivel de acoplamiento se comporta bajo por lo que es un factor aceptable desde el punto de vista de implementación del software. También se puede mencionar que la cantidad de pruebas y la complejidad para el mantenimiento se mantienen en niveles bajos según la métrica empleada, por lo que se demuestra el uso de un buen diseño de software. Además, se realizaron un conjunto de pruebas al sistema con el fin de obtener un producto con la mayor calidad posible. Es importante destacar además que el caso de prueba presentado no es el único diseñado para la aplicación sino que el objetivo es plasmar la forma en que se realizaron los mismos.

Conclusiones Generales

Durante el desarrollo del trabajo de diploma y el tiempo dedicado a la investigación, diseño, implementación y prueba de la solución propuesta, se adquirieron conocimientos necesarios para el desarrollo del sistema y fundamentales para la formación profesional del autor. Luego de la realización de las actividades anteriores:

- Se desarrolló el Sistema de gestión del proceso Diligencias Previas para el proyecto TPC que contribuye con el mejoramiento de la gestión de la documentación oficial que transita por los Tribunales Populares Cubanos, dándose cumplimiento al objetivo propuesto.
- El estudio realizado sobre metodologías de desarrollo de software, lenguajes de programación y herramientas, teniendo presente las necesidades de los clientes y usuarios finales posibilitó justificar la selección de las utilizadas en este trabajo.
- La construcción del Modelo del diseño permitió generar los elementos necesarios para desarrollar la implementación del sistema, al concebir una representación técnica del mismo.
- La validación del sistema mediante la aplicación de las pruebas de caja negra arrojó resultados satisfactorios, demostrándose la fiabilidad del *software* desarrollado.

Recomendaciones

Luego de haber concluido el trabajo se recomienda:

- Integrar los reportes de Diligencias Previas al subsistema una vez que estos se hayan concluido.
- Integrar el proceso de Diligencias Previas con los demás procesos del módulo.

- Realizar encuestas periódicas sobre la efectividad del sistema, el grado de satisfacción de los usuarios, así como funcionalidades que se le puede incorporar. Todo esto previendo un futuro perfeccionamiento del sistema.

Referencias bibliográficas

ALARCÓN DE QUESADA, RICARDO. 1998. *Ley de los Tribunales Populares.* Habana : s.n., 1998.

Bello. 2006. Diligencias Previas . *Diligencias Previas al Proceso Ejecutivo.* 2006.

Canales Mora, Roberto. 2004. AdictosAlTrabajo.com. [Online] Febrero 2, 2004. [Cited: Abril 7, 2011.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=vp.

Caride, Javier. 2008. Desarrollo de webs y aplicaciones. [Online] Diciembre 3, 2008. [Cited: Mayo 20, 2011.] http://www.desarrollosweb.net/2008/12/03/cookbook-extjs-cargar-un-registro-de-un-gridpanel-en-un-formpanel/.

case-tools. 2011. *Herramientas CASE .* 2011.

Castro Morell, MSc. Daniel E. 2008. *Sistema para la tramitación de.* Villa Clara : s.n., 2008.

CEGEL. 2009. *Informe del proyect TPC.* Habana : s.n., 2009.

Desarrollo web. 2007. DesarrolloWeb.com . [Online] 2007. [Cited: Marzo 12, 2011.] www.desarrolloweb.com/php/ .

Desarrolloweb. 2002. Lenguajes de lado servidor. [Online] Enero 1, 2002. [Cited: Noviembre 25, 2010.] www.desarrolloweb.com.

DesarrolloWeb. 2007. Sistemas gestores de bases de datos. [Online] Julio 2007, 2007. [Cited: Noviembre 26, 2010.] www.desarrolloweb.com/

doctrine-project. 2008. *Doctrine: ORM Open Source para PHP 5.2.* 2008.

Enterprise Architect . 2009. *Información General de Enterprise Architect.* 2009.

ER/Studio Enterprise . 2008. *Embarcadero Lanza ER/Studio Enterprise .* 2008.

EUROPA PRESS. 2009. El sistema de gestión procesal Atlante II. [Online] Junio 8, 2009. [Cited:Noviembre 20, 2010.] www.europapress.es/islas-canarias/noticia-sistema-gestion-procesal-atlante-ii

ExtJS. 2008. Desarrollo en Web. [Online] Octubre 22, 2008. [Cited: Enero 20, 2011.] http://desarrolloweb/2008/10/extjs/.

Figueras Rodríguez, Marlen. 2010. *Requisitos Funcionales para el módulo Diligencias Previas del subsistema Económico del proyecto Tribunales Populares Cubanos.* Habana : s.n., 2010.

