Universidad de las Ciencias Informáticas

Facultad 3

Diseño e Implementación del Módulo Certificación de Antecedentes Penales de la Solución Tecnológica Integral para la Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor/es: Lisday Méndez Velázquez

José Alejandro Castillo Figueroa

Tutor: Ing. Yanelis Vega García

Ciudad de la Habana, Cuba

Junio, 2011

Declaración de Autoría

DECLARACIÓN DE AUTORÍA

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

Para que así conste firmo la presente a los ____ días del mes de Junio del año 2011

Lisday Méndez Velázquez
_____

José Alejandro Castillo Figueroa
_____

Yanelis Vega García
_____
Firma del tutor

Resumen

La gestión y el control de la información manejada por las empresas, constituye uno de los mayores problemas en el mundo actual; la información, unido al conocimiento de los especialistas, son factores que influyen principalmente en el desarrollo productivo de las mismas. Por tanto la eficiencia y calidad con que se tenga acceso a la información actualizada constituye un pilar fundamental en el propio desarrollo de las organizaciones.

La División de Antecedentes Penales constituye una institución única de su tipo dentro del territorio venezolano. Su principal objetivo es gestionar los antecedentes penales de los ciudadanos venezolanos.

Este trabajo se basa en las actividades realizadas por los autores en los roles de diseñador e implementador como integrantes del proyecto Sistema para la Gestión de Antecedentes Penales (DAP). En él se muestra el diseño y la implementación de la solución brindada para el módulo de Certificaciones de Antecedentes Penales, a partir de los requisitos definidos anteriormente por los analistas.

PALABRAS CLAVE

Antecedentes Penales, Certificaciones, Gestión, Sistema

Índice de Tablas

Índice de Tablas

Índice de Tablas

Introducción

Introducción

Las Tecnologías de la Información y las Comunicaciones (TIC) ocupan un lugar fundamental en el desarrollo de la sociedad y la economía mundial. Sin ellas la sociedad no se estuviera desarrollando de manera tan vertiginosa. El concepto de TIC nace con la convergencia tecnológica de la electrónica, el software y las infraestructuras de las telecomunicaciones y proveen herramientas que ofrecen la posibilidad de encontrar soluciones novedosas ante los desafíos que se impone el hombre en su bregar diario. Cuba no se ha quedado detrás en este tema pues a pesar de ser un país subdesarrollado y con escasos recursos, se ha propuesto desarrollar esta rama con el objetivo de llevar la informática a todos los rincones de la isla y de la misma forma contribuir al desarrollo de otras sociedades mediante acuerdos de colaboración, como es el caso de la República Bolivariana de Venezuela.

Con el triunfo del Comandante Hugo Rafael Chávez Frías se abrieron nuevos horizontes para este país, estableciéndose un proceso de transformación basado en los principios de igualdad y justicia, con el objetivo de lograr cambios políticos, económicos y sociales que garanticen la seguridad y calidad de vida del pueblo venezolano. En el marco de la Alternativa Bolivariana para las Américas (ALBA) se estableció de común acuerdo entre Cuba y Venezuela, el compromiso para desarrollar programas y proyectos de cooperación. Los mismos fueron creados con el objetivo de fomentar el progreso de sus economías y de aprovechar las ventajas recíprocas. Nacida al calor de la Batalla de Ideas, la Universidad de las Ciencias Informáticas (UCI) es una de las principales protagonistas de esta historia.

La Solución Tecnológica Integral para la Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela es uno de estos proyectos que dotará de una infraestructura informática que posibilite organizar y administrar de forma eficiente la información referente a los antecedentes penales de los ciudadanos de la nación.

La División de Antecedentes Penales (DAP) adscrita a la Dirección General de Justicia, Instituciones Religiosas y Cultos, perteneciente al Vice Ministerio de Política Interior y Seguridad Jurídica, tiene como responsabilidad el registro constitutivo de los antecedentes penales de los ciudadanos venezolanos y la emisión de certificaciones de antecedentes penales, en un plazo razonable, a los distintos entes tanto

nacionales como internacionales autorizados por la legislación venezolana vigente, garantizando respuestas efectivas a las solicitudes realizadas ante esta oficina en la medida de sus responsabilidades.

La DAP es central, por lo que todo el proceso se realiza en Caracas con las consecuentes demoras y dificultades en la recepción y entrega de la información desde todas las partes del país así como la expedición de certificaciones.

El constante deterioro de los expedientes en formato duro almacenados desde los últimos cincuenta años de trabajo aproximadamente, hace muy engorroso su manipulación debido a rasgaduras, escritura borrosa, y tintas corridas. La pobre infraestructura tecnológica en cuanto a condiciones de redes de transmisión de datos y equipamiento existentes actualmente en la entidad, todo esto sumado al inconveniente de no contar con alternativas para la manipulación digital y búsqueda de información por categorías son algunos de los problemas presentes.
La Solución Tecnológica está compuesta por la creación de un Centro se Solución de Gestión, que garantice la automatización de los principales procesos asociados a la obtención de objetos digitales con valor legal a partir del archivo físico que puedan ser integrados con la solución propuesta, incrementen los niveles de gestión y permitan una mejor conservación y protección del fondo documental.

La DAP cuenta con una aplicación informática que se conoce como el Sistema de Registro y Control de Antecedentes Penales (SIRCAP), que posee una tecnología obsoleta y no abarca todos los procesos que se realizan actualmente en esta institución, lo que impide a los funcionarios el correcto desarrollo de la gestión de los antecedentes penales de los ciudadanos venezolanos.

Este software recoge la información de solo tres años a partir del 2007, por lo que no resuelve el problema ya que es necesario realizar búsquedas en el archivo físico para el caso de existencia de antecedentes penales de años anteriores.

El Centro de Solución de Gestión estará concebido por 8 Áreas de Trabajo para facilitar la implementación y automatización del proceso de certificación de antecedentes penales.

Con el fin de dar respuesta a las necesidades planteadas por la DAP se crea en el Centro de Gobierno Electrónico, perteneciente a la Facultad 3 en la Universidad de las Ciencias Informáticas, el proyecto de desarrollo Solución Tecnológica Integral para la Automatización y Modernización de la División de

Introducción

Antecedentes Penales de la República Bolivariana de Venezuela, el cual dotará de una infraestructura informática que posibilite organizar y administrar de forma eficiente la información referente a los antecedentes penales, al abarcar los procesos de gestión de los datos de los sancionados, servicios de inscripción, certificación y al mismo tiempo, brindará información al estado y a las instituciones delimitadas por la ley.

Con la implantación de este sistema se pretende mantener actualizada la información de todo lo que se inscribe y disminuir el tiempo de respuesta a la solicitud de certificación y mejorar el ambiente de trabajo contribuyendo así a ofrecer un mejor servicio a la ciudadanía y al propio Estado.

El proyecto DAP se ha establecido como objetivo garantizar la calidad, confiabilidad y rapidez en la gestión de la información de la Coordinación de Antecedentes Penales del Ministerio del Poder Popular para las Relaciones Interiores y de Justicia en Venezuela.

Con un sistema integral automatizado, que brinde el servicio de solicitud y respuesta de Certificación en un tiempo prudente acorde a la Legislación que rige las funcionalidades de este Registro y la Ley de Agilización de Trámites, y mantenga actualizada la información de los Antecedentes Penales de los ciudadanos del país, concentrando en él toda la información en soporte digital, se logra mantener la calidad, confiabilidad y rapidez de la información penal, así como el aumento de la productividad y humanización del trabajo en el Registro de Antecedentes Penales.

El proceso de certificación de antecedentes penales afecta de manera directa a la población venezolana pues esta es una institución centralizada con una población de alrededor de 26 millones de habitantes todo aquel que requiera conocer o presentar de manera legal sus antecedentes penales deberá dirigirse a la División, el proceso de certificación hoy se realiza en una taquilla en la que se presenta la solicitud con los recaudos necesarios, estos datos son anotados por el funcionario de taquilla, luego pasa a los operadores que se encargan de buscar en una BD si la persona posee antecedentes penales y luego hacer un aplantilla en Word donde recopile todos los antecedentes penales de la persona, si esta no posee entonces debe crear otro tipo de respuesta. Este proceso no pasa por un control de calidad o revisión legal en el que un abogado diga si todos los datos son correctos y si se entrega el tipo de respuesta que lleva, estas respuesta son firmadas por el Director de la división y pasan a taquilla otra vez para su entrega, este es un proceso que hoy para una persona tiene una habilitación de 5 días y que

podría bajarse a cuestión de uno con la solución que se propone insertando en el mismo un proceso de revisión legal de los trámites, automatizando los procesos en cada una de las áreas que intervienen.

Para llevar a cabo la implementación de la solución informática se ha modelado un sistema dividido en subsistemas, siendo uno de ellos el módulo de Certificación, que constituye el objeto de la presente investigación.

En una primera fase de desarrollo de la solución que permitió la identificación de los requisitos funcionales y no funcionales del sistema, se diseñó el modelo del sistema y se validó la propuesta a partir de los prototipos aprobados con el cliente, dando paso a una próxima etapa de desarrollo donde se identifica el siguiente problema de la investigación:

¿Cómo satisfacer los requisitos acordados con el cliente de manera que contribuya con el proceso de certificación de antecedentes penales en la República Bolivariana de Venezuela?

El **objeto de estudio** estará enmarcado en el proceso de desarrollo de software.

Como **objetivo general** de la investigación se plantea:

Realizar el diseño e implementación del módulo Certificación de Antecedentes Penales de la Solución Tecnológica Integral para la Automatización y Modernización de la División de Antecedentes Penales.

Para darle cumplimiento al objetivo general propuesto se han definido los siguientes **objetivos específicos**:

- Elaborar el marco teórico de la investigación.
- Realizar los modelos de diseño e implementación.
- Validar la solución propuesta.

El **campo de acción** en el cual estará enmarcada la investigación es en la fase de diseño e implementación en el desarrollo de software.

Para guiar la investigación propuesta se plantea la siguiente **idea a defender**:

Introducción

Realizando el diseño e implementación del módulo certificación de antecedentes penales entonces se contribuirá a mejorar el proceso de emisión de certificados sobre antecedentes penales almacenados en la DAP.

Tareas de la investigación

1. Estudio de los sistemas registrales y de gestión documental.
2. Análisis de las herramientas para el desarrollo de software, herramientas CASE, metodología de software, plataforma de desarrollo y gestores de bases de datos.
3. Desarrollo de los artefactos de la realización de casos de uso que comprende el modelo de clases y los diagramas de interacción.
4. Realización del Modelo de Diseño para los módulos del sistema correspondiente.
5. Implementación del módulo correspondiente de acuerdo a las normas y estándares establecidos por el equipo de proyecto.
6. Diseño y aplicación de pruebas unitarias y de integración a los módulos implementados para validar la solución propuesta.

Métodos de investigación científica:

Los métodos de investigación a emplear tienen como punto central la interacción dialéctica de los métodos teóricos y empíricos.

Los **métodos teóricos** a utilizar son los siguientes:

- **Análisis histórico-lógico:** permitió un estudio del estado del arte haciendo un análisis de los fenómenos relacionados con los sistemas informáticos implementados en el país y en el mundo, así como sus ventajas y desventajas. Además de un estudio de la gestión de indicadores.
- **Analítico-Sintético:** permitió buscar la esencia de los fenómenos, en este caso de los sistemas informáticos implementados en Cuba y el mundo, los rasgos que los caracterizan y los distinguen.

Los **métodos empíricos** a utilizar son los siguientes:

- **Observación:** con el objetivo de obtener información, datos y otros aspectos con los que no se contaba para el desarrollo de la tesis.
- **Análisis de las fuentes de información:** Para consultar fuentes de información relacionadas con el tema de la investigación en la ingeniería de confiabilidad.

Estructura capitular:

Capítulo I: Fundamentación teórica. Se abordarán todos los elementos teóricos que sustentan el problema científico y los objetivos del trabajo además, de un análisis de las diferentes herramientas, metodologías y tecnologías usadas.

Capítulo II: Descripción de la solución. Se propone el diseño de la solución a partir de la especificación de los requisitos de software. Se exponen los patrones de diseño utilizados y se realiza una breve descripción de la arquitectura para lograr el entendimiento del diseño. Se muestran como resultado el modelo de diseño, los diagramas de clases y la descripción de cada una de ellas por casos de uso además, las normas de diseño, el diagrama de paquetes y el modelo de datos utilizado en el sistema.

Capítulo III: Implementación y Pruebas. Quedarán establecidas las interfaces del sistema de gestión para el módulo de certificación, así como la validación de la solución propuesta, se realizarán las pruebas unitarias al producto y la aplicación de las métricas de diseño.

Capítulo 1: Fundamentación Teórica

Este capítulo abarca temas de la investigación relacionados con algunos conceptos básicos relacionados con términos legales necesarios para el desarrollo de la presente investigación, las diferentes herramientas y metodologías a utilizar en el módulo de certificación, tanto para el diseño como para la implementación. Se describirán los frameworks más usados en las comunidades de desarrollo, así como su importancia además, se caracterizarán algunos IDE's utilizados en el desarrollo.

1.1. Definición de documento legal

En el diccionario llamado "Hacia una terminología archivística", se define documento como toda expresión en lenguaje natural, convencional y cualquier otra expresión drástica, sonora o en imagen recogidas en cualquier tipo de soporte material, incluso los soportes informáticos.[1] (Autores, 2005)

Otra definición se puede encontrar en el Diccionario Jurídico de Cabanellas[2] el cual define al documento como instrumento, escritura, escrito con que se aprueba, confirma o justifica alguna cosa o al menos que se aduce con tal propósito.

Los documentos legales son aquellos que están expedidos por alguna autoridad oficial como ejemplo de ello tenemos el acta de nacimiento, el cual es expedido por la oficina del registro civil.