Flanagan, David. 2002. *The Definitive Guide .* 2002. 4ta Edición.

Garlan, David and Shaw, Mary. 1994. *An introduction to Software Architecture.* Carnegie Mellon : s.n., 1994.

Gómez Baryolo, Oiner. 2010. *Marco de Trabajo SAUXE.* Habana : s.n., 2010.

Gracia, Joaquin. 2005. *Patrones de diseño.* 2005.

Guerra, Prof: Lic. Roberto. 2010. Scribd Inc. [Online] Mayo 20, 2010. [Cited: Noviembre 20, 2010.] http://es.scribd.com/doc/31440864/Metodologia-RUP.

IBM. 2008. *IBM - Rational Rose Enterprise.* 2008.

IEEE. 1993. *Software Engineering.* s.l. : IEEE Standars Collection, 1993.

Jacobson, Ivar and y otros. 1992. Object-Oriented Software Engineering—A Use Case. [book auth.] Addison-Wesley. *A Use Case Driven Approach.* Wokingham : s.n., 1992.

JOFRÉ ROJAS, ENRIQUE. 2002. *Modelo de Diseño.* Chile : s.n., 2002.

Lawebdelprogramador. 2011. PostgreSQL PE | La base de datos libre mas avanzada del mundo. [Online] Enero 15, 2011. [Cited: Marzo 21, 2011.] www.lawebdelprogramador.com.

Martines, Msc. Raysa de la Caridad. 2000. El proceso ejecutivo. Camaguey : s.n., 2000.

Martínez Reyes, Raysa de la Caridad. 2008. *El proceso ejecutivo: incidencia de la letra de cambio y su práctica judicial en las salas de lo económico.* Las Tunas : s.n., 2008.

Microsoft. 2008. ¿Qué es la Arquitectura de Software? [Online] Mayo 12, 2008. [Cited: Diciembre 10, 2010.] www.arqsoft.com.

—. 2008. *Estilos en Arquitectura de Software.* 2008.

Montané Izaguirre, Ing Juan Carlos. 2010. *Plan de Desarrollo de Software.* Habana : s.n., 2010.

Muñóz, Dra.Coral. 2008. *MÉTRICAS DE LA CALIDAD DEL SOFTWARE.* La Mancha : s.n., 2008.

Muñoz, P. 2010. La Tribuna_digital. 2010.

Navarro, Maikel. 2009. *Documento de Arquitectura de Software.* Habana : s.n., 2009.

netbeans.org. 2008. *NetBeans un Entorno de Desarrollo Integrado.* 2008.

Núñez. 2006. III ENCUENTRO INTERNACIONAL.JUSTICIA Y DERECHO . LAS DILIGENCIAS PREVIAS AL PROCESO EJECUTIVO. [book auth.] Lic. Georgina Carmenate. *UNA VISIÓN DE SU TRAMITACIÓN.* Las Tunas : s.n., 2006.

País Vasco. 2008. El control electrónico en el sistema penal. [Online] Febrero 10, 2008. [Cited: Noviembre 15, 2010.] www.tesisenred.net/.

Payá, Martín. 2008. Módulo 2. Tema 12: Modelo de Implementación. [Online] Enero 20, 2008.[Cited:Mayo 13, 2011.] http://www.dsi.uclm.es/asignaturas/42530/pdf/M2tema12.pdf.

Pelaez, Carlos. 2002. *Implementación de Proyectos de Informática Jurídica en el Poder Judicial Boliviano.* Bolivia : s.n., 2002.

Riestra, Emma. 1996. DERECHO E INFORMÁTICA EN MÉXICO. [book auth.] Juan José Ríos Estavillo. *INFORMÁTICA JURÍDICA Y DERECHO DE LA INFORMÁTICA.* 1996.

Rodríguez Urrutia, Misael. 2010. *Manual de diseño proyecto TPC.* Habana : s.n., 2010.

Rosenberg, Stapko and Gallo. 1999. MANUAL DE ESTILO DE PROGRAMACIÓN. [Online] Enero 24, 1999. [Cited: Febrero 18, 2011.] www.ahristov.com/taller/blog/Estilo-de-Programacion-9-01.pdf - .

Rumbaugh, et al. 2000. El Proceso Unificado de Desarrollo de Software. [book auth.] Addison Wesley. 2000.

RUP. 2010. La tecla de escape. [Online] Octubre 17, 2010. [Cited: Noviembre 25, 2010.] http://latecladeescape.com/ingenieria-del-software/metodologias-de-desarrollo-del-software.html.