Según el artículo 26A se considera documento todo soporte material que exprese o incorpore datos, hechos o narraciones con eficacia probatoria o cualquier otro tipo de relevancia jurídica.[3]

La era de la información se está haciendo cada día más presente, donde los documentos con probado valor legal toman protagonismo y se hacen imprescindibles, debido a las diferentes formas en las que se puede presentar un documento y las vías para demostrar autenticidad en los mismos.

---

1 Grupo Iberoamericano de tratamiento de archivos administrativos. Hacia un diccionario de terminología archivística. GITAA. Santafé de Bogotá. D.C. Archivo General de la Nación. 1997. 133p.

[2]Cabanellas, Guillermo. Diccionario Enciclopédico de Derecho Usual. Argentina. Editorial Eliasta S.R.L

[3]Definiciones según Código Penal Español aprobado por Ley Orgánica 10/1995, de 23 de noviembre.

Capítulo 1: Fundamentación Teórica

1.1.1. Certificación de documentos

Certificar consiste en expedir un documento público: Certificación de Antecedentes Penales, el que contiene los datos relativos a los sancionados o penados, los delitos cometidos con sus respectivas sanciones así como el tiempo de ejecución de la sentencias y que son asentados en un asiento registral.
La certificación es un documento público con carácter probatorio y valor legal ante terceros, por eso se dice que es un medio de prueba, de ahí la importancia de que todo sistema informático para una organización registral como lo es el Registro de Antecedentes Penales, tiene que ajustarse a principios jurídicos que aseguren que los procesos de inscripción y certificación desarrollados digitalmente se correspondan con los principios de Seguridad Jurídica. (Amoroso, 2009)

Hoy en día, los procesos de certificación se han convertido en un elemento esencial para el desempeño eficiente de empresas, personas y organizaciones. De esta forma, en los últimos años se ha evidenciado el gran auge que han tenido estos procesos y que han desencadenado la aparición de una terminología a veces errónea y, en muchos casos, confusa sobre el propio concepto de certificación, sus características, funciones y efectos.

El sistema que se implemente debe cumplir con las leyes establecidas en un país, en el caso que nos ocupa las leyes son: Constitución de la República Bolivariana de Venezuela, Código Orgánico Procesal Penal, Ley de Registro de Antecedentes Penales, Ley Orgánica de la Administración Pública, Decreto con Fuerza de Ley y Firma digital. Hoy en día los documentos electrónicos están cobrando vigencia debido a su fácil manipulación, en el mundo existen software que se dedican a certificar documentos, estos son hechos a la medida, cumpliendo con el basamento legal establecido por cada país. Todo esto es necesario para otorgar a los documentos un valor legal.

1.1.2. Antecedentes Penales

El término antecedente se utiliza normalmente para referirse a aquellas circunstancias que se han producido con anterioridad y anticipación a otras y que normalmente pueden servir para juzgar situaciones o acontecimientos posteriores o bien para comparar hechos pasados con hechos presentes y futuros.

Capítulo 1: Fundamentación Teórica

Antecedentes penales son aquellas anotaciones que se realizan en un registro correspondiente, dependiente del Ministerio del Interior, como ser los registros que lleva el poder judicial de una región en particular, de las condenas impuestas a los individuos como consecuencia de la comisión de algún delito.

1.2. Metodologías de desarrollo de software

La ingeniería de software incluye el análisis previo de la situación, el diseño del proyecto, el desarrollo del software, las pruebas necesarias para confirmar su correcto funcionamiento y la implementación del sistema.

La metodología indica cómo hay que obtener los distintos productos parciales y finales. La ingeniería de software es una tecnología multicapa en la que, según Pressman, se pueden identificar: los métodos, el proceso y las herramientas. La finalidad de una metodología de desarrollo es garantizar la eficacia y la eficiencia minimizando las pérdidas de tiempo en el proceso de generación de software. (Pressman, 2007)

1.2.1. Proceso Unificado de Desarrollo (RUP)

El Proceso Unificado de Desarrollo es una metodología para el desarrollo de software orientado a objetos. Es un proceso de desarrollo de software, definido como un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. (Sanchez, 2004)

Está constituido por 9 flujos de trabajo (los 6 primeros son conocidos como flujos de ingeniería y los 3 últimos de apoyo): modelamiento del negocio, requisitos, análisis y diseño, implementación, prueba, instalación, administración de configuración y cambios, administración de proyectos, ambiente, los cuales tienen lugar sobre 4 etapas o fases: inicio, elaboración, construcción y transición. Esta metodología es adaptable para proyectos a largo plazo y establece refinamientos sucesivos de una arquitectura ejecutable.

RUP es un proceso de desarrollo de software que junto a UML constituyen la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. (Jacobson, 2000)

1.2.2. XP

Se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes y simplicidad en las soluciones implementadas. Esta metodología se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. Para la especificación de las funcionalidades del sistema se utilizan tarjetas de papel (historias de usuarios) en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. (Urbino, 2010)

1.2.3. SCRUM

Más que una metodología de desarrollo de software, es una forma de auto-gestión de los equipos de trabajo en forma general. Un grupo de integrantes del equipo de trabajo decide cómo hacer sus tareas y cuánto van a tardar en ello. Ayuda a que trabajen todos juntos, en la misma dirección, con un objetivo claro.

Permite además seguir de forma clara el avance de las tareas a realizar, de forma que los jefes puedan ver día a día cómo progresa el trabajo. Sin embargo, no es una metodología de desarrollo, puesto que no indica qué se debe hacer para realizar el código. Debería, por tanto, complementarse con alguna otra metodología de desarrollo. Además interactúa con las metodologías ágiles y en especial con XP. (Rodríguez, 2010)

1.3. Lenguajes de Modelado

Un sistema, tanto del mundo real como en el mundo del software, es bastante complejo, por ello es necesario dividir el sistema en partes o fragmentos si se quiere entender y administrar su complejidad. Estas partes se pueden representar como modelos que describan sus aspectos esenciales. Por tanto, un paso útil en la construcción de un sistema de software es el de crear modelos que organicen y comuniquen los detalles más importante de la vida real con que se relacionan y del sistema a construir. (Caro, 2007)

1.3.1. Lenguaje Unificado de Modelado (UML)

El Lenguaje de Modelado Unificado UML es un lenguaje estándar para escribir planos de software. Puede utilizarse para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra gran cantidad de software. Es un lenguaje que ayuda a interpretar grandes sistemas mediante gráficos o texto, obteniendo modelos explícitos que contribuyen a la comunicación durante el desarrollo, ya que al ser estándar, pueden ser interpretados por personas que no participaron en su diseño. En este contexto, UML sirve para especificar modelos no ambiguos y completos. UML es un método formal de modelado. Esto aporta las siguientes ventajas:

- Mayor rigor en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se puede automatizar determinados procesos y permite generar código a partir de los modelos y a la inversa. Esto hace que el modelo y el código estén actualizados, con lo que siempre se puede mantener la visión en el diseño, de más alto nivel, de la estructura de un proyecto. (Orallo, 2007)

UML propone diagramas con la finalidad de presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. Un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementarlo.

1.3.2. Definición Integrada para el Modelado de Funciones (IDEF0)

IDEF0 es una técnica de modelación concebida para representar de manera estructurada y jerárquica las actividades que conforman un sistema o empresa, y los objetos o datos que soportan la interacción de esas actividades. Un modelo IDEF0 se compone de una serie jerárquica de diagramas que permiten mediante niveles de detalle, describir las funciones especificadas en el nivel superior. En las vistas superiores del modelo la interacción entre las actividades representadas permite visualizar los procesos fundamentales que sustentan la organización. Los elementos gráficos utilizados para la construcción de los diagramas IDEF0 son cuadros y flechas.

Permite representar el proceso cronológicamente. Se describe el flujo orientado al cliente final de ese negocio, cruzando todas las actividades de la organización que dan cumplimiento a la solicitud de producto o servicio que realiza el cliente, representando así la "cadena de valor" de la empresa. Permite

incorporar en el flujo los datos que entran y salen de las actividades, así como las reglas del negocio y los actores, todo en la misma vista. (Hanrahan, 1995)

1.3.3. Notación para el Modelado de Procesos de Negocio (BPMN)

El objetivo principal de BPMN es proporcionarle a toda las personas involucradas en el negocios una notación factible para entender cómo funcionan los procesos que componen una empresa, desde el analista de negocio que es el encargado de hacer el borrador inicial, pasando por los desarrolladores que son los encargados de implementar todo el sistema que ejecutara dichos procesos, hasta las personas que son los encargados de ejecutar y gestionar los procesos. BPMN es considerado como un mecanismo simple para crear modelos de procesos de negocio, que lo maneja a través de las cuatro categorías básicas de elementos: objetos de flujo, objetos conectores, rol de proceso, artefactos.

BPMN define un Diagrama de Procesos de Negocio (BPD, del inglés Business Process Diagram), que se basa en una técnica de grafos de flujo para crear modelos gráficos de operaciones de procesos de negocio. Un modelo de procesos de negocio, es una red de objetos gráficos, que son actividades (trabajo) y controles de flujo que definen su orden de rendimiento. El BPMN se compone de varios conjuntos de elementos que abarcan la representación, tanto de los Objetos del flujo y sus conexiones como los instrumentos de ayuda que son las Bandas (Swimlanes) y los Artefactos. Además está diseñado para cubrir muchos tipos de modelados y para permitir la creación de segmentos de proceso así como procesos de negocio end-to-end, con diferentes niveles de fidelidad. (Barriento, 2008)

1.4. Lenguajes de programación

Los lenguajes de programación son un conjunto de programas que controlan el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Están formados por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. (Programacion, 2010)

1.4.1. Java

Java es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha sintaxis de C y C++, pero tiene un modelo de objetos

más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria. (Becerril, 1998)

Unas de las principales características de este lenguaje son:

- **Simple**: Elimina muchas de las características desventajosas de otros lenguajes como C++, para mantener reducidas las especificaciones del lenguaje y añadir características muy útiles como el reciclado de memoria dinámica. El mismo reduce en un 50% los errores más comunes de programación con lenguajes.

- **Robusto**: Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria. Estas características reducen drásticamente el tiempo de desarrollo de aplicaciones en Java. El mismo realiza todo esto sin necesidad de que el programador se lo indique.

- **Seguro**: Las aplicaciones de Java resultan extremadamente seguras, ya que no acceden a zonas delicadas de memoria o de sistema, con lo cual evitan la interacción de ciertos virus, es decir, la seguridad se integra en el momento de compilación, con el nivel de detalle y de privilegio que sea necesario.

- **Portable**: Más allá de la portabilidad básica por ser de arquitectura independiente, Java implementa otros estándares de portabilidad para facilitar el desarrollo. Un programa compilado de Java puede ser utilizado por cualquier computadora que tenga implementado el intérprete de Java.

- **Multiprocesador**: El beneficio de ser multiprocesador consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real. Aunque el comportamiento en tiempo real está limitado a las capacidades del sistema operativo subyacente (Unix, Windows), aun supera a los entornos de flujo único de programa, tanto en facilidad de desarrollo como en rendimiento.
- **Dinámico**: Java, para evitar que los módulos de clases haya que estar transfiriéndolos de la red cada vez que se necesiten, implementa las opciones de persistencia, para que no se eliminen cuando se borre la memoria caché de la máquina.

Por los beneficios planteados anteriormente y por resaltar Java como lenguaje de programación adecuado para aplicaciones de gran envergadura, fue seleccionado para el desarrollo de SIGESAP y en particular del Módulo de Certificación.

Capítulo 1: Fundamentación Teórica

1.5. Entorno de desarrollo integrado de programación

1.5.1. Netbeans

Es un IDE multilenguaje completo y modular, constituye una plataforma para construir aplicaciones proporcionando aplicaciones completas para el cliente. Mediante este IDE se pueden desarrollar aplicaciones de escritorio, web, mobile y Enterprise. Funciona sobre disimiles sistemas operativos como OpenSolaris, Linux, Windows y Mac OS. (Netbeans, 2011)

Entre las características más sobresalientes del Netbeans como IDE se encuentran: poderoso, extensible, abierto y gratis además permite operar un conjunto de prácticas. Incluye varias mejoras en el editor de código, ya que completa el código más inteligentemente y resaltado. Su instalación y actualización es mucho más simple que otros IDE' s de desarrollo. Permite crear gráficos de juegos para celulares e incalculables mejoras para SOA y UML.

En cuanto al trabajo en grupo desarrolla aplicaciones colaborativas, dando la opción para mejorar la mensajería instantánea y comparte el entorno de desarrollo completo.

1.5.2. Eclipse

Es una plataforma para el desarrollo de software. Incluye un entorno de desarrollo de plugins y Java Development Tools además, de la fuente y la documentación de usuario y programador. Es un software de desarrollo creado por IBM inicialmente para desarrollar sobre JAVA. Es un IDE multilenguaje que tiene vinculación con gran cantidad de plugins y cuenta con un completamiento de código muy superior al de Netbeans. (Integrado, 2010)

1.6. Framework para el desarrollo

Un framework (término en inglés para marco de trabajo) es un diseño reutilizable de todo o parte de un sistema software descrito por varias jerarquías de herencia de clases, generalmente algunas abstractas, y por las colaboraciones que se establecen entre las instancias de estas clases. La reutilización se produce mediante la instanciación del framework. (Félix Prieto, 2000)

Framework es un concepto sumamente genérico, se refiere a “ambiente de trabajo, y ejecución”. En general los framework son soluciones completas que contemplan herramientas de apoyo a la construcción (ambiente de trabajo o desarrollo) y motores de ejecución (ambiente de ejecución).

Un framework es una estructura de soporte mediante la cual puede ser desarrollado un proyecto de software. Generalmente incluye programas, bibliotecas y un lenguaje interpretado que optimizan y aceleran el proceso de desarrollo del software. Estos diagramas dan una vista general de lo que se desea desarrollar para una mejor comprensión.

1.6.1. SWING

Es un conjunto de clases de Java que simplifica la construcción de aplicaciones de escritorio. Permite tener un sistema de ventanas y componentes gráficos, independiente del sistema operativo y la librería de dibujo que se tengan disponibles en la máquina clientes.

Swing es un extenso conjunto de componentes que van desde los más simples, como etiquetas, hasta los más complejos, como tablas, árboles, y documentos de texto con estilo. Casi todos los componentes Swing descienden de un mismo padre llamado JComponent que desciende de la clase de AWT Container. La característica más notable de los componentes Swing es que están escritos el 100% en Java y no dependen de componentes nativos. (Autores, 2011)

1.7. Tecnología de servidor

1.7.1. Enterprise Java Beans (EJB)

EJB es la tecnología del lado del servidor para la arquitectura de componentes de la Plataforma Java, (Java EE). La tecnología EJB permite el desarrollo rápido y simplificado de aplicaciones distribuidas, transaccionales, seguras y portátiles basadas en tecnología Java. EJB 3.0 define la nueva API EJB simplificado dirigido a la facilidad de desarrollo e incluye el nuevo Java Persistence API para la gestión de la persistencia. (JDeveloper, 2011)

Son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE. Su especificación detalla cómo los servidores de aplicaciones proveen objetos desde el lado del servidor que son, precisamente, los EJB:

- Comunicación remota utilizando CORBA
- Control de la concurrencia
- Servicios de nombres y de directorio
- Ubicación de componentes en un servidor de aplicaciones.

Los EJB proporcionan un modelo de componentes distribuido estándar del lado del servidor. El objetivo de los EJB es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia y seguridad) para centrarse en el desarrollo de la lógica de negocio en sí. El hecho de estar basado en componentes permite que éstos sean flexibles y sobre todo reutilizables.

En el caso específico del proyecto SIGESAP es usado EJB como una tecnología. Lo que permite tener bien delimitadas cuales son las funciones del servidor y publicar en un objeto EJB los gestores remotos que permiten el acceso a los mismos. Un aspecto a destacar sobre el uso y las prestaciones de EJB es un elemento de seguridad que permite restringir el uso de clases y métodos específico dentro de estas a usuario en particular, logrando así que ningún usuario pueda tener accesos a funcionalidades que no le corresponden.

1.7.2. Java Persistence API (JPA)

JPA es el estándar para la gestión de la persistencia y provee facilidades para el mapeo objeto / relacional a desarrolladores de aplicaciones, haciendo uso del modelo de dominio de Java para administrar bases de datos relacionales. JPA es parte de la plataforma Java EE.
Es la ORM para el mapeo de objetos de acceso a datos para que sean entendidos por el lenguaje de programación.

1.7. Glassfish

Contenedor de aplicaciones que implementa la plataforma JavaEE5, por lo que soporta las últimas versiones de tecnologías como: JSP, JSF, Servlets, EJBs, Java API para Servicios Web (JAX-WS), Arquitectura Java para Enlaces XML (JAXB), Metadatos de Servicios Web para la Plataforma Java 1.0, y muchas otras tecnologías. (Miranda Gonzalez, 2005)

Anteriormente mencionamos que Glassfish implementa la Plataforma JavaEE5. Java Enterprise Edition (JEE) es esencialmente una forma estándar de desarrollar Aplicaciones Java Empresariales que sean portables, esto quiere decir, que puedan ser utilizadas en más de un servidor sin importar el fabricante, sin necesidad de hacerles cambio alguno. Existe una versión de Glassfish que incluye soporte comercial por parte de Sun Microsystems, ésta versión está enfocada a empresas y universidades.

1.8. Sistema Gestor de Bases de Datos (SGBD)

Un SGBD es un conjunto de programas que permiten crear y mantener una Base de datos, asegurando su integridad, confidencialidad y seguridad. Por tanto debe permitir:

- Definir una base de datos: especificar tipos, estructuras y restricciones de datos.

- Construir la base de datos: guardar los datos en algún medio controlado por el mismo SGBD.

- Manipular la base de datos: realizar consultas, actualizarla, generar informes. (Collector, 2004)

1.8.1. Oracle

Es uno de los SGBD más utilizados en la gestión de grandes bases de datos debido a su alto rendimiento en transacciones y disponibilidad. Además, presenta otras ventajas como portabilidad, compatibilidad y gestión de seguridad, pero su uso exige ser pagado a un elevado precio, lo que provoca que sea rechazado por muchas empresas que se dedican al desarrollo de software de gestión. (Kevin, 2008)

1.8.2. MySQL

Es un sistema de gestión de bases de datos relacional, licenciado bajo la GPL de la GNU, creada por la empresa sueca MySQL AB. Las principales características de este SGBD son las siguientes:

- Aprovecha la potencia de sistemas multiprocesadores, gracias a su implementación multihilo.
- Rápido y fácil de usar.
- Fácil de instalar y configurar.
- Cuenta con una infinidad de librerías y herramientas que facilita su uso.
- Capacidad de gestionar y almacenar grandes cantidades de datos.

- Gran portabilidad entre sistemas.
- Gestión de usuarios y contraseñas, manteniendo un buen nivel de seguridad en los datos. (Corporation, 2010)

1.8.3. Postgresql

Postgresql es un gestor de bases de datos orientadas a objetos soportando un conjunto de funcionalidades avanzadas, lo que lo sitúa al mismo o a un mejor nivel que muchos SGBD comerciales.

Está ampliamente considerado como uno de los SGBD de código abierto más avanzado del mundo. Este proporciona un gran número de capacidades como: consultas complejas, manejo de vistas, integridad referencial, bloqueo de tabla y filas, entre otras. Originalmente, al igual que otros proyectos de software libre, se desarrolló para operar dentro del sistema operativo Linux, pero las mismas exigencias de los usuarios han contribuido a que en la actualidad se disponga de versiones para Windows. (Lockhart, 2001)

Se utilizará como sistema gestor de base datos Postgresql, para el desarrollo del proyecto SIGESAP, en particular del Módulo de Certificación, por el gran prestigio a nivel mundial. Postgres es considerado como uno de los sistemas de bases de datos más completos, destacando: soporte de transacciones, estabilidad, escalabilidad y soporte multiplataforma.

1.9. Patrones de diseño

El catálogo de patrones más famoso es el contenido en el libro “DesignPatterns: Elements of Reusable Object-Oriented-Software”, cuyos autores son Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides (normalmente referenciados como la Banda de los Cuatro – TheGangofFour – o simplemente por GoF)

Los patrones más importantes a usar en el desarrollo son:

De Creación: abstraen el proceso de creación de instancias.

- Singleton: para la creación de una instancia de las clases que sean necesarias, como los gestores del negocio las clases de Internacionalización, entre otras.

Estructurales: se ocupan de cómo clases y objetos son utilizados para componer estructuras de mayor tamaño.

- Facade Mediador o Mediator: para proporcionar una interfaz para la creación de familias de objetos interdependientes o interrelacionados, sin especificar sus clases concretas. El patrón se utilizó para la creación de la clase FachadaGestoresRemotos para la facilitar la conexión del los gestores cliente y servidor.

De Comportamiento: atañen a los algoritmos y a la asignación de responsabilidades entre objetos.

- Iterator: para proporcionar una forma coherente de acceder secuencialmente a los elementos de una colección, independientemente del tipo de colección subyacente. El patrón se utilizó para recorrer las estructuras de datos utilizadas.

Los patrones GRASP (General Responsibility Assignment Software Patterns) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Se utilizan en todos las clases gestores: cliente, servidor y acceso a datos.

- Experto: para que cada objeto realice la funcionalidad de acuerdo a la información que domina, la cuestión a la hora de diseñar es asignar responsabilidades a la clase que mayor información posee para cumplir con dicha tarea.
- Creador: para encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Brinda soporte a un bajo acoplamiento lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización.
- Bajo Acoplamiento: para estimular la asignación de responsabilidades de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento. El Bajo Acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables.
- Alta cohesión: para brindar una alta cohesión funcional cuando los elementos de un componente (clase, por ejemplo) "colaboran para producir algún comportamiento bien definido".
- Controlador: para asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones:
    - El "sistema" global (controlador de fachada).

- La empresa u organización global (controlador de fachada).
- Algo en el mundo real que es activo (por ejemplo, el papel de una persona) y que pueda participar en la tarea (controlador de tareas).
- Un manejador artificial de todos los eventos del sistema de un caso de uso, generalmente denominados "Manejador<Nombre Caso de Uso>" (controlador de casos de uso).

1.10. Herramientas Case

Las herramientas CASE (Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software. Estas herramientas tienen un sin número de beneficios ofrecidos por la tecnología CASE, se pueden encontrar los siguientes:

- Facilidad para llevar a cabo la tarea de revisión de especificaciones del sistema así como de representaciones graficas (lo que aumenta la posibilidad de realizar la tarea).
- Facilidad para desarrollar prototipos de sistemas por medio de la capacidad para cambiar especificaciones.
- Generación de código. (Mosquera, 2008)

1.10.1. Visual Paradigm

Permite la generación de código para varios lenguajes, (en especial Java). Es una herramienta de código abierto y presenta un entorno de creación de diagramas para UML 6.2. Su diseño está centrado en casos de uso y enfocado al negocio generando un software de mayor calidad, presenta capacidades de ingeniería directa e inversa y disponibilidad en múltiples plataformas. (Sierra, 2011)
Una de las características más importantes de su uso es que brinda la posibilidad de sincronización del modelo de diseño y el código en todo el ciclo de desarrollo, permitiendo la facilidad de programar directamente sobre el código fuente generado y a su vez actualizar el diseño con cambios que se realicen en la programación.

1.10.2. Rational Rose

Proporciona un desarrollo iterativo que permite cubrir todo el ciclo de vida de un proyecto, en el que se puede especificar, analizar, diseñar un sistema antes de codificarlo; ayudando a establecer una trazabilidad real entre el modelo (análisis y diseño) y el código ejecutable además, permite generar código a partir de los modelos y realizar ingeniería inversa (crear modelo a partir código). (Innova, 2007)

Sin embargo, aún cuando estas características la convierten en una herramienta muy utilizada actualmente, su particularidad de ser propietaria (no gratuito, restricciones de licencia, código fuente) la hacen ser rechazada por muchas empresas y desarrolladores de software.

1.10.3. Enterprise Architect

Herramienta UML que abarca el ciclo de vida completo del desarrollo de software que provee una estructura competitiva en el modelado de negocio, diseño de software, ingeniería de sistemas, arquitectura de empresas, gestión de requisitos y prueba.

Permite integrar al equipo de desarrollo proporcionando una visión de trabajo compartido. Enterprise Architect es una herramienta multiusuario, basada en Windows, diseñada para ayudar a construir software robusto, fácil de mantener, fácil de usar, rápido y flexible. Ofrece salida de documentación flexible y de alta calidad. Provee una trazabilidad completa desde el análisis de requerimientos hasta los artefactos de análisis y diseño, a través de la implementación y el despliegue. Las bases de Enterprise Architect están construidas sobre la especificación de UML 2.0. (Sparxsystems, 2008)

1.11. Conclusiones

A partir del estudio realizado se determinó utilizar:

- La metodología RUP por ser rica en documentación y facilitarle la comprensión además, fue la seleccionada por la dirección del proyecto Solución Tecnológica Integral para la División de Antecedentes Penales al cual pertenece la actual investigación.

- Se utilizará el lenguaje de modelado de sistema UML en su versión 6.2 por ser un lenguaje muy utilizado en el mundo, que facilita la comprensión de lo que se desea modelar además, por ser el lenguaje que propone la metodología RUP.
- Se empleará como herramienta CASE Visual Paradigm en su versión 3.4, por ser una herramienta muy buena para el modelado, que da soporte a varios lenguajes de modelado como BPMN y UML, permite realizar todo tipo de diagramas como prototipos de interfaz de usuario y modelado de bases de datos. Es una herramienta con licencia comercial y en la UCI se puede hacer uso de ella.
- Netbeans en su versión 6.9.1 por ser unos de los IDE's de desarrollo para java más completos además, el cliente solicitó una aplicación en una plataforma libre.
- Swing como framework para generar interfaces gráficas y la capa de presentación.
- Se decidió utilizar EJB por ser una tecnología de servidor, para la aplicación de los Enterprise Beans.
- Postgress 8.4 puede funcionar en múltiples plataformas.
- Glassfish 2.1 por ser uno de los mejores contenedores de aplicaciones y más completos.
- Se utilizó un hibrido entre las arquitecturas Cliente – Servidor, n-capas y basado en componentes tomando lo mejor de cada una de ellas para hacer más funcional la aplicación.

Capítulo 2: Descripción de la solución

En el presente capítulo se expondrán los diagramas de clases del diseño con sus correspondientes casos de uso haciendo énfasis en el modelo de diseño, el cual le permite a los desarrolladores entender mejor el trabajo que van a realizar además, se propone la elaboración del mismo y se documentan los artefactos de desarrollo que fueron elaborados como resultado de esta tarea.

2.1 Propósito del Diseño

El diseño es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción, lo cual contribuye al desarrollo de una arquitectura estable y sólida. Este flujo de trabajo es un refinamiento del análisis que tiene en cuenta los requisitos funcionales. La realización del diseño permite ser una guía para que los desarrolladores puedan leer y entender el producto que se desea construir. En él se definen las clases y asociaciones que se utilizarán en la implementación, así como las interfaces y los algoritmos de los métodos utilizados para implementar las operaciones. Los objetivos del diseño son:

- ✓ Desarrollar un diseño para el sistema.
- ✓ Adaptar el diseño a las particularidades del sistema para que sea consistente con el entorno de implementación.

2.2 Estándares del diseño

2.2.1 Nomenclaturas

El esquema de nombres es una de las ayudas más importantes para entender el flujo lógico de una aplicación. Un nombre debe más bien expresar el "qué" y no el "cómo". A continuación como nombrar cada uno de los elementos del proyecto, teniendo en cuenta que la nomenclatura se debe definir en español.

En los casos que los nomencladores sean compuestos por varias palabras se usará una notación de Camello. La notación Camello consiste en escribir los identificadores con la primera letra de cada palabra

en mayúsculas y el resto en minúscula: EndOfFile. Se llama notación “Camello” porque los identificadores recuerdan las jorobas de un camello. Existen dos variantes:

1.1 UpperCamelCase: en esta variante la primera letra también es mayúscula.

2.1 lowerCamelCase: la primera letra es minúscula.