Saavedra López, Esteban. 2008. Frameworks para desarrollo de aplicaciones Web. [Online]Noviembre1,2008.[Cited:Diciembre3,2010.] www.slideshare.net/estebansaavedra/frameworks-para-desarrollo-de-aplicaciones-web-presentation.

—. 2008. Frameworks para desarrollo de aplicaciones Web. [Online] Septiembre 12, 2008. [Cited: Frebrero 10, 2011.] www.opensourceworldconference.com.

Scrum. 2009. proyectosagiles. [Online] 2009. [Cited: Diciembre 12, 2010.] http://www.proyectosagiles.org/que-es-scrum.

Universidad de la Habana. 2002. *SisEco.* Habana : s.n., 2002.

Visual Parading . 2004. *Visual Paradigm for UML.* 2004.

Anexos

Anexo 1: **Instrumento de medición de la métrica Tamaño operacional de clase (TOC)**.

Atributos	Categoría	Criterio
Responsabilidad	Baja	< =Prom
	Media	Entre Prom. y 2* Prom.
	Alta	> 2* Prom.
Complejidad de Implementación	Baja	< =Prom
	Media	Entre Prom. y 2* Prom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2* Prom.
	Media	Entre Prom. y 2* Prom.
	Alta	< =Prom

Tabla 5. Rango de valores para los criterios de evaluación de la métrica TOC.

Clases	Procedimientos	Responsabilidad	Complejidad	Reutilización
RegistrarEscritoPromocionalController	21	Alta	Alta	Baja
DpersonanaturalExt	1	Baja	Baja	Alta
DpersonajuridicaExt	1	Baja	Baja	Alta
RegistrarescritoController	8	Media	Media	Media
NtipoescritoExt	2	Baja	Baja	Alta
NescritoExt	2	Baja	Baja	Alta
NformacomparecerExt	2	Baja	Baja	Alta
DpersonaExt	5	Baja	Baja	Alta
NpersonaExt	5	Baja	Baja	Alta
CrearactacomparecenciaController	6	Media	Media	Media
DexpedienteExt	9	Media	Media	Media
NprocedimientoExt	2	Baja	Baja	Alta

ModificarcomposiciontribunalController	4	Baja	Baja	Alta
CrearProvidenciaDeASController	7	Media	Media	Media
RegistrarresultaController	7	Media	Media	Media
ResolviendofondodelasuntoController	4	Baja	Baja	Alta

Tabla 6. (Resultados de la evaluación de la métrica TOC y su influencia en los atributos de calidad (Responsabilidad, Complejidad de Implementación y Reutilización))

Anexo 2: Instrumento de medición de la métrica Relación entre Clases (RC).

Atributos	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
Complejidad de Mantenimiento	Baja	<= Prom.
	Media	Entre Promedio y 2*Prom.
	Alta	> 2*Prom.
Reutilización	Baja	>2* Prom.
	Media	Entre Promedio y 2*Prom.
	Alta	<= Prom.
Cantidad de Pruebas	Baja	<= Prom.
	Media	Entre Promedio y 2*Prom.
	Alta	> 2*Prom.

Tabla 7. Rango de valores para los criterios de evaluación de la métrica RC.

Clases	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad Mant.	Reutilización	Cantidad de Pruebas
RegistrarEscritoPromocionalController	0	Ninguno	Baja	Alta	Baja
DpersonanaturalExt	1	Bajo	Baja	Alta	Baja
DpersonajuridicaExt	1	Bajo	Baja	Alta	Baja
RegistrarescritoController	0	Ninguno	Baja	Alta	Baja
NtipoescritoExt	1	Bajo	Baja	Alta	Baja

NescritoExt	1	Bajo	Baja	Alta	Baja
NformacomparecerExt	1	Bajo	Baja	Alta	Baja
DpersonaExt	0	Ninguno	Baja	Alta	Baja
NpersonaExt	1	Bajo	Baja	Alta	Baja
CrearactacomparecenciaController	0	Bajo	Baja	Alta	Baja
DexpedienteExt	6	Alto	Alta	Baja	Alta
NprocedimientoExt	4	Alto	Alta	Baja	Alta
ModificarcomposiciontribunalController	0	Ninguno	Baja	Alta	Baja
CrearProvidenciaDeASController	0	Ninguno	Baja	Alta	Baja
RegistrarresultaController	0	Ninguno	Baja	Alta	Baja
ResolviendofondodelasuntoController	0	Ninguno	Baja	Alta	Baja

Tabla 8. (Resultados de la evaluación de la métrica RC y su influencia en los atributos de calidad (Acoplamiento, Complejidad de Mantenimiento, Reutilización y Cantidad de Pruebas))