En el caso del SIGESAP se utilizará la segunda variante para definir los atributos y variables privadas y en el caso contrario se utilizará la primera variante, aplicando la técnica verbo-sustantivo.

Los paquetes (o package) deberán comenzar con el nombre del sistema (SIGESAP) seguido por el subsistema, luego el módulo y por último la capa lógica a la que pertenece, quedando de la siguiente forma *Sigesap.Subsistema.Modulo.Capa*. Para el caso de los paquetes se usará la misma nomenclatura definida para los métodos de clases y clases.

- Camello se utilizará en los nombres de las Entidades persistentes, Entidades de Dominio, Propiedades y Funcionalidades del sistema.
- Las clases persistentes comenzarán con las siglas definidas en la BD para cada tabla.
- Los formularios de tipo popup comienzan con las siglas Frm, y los de tipo panel con las siglas Pnl.
- Los gestores de negocio del cliente comienzan con las siglas ngestorc y los del servidor con las siglas GestorNC.
- Las clases de la lógica de negocio de acceso a datos comienzan con las siglas GestorAD.
- Las clases de la lógica de negocio del servidor comienzan con las siglas GestorNS.
- Las interfaces publicadas en el servidor terminan con el prefijo Remoto.
- Las clases interceptoras de validación terminan con el prefijo Interceptor.
- Las excepciones terminan con el sufijo Exception.
- Las relaciones entre clases usarán el estereotipo <<use>>.
- Las relaciones entre clases e interfaces usarán el estereotipo <<realize>>.
- Las relaciones entre clases y paquetes usarán el estereotipo <<access>>.

Nomenclatura para los componentes de formularios:

- Los botones (JButton) comienzan con las siglas btn.

- Los campos de texto (JTextField) comienzan con las siglas tfd.
- Los campos fecha (Date) comienzan con las siglas fch.
    - Para el componente JDateChooser ------ dtc
    - Para el componente JDayChooser ------ dyc
- Los componentes de tipo Jscrool usarán las siglas sc.
- Las cajas de texto (JComboBox) comienzan con las siglas cmb.
- Las áreas de texto (JTextArea) comienzan con las siglas txt.
- Los labels (JLabel) comienzan con las siglas lbl.
- Los cuadros de chequeo (JCheckBox) comienzan con las siglas chb.
- Los radiobutton (JRadioButton) comienzan con las siglas rbt.
    - Las tablas (JTable) comienzan con las siglas tbl.

2.3 Pautas generales para el diseño

La separación de los componentes respecto a los bordes del área de trabajo (margen izquierdo, derecho, superior e inferior) debe ser de 10 píxeles como mínimo.

- En un formulario, los datos asociados a una misma cosa se agruparán utilizando paneles con título. El título del panel se escribe en formato de oración, en negrita y sin dos puntos al final.

  Ej.: La dirección, contiene ciudad, calle. Se pondría un panel con título **Dirección** y dentro todos los campos que describen la dirección.

- Cuando el formulario tiene varios componentes. El nombre de las etiquetas se escribe encima de los componentes. El texto en las etiquetas se escribe en formato de oración, sin utilizar negrita y terminado en dos puntos. Los componentes irán debajo, a una distancia de 8 píxeles, cada componente de entrada de datos debe tener una altura de 23 píxeles. La separación horizontal entre componentes debe ser de 20 píxeles y vertical de 15 píxeles. Todo alineado a la izquierda, y en caso de que aparezcan unos debajo de los otros se debe tratar de que todos los componentes tengan un mismo tamaño (el del mayor componente).

- En caso de que el formulario tenga pocos componentes, se pondrá primero la etiqueta y al lado el componente, tanto los componentes como las etiquetas con las mismas pautas anteriores. Además se alinean las etiquetas a la izquierda y los componentes también a la izquierda.
- Cuando la etiqueta no es para una entrada de datos, sino para mostrar los mismos, entonces irá en negrita.

  Ej.: **Nombre y apellidos:** Yusel Sablón Fernández.
- El texto de las etiquetas, títulos de columnas de tablas, botones. debe ser siempre en formato de oración.
- Los combos deben mostrar como texto inicial –Seleccione--. Los ítems para seleccionar deben escribirse en formato de oración y deben aparecer en orden alfabético, a no ser que representen un flujo, en ese caso aparecerían de acuerdo al orden correspondiente. Cuando haya una opción Otros, estará ubicada al final.
- Cuando haya que introducir una fecha debe ponerse un componente para seleccionar la misma, evitar las entradas manuales del usuario.
- Los Botones siempre estarán situados en la parte inferior derecha de los formularios**,** con una altura de 25 píxeles y el ancho depende del Caption, siempre dejando un espacio de 8 píxeles delante y detrás de la palabra o el ícono. Sólo en el caso de los mensajes de confirmación, avisos y error los botones irán en el centro. El texto que contienen debe estar en formato de oración y sin utilizar negrita. Cuando aparezcan varios botones uno al lado del otro, de ser posible todos deben tener el mismo ancho y la separación entre ellos debe ser de 10 píxeles. El ancho mínimo de los botones debe ser de 75 píxeles.
- Los botones no llevan íconos, el Anterior y Siguiente llevan los caracteres <,> respectivamente.
- Los botones de SI y NO en los mensajes de confirmación deben llevar íconos y tendrán un ancho de 50 píxeles.

Capítulo 2: Descripción de la Solución

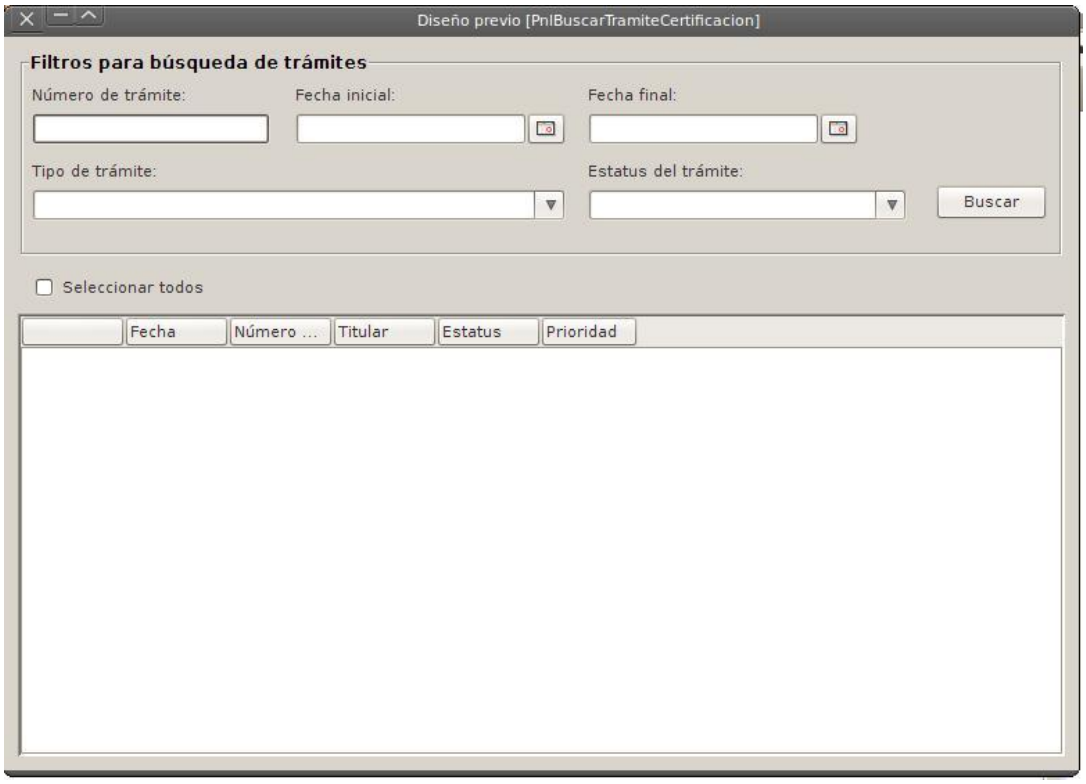

Fig. 1 Panel que muestra Pautas generales para el diseño aplicadas.

2.4 Diseño de la Base de Datos

Como parte del flujo de trabajo del diseño se crea el artefacto diseño de la Base de Datos, a continuación se hace un resumen de los elementos que comprenden la Base de Datos del proyecto haciendo énfasis en los que poseen mayor incidencia en el módulo de certificación.

Está compuesta por dos esquemas:

Framework: Es una aplicación de soporte, creada con el objetivo de facilitar la implementación de todos los módulos que componen la solución tecnológica integral para la división de antecedentes penales.

-Secuencias (13): Se usan para los tipos de datos que toman valores autoincrementales.

.seq_n_area_f

.seq_n_estado_usuario
.seq_n_modulo
.seq_n_rol
seq_t_excepcion
seq_t_imagen
.seq_t_nomenclador
.seq_t_oficina
.seq_t_politica_sistema
.seq_t_punto
.seq_t_sesion
seq_t_traza
.seq_t_usuario

-Tablas (18)

(n).....nomencladores , almacenan datos que constituyen carga inicial para la aplicación, por ejemplo todos los estados de la república bolivariana de Venezuela y sus provincias, el nombre de los entes que solicitan y envían antecedentes penales a la división entre otros indispensables para el trabajo en los módulos, a continuación se listan:

.n_area_f
.n_estado_usuario
n_modulo
.n_rol

(t).....entidades persistentes que almacenaran los datos ingresados desde la aplicación:

.t_excepcion
.t_historial_excepciones
.t_historial_trazas
.t_imagen
.t_nomenclador
.t_oficina
.t_persona_registro
.t_politica_sistema
.t_punto
.t_rol_base

.t_sesion
.t_traza
.t_usuario
.t_usuario_rol_base

2-) **Público** : contiene todas las entidades persistentes necesarias para satisfacer a todos los módulos que componen a la solución del software, este esquema contiene 94 tablas de las cuales no todas las entidades son utilizadas por el modulo certificación, objeto de este trabajo de diploma.

-Secuencias (48)

.seq_c_estado_tramite	.seq_n_motivo	seq_n_tipo_solicitante
.seq_combinacion	.seq_n_ocupacion	.seq_n_tipo_titular
.seq_n_accion_archivo	.seq_n_pais	.seq_n_tipo_tramite
.seq_n_area	.seq_n_parametro_oficio	.seq_n_tipo_unidad_documental
.seq_n_categoria	.seq_n_pie_firma	.seq_numero_tramite_certificacion
.seq_n_categoria_judicial	.seq_n_respuesta	.seq_t_cuerpo_oficio
.seq_n_circunscripcion	.seq_n_tipo_ente	.seq_t_delito_cometido
.seq_n_ciudad	.seq_n_tipo_entrega	.seq_t_expediente
.seq_n_delito	.seq_n_tipo_expediente	.seq_t_nota
.seq_n_ente	.seq_n_tipo_hoja_reo	seq_t_oficio
.seq_n_estado	.seq_n_tipo_libertad_condicional	.seq_t_prestamo_expediente
.seq_n_estado_civil	.seq_n_tipo_nota	.seq_t_recaudo
.seq_n_estado_prestamo	.seq_n_tipo_oficio	.seq_t_recaudo_archivado
.seq_n_estado_tramite	.seq_n_tipo_pena	seq_t_salida_impresa
.seq_n_grado	.seq_n_tipo_persona_cache	.seq_t_solicitud
.seq_n_ley	.seq_n_tipo_recaudo	.seq_t_unidad_documental

-Tablas (94)

.appmanager

(c)...Entidades de configuración en las que se representan las combinaciones que se pueden realizar entre los tipos de trámites, los recaudos, los estados y las áreas en las que pueden estar estos.

.c_estado_tramite
.c_flujo_tramite
.c_pie_firma_oficio

c_tipo_oficio_tipo_tramite
.c_tipo_recaudo_tipo_tramite
.c_tipo_solicitante_tipo_tramite
.c_tipo_titular_tipo_tramite

(n)....Nomencladores que almacenan datos que constituyen carga inicial para la aplicación, por ejemplo todos los estados de la República Bolivariana de Venezuela y sus estados, el nombre de los entes que solicitan y envían antecedentes penales a la división entre otros indispensables para el trabajo en los módulos, a continuación se listan:

.n_accion_archivo
.n_area
.n_categoria
n_categoria_judicial
.n_circunscripcion
.n_ciudad
.n_delito
n_ente
.n_ente_judicial
.n_estado
.n_estado_civil
.n_estado_prestamo
.n_estado_tramite
.n_grado
.n_ley
.n_motivo
.n_ocupacion
.n_pais
.n_pie_firma
.n_tipo_ente
n_tipo_entrega
.n_tipo_hoja_reo
.n_tipo_libertad_condicional
.n_tipo_nota
. n_tipo_oficio
.n_tipo_pena
.n_tipo_recaudo
.n_tipo_solicitante
.n_tipo_titular
.n_tipo_tramite
.n_tipo_unidad_documental

(t)....Entidades persistentes que almacenaran los datos ingresados desde la aplicación:

.t_abogado
.t_actualizacion_datos_reo
.t_cancelacion
.t_cancelacion_archivo
.t_carga_trabajo
.t_cedula
.t_cedula_archivo
.t_certificacion_persona_natural
.t_certificacion_persona_natural_archivo
.t_certificacion_personal_ente
.t_certificacion_personal_ente_archivo
.t_cuerpo_oficio
.t_datos_procesales_sentencia_definitivamente_firme
.t_delito_cometido
.t_delito_cometido_datos_procesales_sdf
.t_delito_cometido_hoja_reo
.t_delito_cometido_unidad_documental_comun
.t_expediente
.t_historial_traza_tramite

.t_hoja_reo
.t_indocumentado
.t_inscripcion
.t_inscripcion_archivo
.t_libertad_condicional
.t_motivo_nota
.t_motivo_nota_archivada
.t_nota
.t_nota_aclaratoria
.t_nota_archivada
.t_oficio
.t_oficio_solicitud
.t_oficio_solicitud_archivo
.t_persona
.t_persona_cache
.t_planilla_solicitud
.t_planilla_solicitud_archivo
.t_poder_notariado
.t_poder_notariado_archivo
.t_prestamo_expediente
.t_recaudo
.t_recaudo_archivado
t_recaudo_archivado_tramite_archivado
.t_recaudo_tramite
.t_reo
.t_salida_impresa
.t_sentencia_definitivamente_firme
.t_solicitud_ente
.t_solicitud_ente_archivo
.t_tramite
.t_tramite_archivado
.t_tramite_usuario
.t_traza_tramite
.t_unidad_documental
.t_unidad_documental_comun
.t_unidad_documental_reo

2.5 Arquitectura del sistema

Antes de definir la arquitectura de una aplicación lo primero que se debe hacer es definir la infraestructura con la que se cuenta para desplegar el software, para ello se tienen en cuenta las necesidades de hardware, cuántas computadoras, servidores, desde qué tipo de dispositivos se accederá. Luego se deben seleccionar y combinar los estilos y patrones arquitectónicos que se emplearán, para terminar seleccionando los subsistemas e integrarlos en la arquitectura. El sistema implementado está soportado por una arquitectura cliente – servidor, distribuida en “n” capas, basada en el desarrollo de componentes y beneficiada por las ventajas de EJB.

A continuación los estilos arquitectónicos que más se acercan a las necesidades de la investigación.

2.5.1 Arquitectura Cliente-servidor

La tecnología Cliente/Servidor es el procesamiento cooperativo de la información por medio de un conjunto de procesadores, en el cual múltiples clientes, distribuidos geográficamente, solicitan requerimientos a uno o más servidores centrales. Desde el punto de vista funcional, se puede definir esta tecnología como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información de forma transparente aún en entornos multiplataforma. Se trata de la arquitectura más extendida en la realización de Sistemas Distribuidos.

En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

2.5.2 Arquitectura en capas

En este estilo arquitectónico cada capa proporciona servicios a la capa superior y se sirve de las prestaciones que le brinda la inferior, al dividir un sistema en capas, cada capa puede tratarse de forma independiente, sin tener que conocer los detalles de las demás.

Los sistemas o arquitecturas en capas constituyen uno de los estilos que aparecen con mayor frecuencia mencionados como categorías mayores del catálogo, o por el contrario, como una de las posibles encarnaciones de algún estilo más envolvente.

La programación por capas es un estilo de programación en la que el objetivo primordial es dividir, fraccionar y llegar a separar la Presentación (donde se muestra y se obtienen datos), de la Lógica de Negocio (donde se realizan operaciones) y con todo esto se obtendría independencia, por lo que si hay cambios de arquitectura se puede acceder a los datos o cambiar el negocio o modificar la presentación y solo sería en esa fracción de la capa.

- **Capa de Presentación**

Es la que ve el usuario, presenta el sistema al mismo, le comunica la información y captura la información del usuario dando un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio. Su función es la de proveer una interfaces para realizar la transferencia de datos.

- **Capa de negocio**

Es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa recibe solicitudes de la capa de presentación, y presenta los resultados, solicitando a la capa de datos, mediante el gestor de base de datos para almacenar o recuperar datos de él.

- **Capa de acceso a datos**

Es el puente entre la capa de Lógica de Negocio y el Sistema de Base de Datos. Encapsula la lógica de acceso a datos. Aquí se encuentran componentes que hacen transparente el acceso a la base de datos. Este es el lugar idóneo para implementar los objetos de acceso a datos (DAOs) estos encapsulan la persistencia de los objetos de dominios, proveen la persistencia de los objetos transitorios y las actualizaciones de los objetos existentes en la base de datos-, permitiendo ingresar, obtener, actualizar y eliminar información del Sistema de Bases de Datos.

- **Enterprise Java Beans**

La arquitectura Enterprise Java Beans (EJB, por sus siglas en inglés) permite un nivel de abstracción de las complejidades de la red en el desarrollo de aplicaciones (distribuidas o no) ya sea con tecnología Web, Desktop o ambas. Esto permite que puedan desarrollarse soluciones muy flexibles y de gran complejidad en cortos períodos de tiempo. Además de estas características se garantiza la seguridad, concurrencia, eficiencia, persistencia y escalabilidad mediante tecnologías integradas en la misma arquitectura.

2.6 Diagrama de paquetes

El objetivo principal es obtener una visión más clara del sistema de información orientado a objetos, organizándolo en subsistemas, agrupando los elementos del análisis, diseño o construcción y detallando las relaciones de dependencia entre ellos. El mecanismo de agrupación se denomina *Paquete.*

Estrictamente hablando, los paquetes y sus dependencias son elementos de los diagramas de casos de uso, de clases y de componentes, por lo que se podría decir que el diagrama de paquetes es una extensión de éstos. En MÉTRICA Versión 3, el diagrama de paquetes es tratado como una técnica aparte, que se aplica en el análisis para la agrupación de casos de uso o de clases de análisis, en el diseño de la arquitectura para la agrupación de clases de diseño y en el diseño detallado para agrupar componentes. (Públicas, 2010)

2.6.1 Descripción

Un paquete es una agrupación de elementos, bien sea casos de uso, clases o componentes. Los paquetes pueden contener a su vez otros paquetes anidados que en última instancia contendrán alguno de los elementos anteriores.

- ✓ Dependencias entre paquetes: Existe una dependencia cuando un elemento de un paquete requiere de otro que pertenece a un paquete distinto. Es importante resaltar que las dependencias no son transitivas. Se pueden optimizar estos diagramas teniendo en cuenta cuestiones como: la generalización de paquetes, el evitar ciclos en la estructura del diagrama, la minimización de las dependencias entre paquetes. (Públicas, 2010)

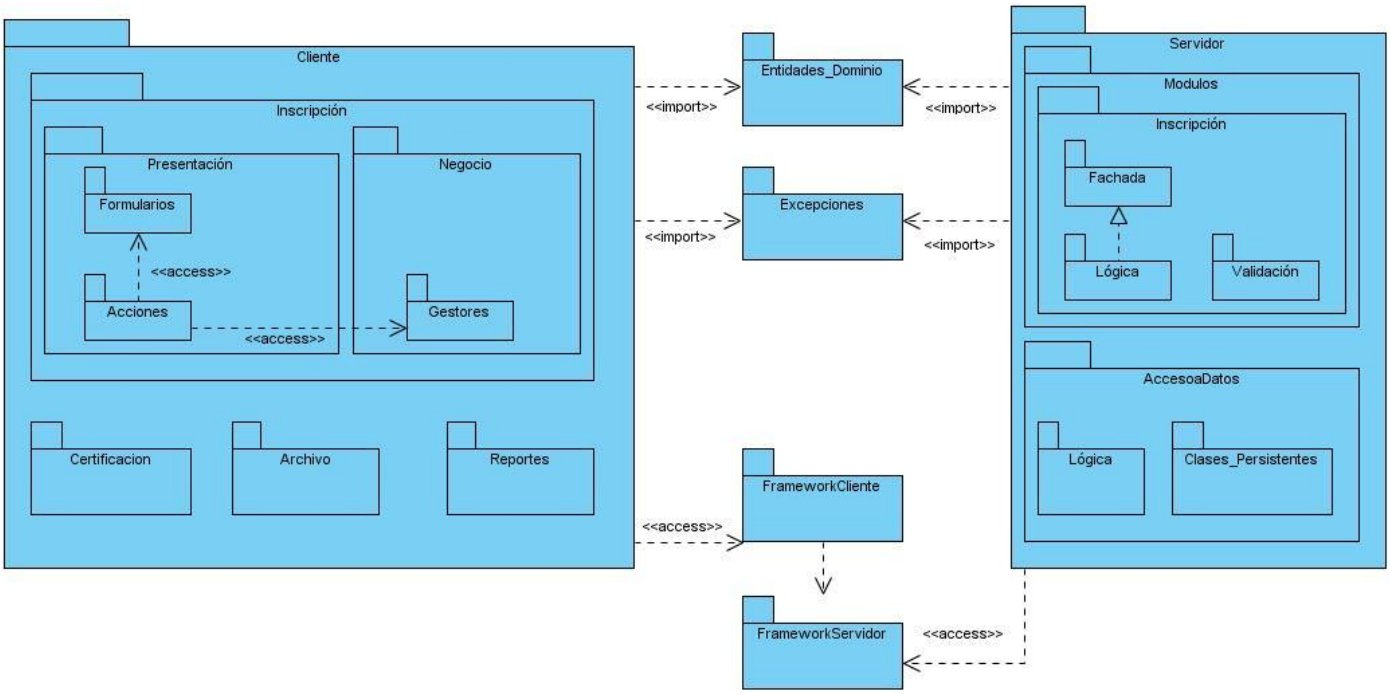

Fig. 2 Diagrama de Paquetes de la Solución de Gestión

2.7 Empaquetado

La aplicación se encuentra dividida en dos subsistemas principales, uno para el Centro de Digitalización y otro para la Solución de Gestión. Los sub paquetes principales dentro del paquete Solución de Gestión son Solucion_Gestion-app-client y Solucion_Gestion-ejb.

2.7.1 Cliente

Solucion_Gestion-app-client: En este sub paquete se almacenan todos los paquetes que contienen las clases, los formularios y las acciones para la parte de presentación.

Imágenes: Dentro de este subpaquete se guardan todos los iconos, banners y otras imágenes que serán utilizadas en la ejecución del proyecto, con el objetivo de lograr una mejor organización y rapidez a la hora de realizar un cambio relacionado con las imágenes del proyecto.

Componentes: En esta se organizan todos los componentes creados por el equipo de proyecto para apoyar el desarrollo del mismo, estos constituyen referencias que la aplicación no puede perder.

i18n: Permite diseñar el software de manera que pueda adaptarse a diferentes idiomas y regiones sin la necesidad de realizar cambios de ingeniería ni en el código.

Negocio: Se encuentran las clases de apoyo a la implementación del negocio cliente que son las encargadas de comunicarse con la Fachada de gestores remotos para así comunicarse con la lógica del servidor.

Presentación: Contiene el paquete de acciones y el de formularios. En el primero se implementaran las acciones de todos los casos de uso, y en el segundo se diseñaran las interfaces de usuario aplicando buenas prácticas de programación.
Se encuentran ubicados tres sub paquetes de códigos "acciones" donde se registraran todas las clases de funcionalidades del sistemas, otro "formularios" para la recogida de datos de y formularios de la aplicación

- **Acciones:** Agrupa las clases controladoras de las funcionalidades que se desean lograr a través de las interfaces. Estas escuchan todos los eventos ocurridos en los paneles, permitiendo así una

mejor reutilización del código y facilidad a la hora de realizar cambios en la apariencia de la aplicación.

- **Certificación:** Reúne todas las acciones de los casos de uso correspondientes al módulo certificación.

Seguridad: Para las validaciones de la aplicación. Se controla la entrada al sistema mediante contraseña, la cual posee un nivel de permiso que está en dependencia del rol.

2.7.2 Servidor

Solucion_Gestion-ejb: En este sub paquete contenedor de otros paquetes que engloban la lógica del negocio en el servidor.

accesoDatos: Contiene todos los gestores que permite conectarse a la base de datos y realizar transacciones a la misma. Cada uno de estos define consultas y obtiene sus resultados a través de la comunicación de los archivos XML almacenados en el paquete **Consultas**.

- **Gestores:** Se encuentran los gestores del negocio servidor y los archivos remotos de estos publican los métodos a los que se acceden desde el negocio cliente.
- **Consultas:** Se encuentran los archivos XML que contienen todas las consultas en algunos casos agrupados por entidades de mayor acceso. Las consultas son escritas con JPA.
- **Factorías:** Este paquete agrupa las clases encargadas de convertir los datos provenientes de la base de datos en entidades del negocio. Así como las entidades del dominio en entidades persistentes.
- **Fachada:** En esta se almacenan los beans locales que solo publican los métodos de cada gestor.
- **Gestores:** Los gestores que implementan los métodos publicados por los beans locales almacenados en el paquete fachada.

2.8 Diagrama de clases del diseño

En un diagrama de clases de diseño se muestran los atributos y métodos de cada clase y se representa de forma sencilla la colaboración y las responsabilidades de las distintas clases que forman el sistema. Se utilizan para modelar la vista de diseño estática de un sistema, esto incluye el vocabulario del sistema, las colaboraciones o esquemas. (ISW, 2010)

Capítulo 2: Descripción de la Solución

Una clase de diseño es una construcción similar en la implementación del sistema. El lenguaje utilizado para especificar estas clases es el lenguaje de programación que se utilizará para la implementación. Las operaciones, atributos, tipos, visibilidad (public, protected, private), se pueden especificar con la sintaxis del lenguaje elegido. Los métodos del diseño tienen correspondencia directa con los métodos de implementación. Una clase de diseño puede proporcionar interfaces si tiene sentido hacerlo en el lenguaje de programación.

El diseño obtenido cumple con los patrones de Bajo acoplamiento y Alta cohesión permitiendo la colaboración entre los elementos de las clases, sin verse afectados la reutilización de los mismos y el entendimiento de estos cuando se encuentran aislados.

A cada clase le fueron asignadas las tareas que podían realizar según la información que poseían, además de crear las instancias de otras clases en correspondencia con la responsabilidad dada; poniéndose de manifiesto los patrones Experto y Creador además, el patrón controlador se usa para asignar la responsabilidad de controlar el flujo de eventos del sistema a clases específicas.

2.9 Descripción de clases del diseño

El diagrama de clases del diseño es el utilizado para representar clases del diseño y sus objetos, como también los subsistemas que contienen dichas clases y las relaciones de cada uno de estos elementos participantes en la realización de uno o varios casos de uso. (Jacobson, 2000)

Capítulo 2: Descripción de la Solución

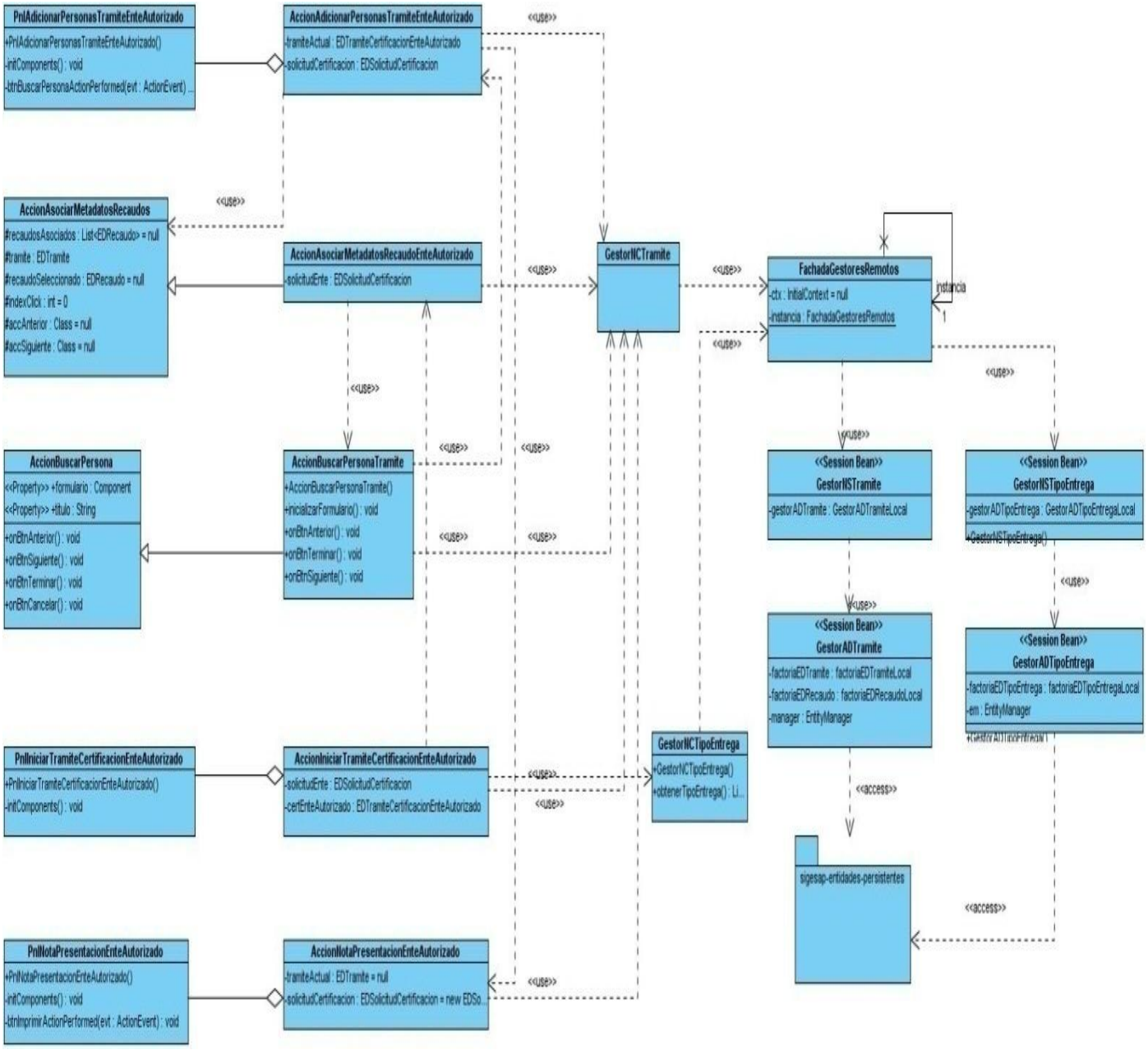

Fig. 2 Diagrama de Clases del Diseño del Caso de Uso IniciarTramiteCertificacionEnteAutorizado

Nombre de la clase: AccionIniciarTramiteCertificacionEnteAutorizado	
Tipo de clase: Controladora	
Descripción: Clase acción que controla las funcionalidades del formulario PnlIniciarTramiteCertificacionEnteAutorizado**.**	
Atributos	**Tipo**
solicitudEnte	EDSolicitudCertificacion
certEnteAutorizado	EDTramiteCertificacionEnteAutorizado
Responsabilidades de la clase:	
Nombre:	inicializarFormulario()
Descripción	Permite cargar el formulario correspondiente y llamar a los métodos.
Nombre:	getFormulario()
Descripción:	Obtiene un formulario específico.
Nombre:	onBtnSiguiente()
Descripción:	Va al formulario siguiente.
Nombre:	onBtnAnterior()
Descripción:	Regresa al formulario anterior.
Nombre:	onBtnTerminar()
Descripción:	Para volver a la acción principal.
Nombre:	obtenerTipoEntrega()
Descripción:	
Nombre:	cargarDatosFormulario()
Descripción:	

Nombre:	llenarSolicitud()
Descripción:	

Tabla 2. Descripción de la clase ***AccionIniciarTramiteCertificacionEnteAutorizado****.*

Nombre de la clase: AccionAdicionarPersonaErrorDatos	
Tipo de clase: Controladora	
Descripción: Clase acción que controla las funcionalidades del formulario PnlAdicionarPersonaErrorDatos.	
Atributos	**Tipo**
esIndocumentado	**boolean**
Responsabilidades de la clase:	
Nombre:	inicializarFormulario()
Descripción	Permite cargar el formulario correspondiente y llamar a los métodos.
Nombre:	getFormulario()
Descripción	Obtiene un formulario específico.
Nombre:	onBtnSiguiente()
	Va al formulario siguiente.
Descripción	
Nombre:	onBtnAnterior()
Descripción	Regresa al formulario anterior.
Nombre:	onBtnTerminar()
Descripción	Para volver a la acción principal.

Nombre	obtenerPersona()
Descripción	Obtiene la persona de la cual se pasaron los datos por parámetros.

Tabla 4. Descripción de la clase ***AccionAdicionarPersonaErrorDatos.***

Nombre de la clase: AccionAdicionarPersonaTramiteEnteAutorizado	
Tipo de clase: Controladora	
Descripción: Clase acción que controla las funcionalidades del formulario PnlAdicionarPersonaTramiteEnteAutorizado.	
Atributos	**Tipo**
tramiteActual	EDTramiteCertificacionEnteAutorizado
solicitudCertificacion	EDSolicitudCertificacion
idIndocumentado	int
Responsabilidades de la clase:	
Nombre:	inicializarFormulario()
Descripción	Permite cargar el formulario correspondiente y llamar a los métodos.
Nombre:	getFormulario()
Descripción	Obtiene un formulario específico.
Nombre:	onBtnSiguiente()
Descripción:	Va al formulario siguiente.
Nombre:	onBtnAnterior()
Descripción	Regresa al formulario anterior.

Nombre:	onBtnTerminar()
Descripción	Para volver a la acción principal.
Nombre	buscarPersona()
Descripción	Este método llama a la siguiente acción en el flujo de trabajo, busca las personas a añadir al trámite para ente autorizado.
Nombre	recibirPersona (EDPersona personaBuscar)
Descripción	
Nombre	recibirSolicitud (EDSolicitudCertificacion solicitud, EDTramiteCertificacionEnteAutorizado tramite)
Descripción	
Nombre	recibirTramite(EDTramiteCertificacionEnteAutorizado tramite)
Descripción	
Nombre:	registrarSolicitud()
Descripción:	
Nombre:	onEliminarPersonas()
Descripción:	
Nombre:	onCambiarEstadoSeleccionados()
Descripción:	
Nombre:	onRegistrarDatosError()
Descripción:	

Tabla 6. Descripción de la clase ***AccionAdicionarPersonaTramiteEnteAutorizado***

Nombre de la clase: AccionAdicionarPersonaTramiteEnteAutorizado	
Tipo de clase: Controladora	
Descripción: Clase acción que controla las funcionalidades del formulario PnlAdicionarPersonaTramiteEnteAutorizado.	
Atributos	**Tipo**
tramiteActual	EDTramite
solicitudCertificacion	EDSolicitudCertificacion
tramiteActual	EDTramite
Responsabilidades de la clase:	
Nombre:	inicializarFormulario()
Descripción:	Permite cargar el formulario correspondiente y llamar a los métodos.
Nombre:	getFormulario()
Descripción:	Obtiene un formulario específico.
Nombre:	onBtnSiguiente()
Descripción:	Va al formulario siguiente.
Nombre:	onBtnAnterior()
Descripción:	Regresa al formulario anterior.
Nombre:	onBtnTerminar()
Descripción:	Para volver a la acción principal.
Nombre:	recibirTramite(EDTramite tramite)

Descripción:	Este método recibe de la acción AdicionarPersona el trámite en curso.
Nombre:	mostrarResumenSolicitud(EDSolicitudCertificacionsolicitud, EDTramite tramite)
Descripción:	
Nombre:	onVistaPrevia()
Descripción:	

Tabla 8. Descripción de la clase ***AccionAdicionarPersonaTramiteEnteAutorizado***

Nombre de la clase: AccionAsociarMetadatosRecaudoEnteAutorizado	
Tipo de clase: Controladora	
Descripción: Se relaciona con la acción AsociarMetadatosRecaudos, permitiendo entrar todos los datos necesarios para el trámite en curso.	
Atributos	**Tipo**
solicitudEnte	EDSolicitudCertificacion
Responsabilidades de la clase:	
Nombre:	inicializarFormulario()
Descripción:	Permite cargar el formulario correspondiente y llamar a los métodos.
Nombre:	getFormulario()
Descripción:	Obtiene un formulario específico.
Nombre:	onBtnSiguiente()
Descripción:	Va al formulario siguiente.

Nombre:	onBtnAnterior()
	Regresa al formulario anterior.
Descripción:	
Nombre:	recibirSolicitud(EDSolicitudCertificacion solicitudEnte, EDTramiteCertificacionEnteAutorizado tramite)
Descripción:	
Nombre:	mostrarFecha()
Descripción:	

Tabla 9. Descripción de la clase ***AccionAsociarMetadatosRecaudoEnteAutorizado*****.**

Nombre de la clase: AccionBuscarPersonaTramite	
Tipo de clase: Controladora	
Descripción: Clase que se utiliza para buscar una persona y asociarla al trámite en curso.	
Responsabilidades de la clase:	
Nombre:	inicializarFormulario()
Descripción:	Permite cargar el formulario correspondiente y llamar a los métodos.
Nombre:	getFormulario()
Descripción:	Obtiene un formulario específico.
Nombre:	onBtnSiguiente()
Descripción:	Va al formulario siguiente.
Nombre:	onBtnAnterior()

Descripción:	Regresa al formulario anterior.	

*Tabla 10. Descripción de la clase **AccionBuscarPersonaTramite**.*

2.10 Conclusiones

A través de la realización del modelo de diseño, se logró:

- Agilizar el proceso de desarrollo, generando solo la documentación necesaria que sirviera de guía para la construcción del software.
- Se desarrollaron varios artefactos como: modelo de diseño y modelo de datos, quedando la aplicación lista para entrar al flujo de pruebas.

Capítulo 3: Implementación y Prueba

En el presente capítulo se describe cómo será implementado el sistema, a través de los estándares de codificación y diagramas de componentes.

También resumirá el conjunto de pruebas realizadas al Módulo Certificación para comprobar la satisfacción de los requisitos funcionales y no funcionales asociados al mismo. Se abordarán los tipos de pruebas realizadas, resultados obtenidos en las mismas y la evaluación de esos resultados.

3.1 Estándares de codificación

Los estándares de codificación son un estilo de programación homogénea en un proyecto. Estos son de vital importancia durante la etapa de construcción del software ya que permite que todo el personal del proyecto pueda entender de forma fácil el código garantizándose la organización y estructura del código fuente. Es decir, los estándares de codificación son un conjunto de reglas a seguir por los desarrolladores con el objetivo de establecer un orden y un formato común en el código fuente del software en desarrollo, a continuación se muestran ejemplos de estos estándares aplicados en el proyecto. (RAP, 2011)

Cada fichero fuente Java contiene una única clase o interface pública. Cuando algunas clases o interfaces privadas están asociadas a una clase pública, pueden ponerse en el mismo fichero que la clase pública.

La clase o interfaz pública debe ser la primera clase o interface del fichero.

Los ficheros fuentes Java tienen la siguiente ordenación:

- Comentarios de comienzo
- Sentencias package e import
- Declaraciones de clases e interfaces

Comentarios de comienzo

Todos los ficheros fuente deben comenzar con un comentario en el que se lista el nombre de la clase, información de la versión, fecha, y copyright:

Declaraciones de clases e interfaces

La siguiente lista describe las partes de la declaración de una clase o interface, en el orden en que deberían aparecer.

- Comentario de documentación de la clase o interface (/**...*/)
- Sentencia class o interface
- Comentario de implementación de la clase o interface si fuera necesario (/*...*/): Este comentario debe contener cualquier información aplicable a toda la clase o interface que no era apropiada para estar en los comentarios de documentación de la clase o interface.
- **Variables de clase (static):** Primero las variables de clase public, después las protected, después las de nivel de paquete (sin modificador de acceso), y después las private.
- **Variables de instancia**: Primero las public, después las protected, después las de nivel de paquete (sin modificador de acceso), y después las private.
- **Constructores**
- **Métodos:** Estos métodos se deben agrupar por funcionalidad más que por visión o accesibilidad. Por ejemplo, un método de clase privado puede estar entre dos métodos públicos de instancia. El objetivo es hacer el código más legible y comprensible.

Al codificar clases e interfaces de Java, se siguen las siguientes reglas de formato:

- Ningún espacio en blanco entre el nombre de un método y el paréntesis "(" que abre su lista de parámetros
- La llave de apertura "{" aparece al final de la misma línea de la sentencia declaración
- La llave de cierre "}" empieza una nueva línea indentada para ajustarse a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{"

```
class Ejemplo extends Object {
int ivar1;
int ivar2;
Ejemplo (int i, int j) {
ivar1 = i;
ivar2 = j;
```

```
}
int metodoVacio () {}
...
}
```

Los métodos se separan con una línea en blanco

- **Sentencias if, if-else, if else-if else**

La clase de sentencias if-else debe tener la siguiente forma:

```
if (condicion) {
sentencias;
}
if (condicion) {
sentencias;
} else {
sentencias;
}
if (condicion) {
sentencia;
} else if (condicion) {
sentencia;
} else {
sentencia;
}
```

3.2. Diagramas de componente

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones.

Una característica de estos diagramas con respecto a otros tipos de diagramas es sin duda su contenido, normalmente contienen componentes, interfaces y relaciones entre ellos. Los componentes pertenecen a un mundo físico, es decir, representan a un bloque de construcción al modelar aspectos físicos de un sistema. Cada uno debe tener un nombre que lo distinga de los demás.

Dada las características de la arquitectura del sistema el diagrama de componentes propuesto cuenta con un Cliente, el cual contendrá los diferentes gestores que atenderán las solicitudes realizadas por el usuario, estos harán uso de la interfaz FachadaGestorRemoto quien hará las

peticiones específicas a los gestores que se encuentran en el Servidor, el cual se encarga de gestionar la lógica de negocio del sistema.

Los gestores del Servidor harán la petición de la información solicitada por el usuario a los gestores pertenecientes a la capa AccesoDatos los cuales buscarán los datos correspondientes a la información solicitada en el paquete de ClasesPersistentes. Una vez obtenido los datos necesarios, cada capa le entregará a su capa superior la información solicitada.

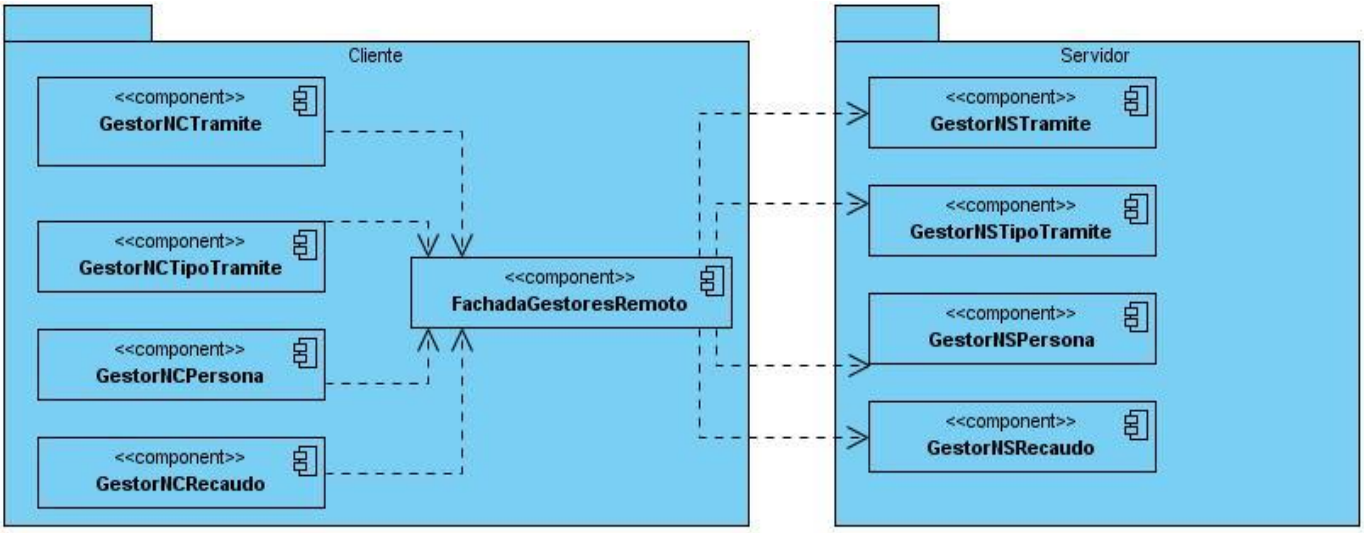

Fig. 3 Diagrama de componente. Vista Negocio

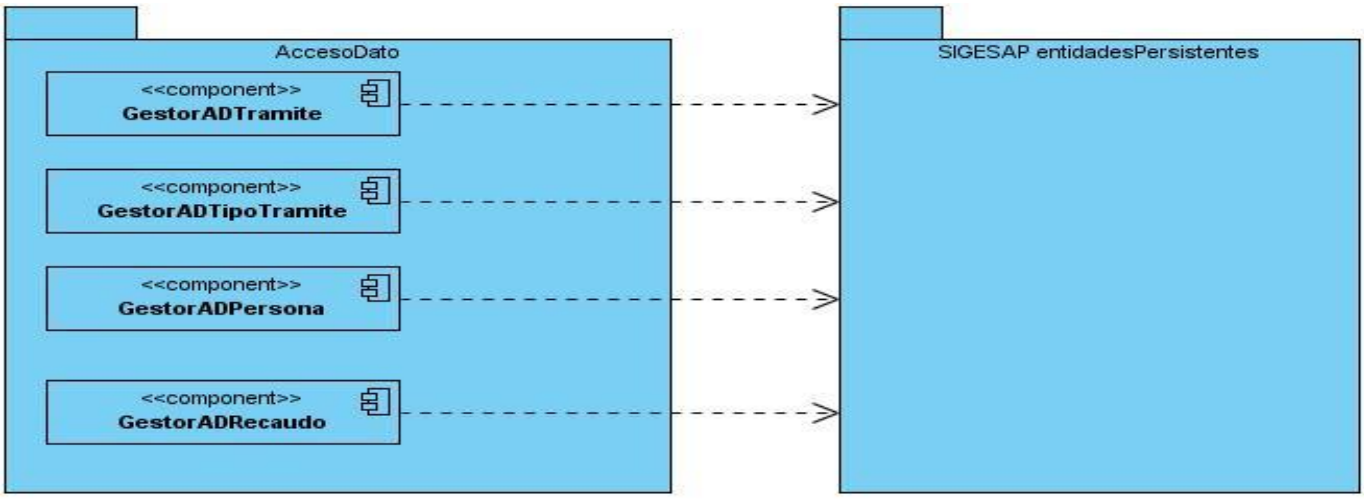

Fig. 3 Diagrama de componente. Vista Acceso a Dato

Capítulo 3: Implementación y Pruebas

3.3. Pruebas

El único instrumento adecuado para determinar el status de la calidad de un producto software es el proceso de pruebas. En este proceso se ejecutan pruebas dirigidas a componentes del software o al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos. En las pruebas se usan casos de prueba, especificados de forma estructurada mediante Técnicas de prueba. (PRUEBASDESOFTWARE, 2005)

3.3.1 Validación del diseño

- **Tamaño operacional de clase (TOC)**

Las métricas empleadas están diseñadas para evaluar los siguientes atributos de calidad:

Responsabilidad: Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.

Complejidad de implementación: Consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.

Reutilización: Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

Tabla 1- Atributos de calidad evaluados por la métrica TOC.

Atributo de Calidad.	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 en la complejidad de implementación de la clase.
Reutilización.	Un aumento del TOC implica una disminución del grado de

	reutilización de la clase.

Para los cuales están definidos los siguientes criterios y categorías de evaluación:

Tabla 2- Criterios de evaluación para la métrica TOC.

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

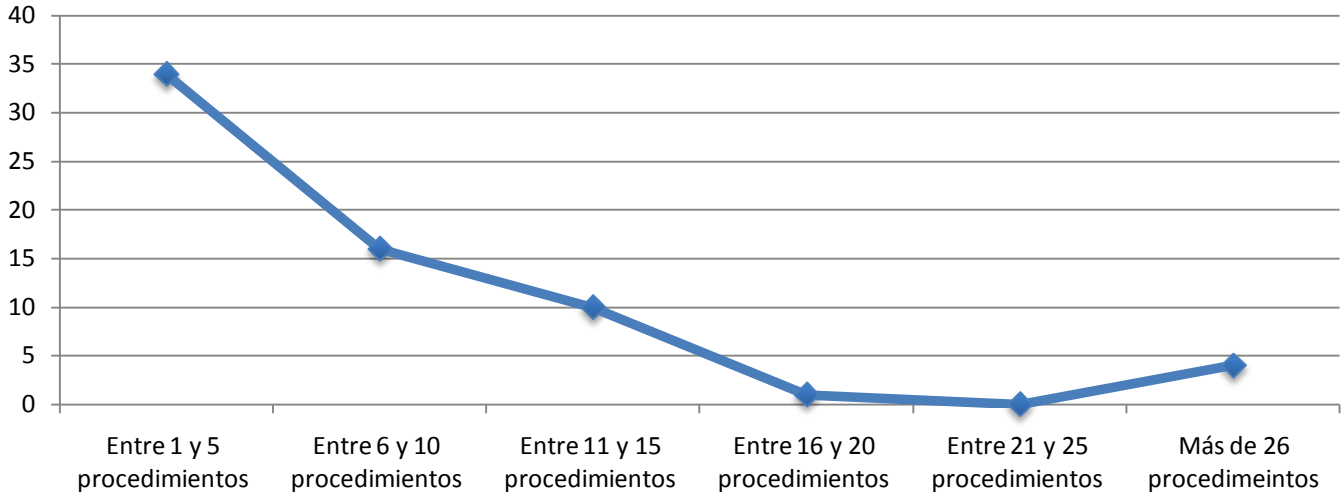

Fig. 4 Representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

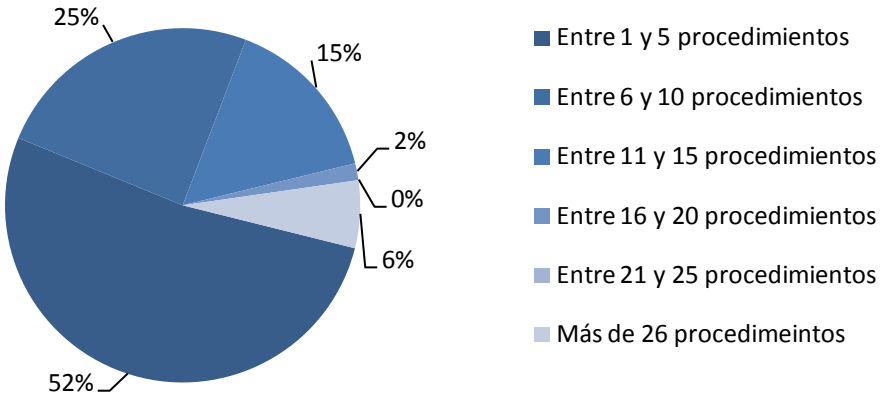

Fig. 5 Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

Representación de la incidencia de los resultados de la evaluación de la métrica TOC en cada uno de los atributos:

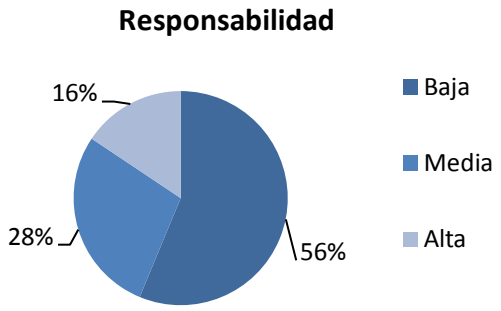

Fig. 6 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad.

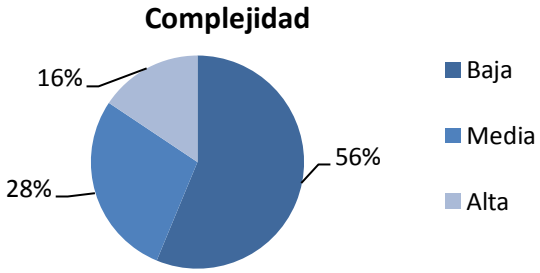

Fig. 7 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación.

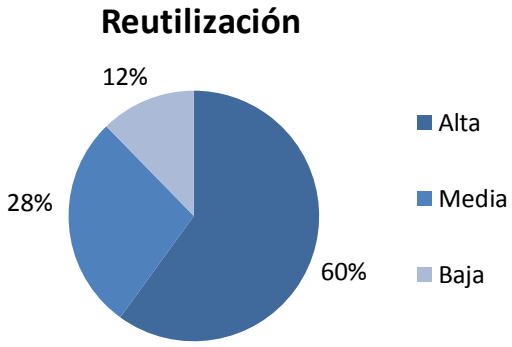

Fig. 8 - Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Al analizar los resultados obtenidos luego de aplicar el instrumento de medición de la métrica TOC, se puede concluir que el diseño propuesto para el módulo de certificación está entre los límites aceptables de calidad, teniendo en cuenta que la mayoría de las clases (60%) posee menos cantidad de operaciones que la media registrada en las mediciones. Los atributos de calidad se encuentran en un nivel medio satisfactorio en el 60% de las clases; de manera que se puede observar cómo se fomenta la Reutilización (elemento clave en el proceso de desarrollo de software) y cómo están reducidas en menor grado la Responsabilidad y la Complejidad de implementación.

- **Relaciones entre clases (RC)**

Con la presente métrica se evalúan los siguientes atributos de calidad:

Responsabilidad: Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.

Complejidad del mantenimiento: Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.

Reutilización: Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

Cantidad de pruebas: Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (componente, modulo, clase, conjunto de clases) diseñado.

Tabla 3: Atributos de calidad evaluados por la métrica RC.

Atributo de Calidad.	Modo en que lo afecta.
Responsabilidad.	Un aumento del RC implica un aumento de la responsabilidad asignada a la clase.
Complejidad del mantenimiento.	Un aumento del RC implica un aumento en la complejidad del mantenimiento de la clase.

Reutilización.	Un aumento del RC implica una disminución del grado de reutilización de la clase.
Cantidad de pruebas.	Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Para los cuales están definidos los siguientes criterios y categorías de evaluación:

Tabla 3- Criterios de evaluación de la métrica RC.

Atributo.	Categoría.	Criterio.
Acoplamiento.	Ninguno.	0
	Bajo.	1
	Medio.	2
	Alto.	>2
Complejidad de mantenimiento.	Baja.	<=Promedio
	Media.	Entre Promedio y 2*Promedio
	Alta.	>2*Promedio
Reutilización.	Baja.	>2*Promedio
	Media.	Entre Promedio y 2*Promedio
	Alta.	<=Promedio
Cantidad de pruebas.	Baja.	<=Promedio
	Media.	Entre Promedio y 2*Promedio
	Alta.	>2*Promedio

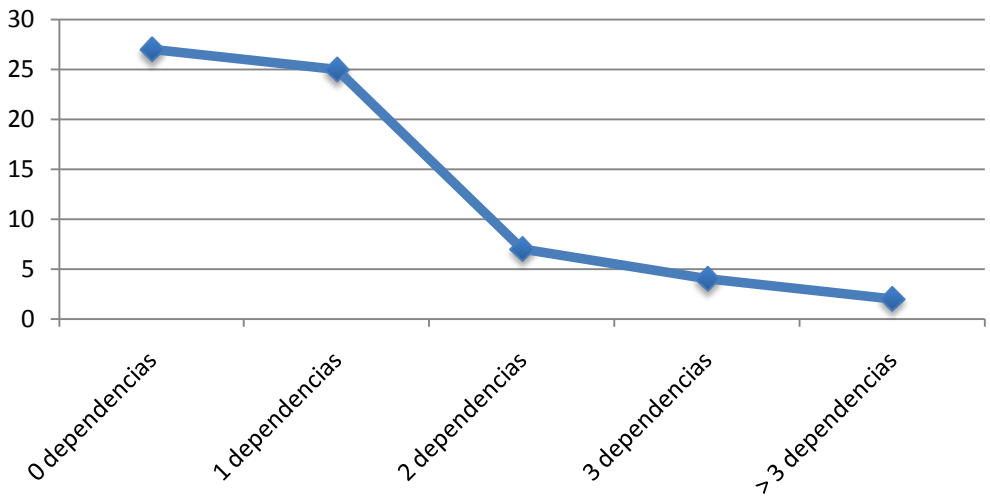

Fig. 9 Gráfica de los resultados de la evaluación de la métrica RC agrupados por la tendencia de los Valores.

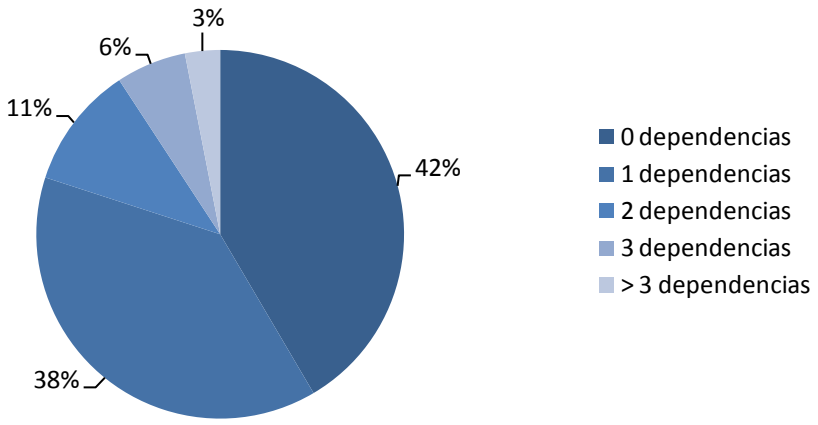

Fig. 10 Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

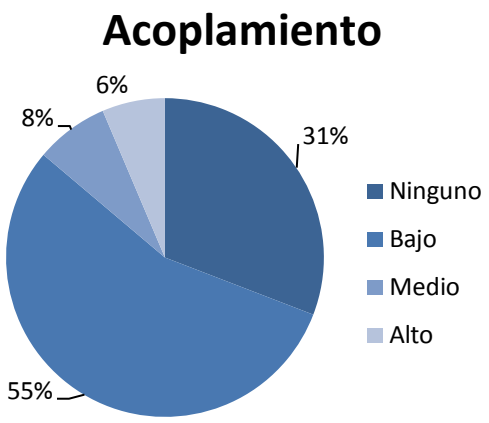

Fig. 11 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.

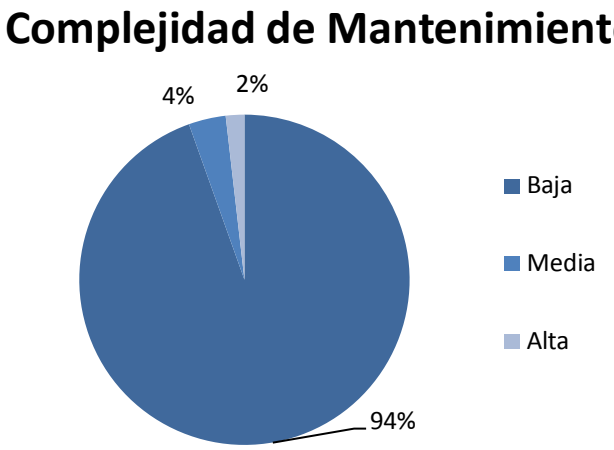

Fig. 12 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento.

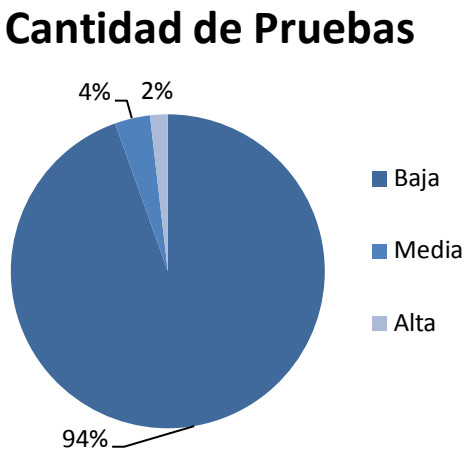

Fig. 13 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

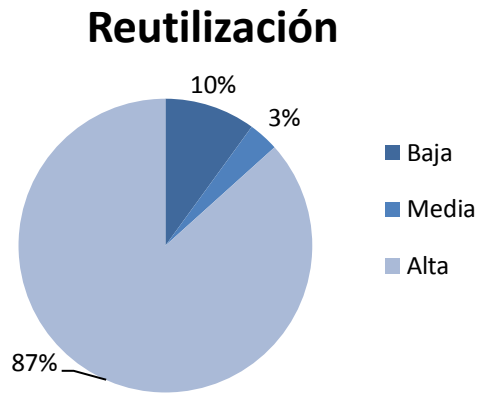

Fig. 14 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización.

Al analizar los resultados obtenidos luego de aplicar el instrumento de medición de la métrica RC, se puede concluir que el diseño propuesto para el módulo de certificación está entre los límites aceptables de calidad, teniendo en cuenta que la mayoría de las clases (53%) poseen menos de 3 dependencias respecto a otras. Los atributos de calidad se encuentran en un nivel satisfactorio; en el 55% de las clases el grado de dependencia o acoplamiento es mínimo, la Complejidad de Mantenimiento, la Cantidad de

Pruebas y la Reutilización se comportan favorablemente para un 94%, 94% y 87% de las clases respectivamente.

3.3.2. Casos de prueba

Una vez generado el código fuente, el software debe ser probado para revelar y corregir el máximo de errores posible antes de su entrega al cliente, haciéndolo con la menor cantidad de tiempo y esfuerzo posible. (Pressman, 2007)

“Las pruebas del software son un elemento para la garantía de la calidad del software y representan una revisión final de las especificaciones, del diseño y de la codificación” (Pressman, 2007)

Como objetivo fundamental de la prueba según Pressman en la quinta edición del libro “Ingeniería de Software un enfoque práctico” se tiene el siguiente: descubrir la máxima cantidad de errores no detectados hasta entonces. Aun así, es necesario señalar que las pruebas no pueden asegurar la ausencia de errores, solo puede demostrar que existen defectos en el software. (Pressman, 2007)

Desde años anteriores, han surgido y desarrollado una variedad de métodos para efectuar pruebas de software. Los más significativos en este contexto son los de prueba de caja blanca y caja negra. (Velasco Elizondo, 2001)

En el caso de la presente investigación se decidió aplicar el método de caja negra.

Pruebas de caja negra: Las pruebas de caja negra, también denominada prueba de comportamiento, se centran en los requisitos funcionales, permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Este tipo de prueba intenta descubrir diferentes errores que los métodos de caja blanca no logran identificar. Las categorías de los errores encontrados por este tipo de prueba son: funciones incorrectas o ausentes, errores de interfaz, estructuras de datos, rendimiento, inicialización y terminación. (Pressman, 2007)

Las pruebas de caja negra se llevan a cabo sobre la interfaz del software, y es completamente indiferente el comportamiento interno y la estructura del programa.

La prueba de la caja negra intenta encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes.

- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

Para confeccionar los casos de prueba de Caja Negra existen distintas técnicas. Algunas de ellas son:

- Particiones de Equivalencia: Es un método de prueba de Caja Negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Esta se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada.
- Análisis de Valores Límite: El Análisis de Valores Límites (AVL) es una técnica de diseño de casos de prueba que complementa la partición equivalente. En lugar de centrarse solamente en las condiciones de entrada, el AVL obtiene casos de prueba también para el campo de salida.
- Métodos Basados en Grafos: Empieza creando un grafo de objetos importantes y sus relaciones, y después diseñando una serie de pruebas que cubran el grafo de manera que se ejerciten todos los objetos y sus relaciones para descubrir los errores.
- Guiada por Casos de Prueba: Verifican las especificaciones funcionales y no consideran la estructura interna del programa. Se realiza sin el conocimiento interno del producto. No validan funciones ocultas por tanto los errores asociados a ellas no serán encontrados.

Con el objetivo de comprobar el correcto funcionamiento de las funcionalidades correspondientes al Módulo de Reportes, el software fue sometido a una serie de pruebas a través de las cuales se pudo corroborar que el sistema responde positivamente a toda actividad que el usuario desee realizar sobre ella. A continuación se muestran los resultados obtenidos al aplicar las pruebas de caja negra a la aplicación.

Para el caso de uso Iniciar Tramite Certificación para Ente Autorizado:

Secciones a probar en el Caso de Uso

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC 1: Iniciar Trámite de Certificación Ente Autorizado.	EC 1.1 Introducir correctamente datos iniciales para la solicitud de certificación Ente Autorizado.	Se introducen los datos iniciales para crear una solicitud de Ente Autorizado, el sistema activa el botón siguiente y se continúa con el flujo.
	E.C 1.2: No se introducen los datos iniciales para la solicitud de certificación Ente Autorizado.	Se deja vacío el tipo de entrega y el sistema no activa el botón siguiente, no permitiendo avanzar en el flujo.
SC 2: Asociar metadatos a recaudos.	E.C 2.1: Ver Diseño de Caso de Prueba Asociar metadatos a recaudos.	
SC 3: Añadir personas a la solicitud.	EC 3.1: Ver Diseño de Caso de Prueba Buscar Persona.	
SC 4: Registrar persona con error.	EC 4.1: Ver Diseño de Caso de Prueba Registrar Indocumentado.	
SC 5: Asociar persona a la solicitud de certificación.	EC 5.1: Selección de una o varias personas correctamente.	Se seleccionan una o varias personas de las obtenidas en el resultado de la búsqueda pudiendo posteriormente el sistema mostrar un mensaje de confirmación para registrar la solicitud.
	EC 5.2: No se selecciona al menos una persona.	El sistema muestra un mensaje de aviso, no permitiendo registrar la solicitud.

	EC 5.3: Cancelar.	El sistema muestra un mensaje de confirmación para cancelar el flujo actual.
SC 6: Registrar una nueva solicitud.	EC 6.1: Seleccionar la opción registrar.	El sistema muestra un mensaje de confirmación crear una nueva solicitud en el sistema. Luego un mensaje informando la correcta creación de la solicitud.
	EC 6.2: Mostrar resumen del trámite.	El sistema muestra una interfaz que contiene el resumen de los datos que conforman la solicitud: • Datos del ente. • Lista de Personas incluidas en la solicitud.
SC 7: Imprimir Nota de Presentación.	EC 7.1: Imprimir.	El sistema verifica la existencia de impresora conectada, si no existe muestra un mensaje de error.

Descripción de variable

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Tipo de entrega	Lista desplegable	No	Nomenclador
2	Prioridad	Campo de Texto	No	Sólo admite números.

Matriz de datos

Escenario	Tipo de entrega	Prioridad	Respuesta del Sistema	Resultado de la Prueba	Flujo Central
EC 1.1: Introducir datos iniciales para la solicitud de certificación Ente Autorizado.	*V* Alguacil	*V* 1	Se habilita el botón siguiente y se continúa con el flujo para insertar una solicitud.	Satisfactorio.	1 Certificación. 2 Iniciar trámite para ente autorizado.
E.C 1.2: No se introducen los datos iniciales para la solicitud de certificación Ente Autorizado.	*V* (Vacio)	*V* 2	No se habilita el botón siguiente y imposibilitando continuar con el flujo para insertar una solicitud.	Satisfactorio.	1 Certificación. 2 Iniciar trámite para ente autorizado.

3.4. Resultados de las pruebas

Se analizaron los casos de prueba aplicados al sistema para verificar los niveles de calidad con que el mismo fue realizado. Los resultados obtenidos fueron satisfactorios.

A continuación se muestran algunas estadísticas del proceso:

Cantidad de casos de prueba	12
Cantidad de personas involucradas	6

Cantidad de no conformidades	15
Etapas en las que se realizó el proceso	3

Las no conformidades giraron en torno a los siguientes temas:

- Selección de múltiples elementos, donde solo debía ser único.
- Errores de concordancia en los mensajes mostrados al usuario.
- Excepciones no controladas.
- Ausencia de algunos iconos.

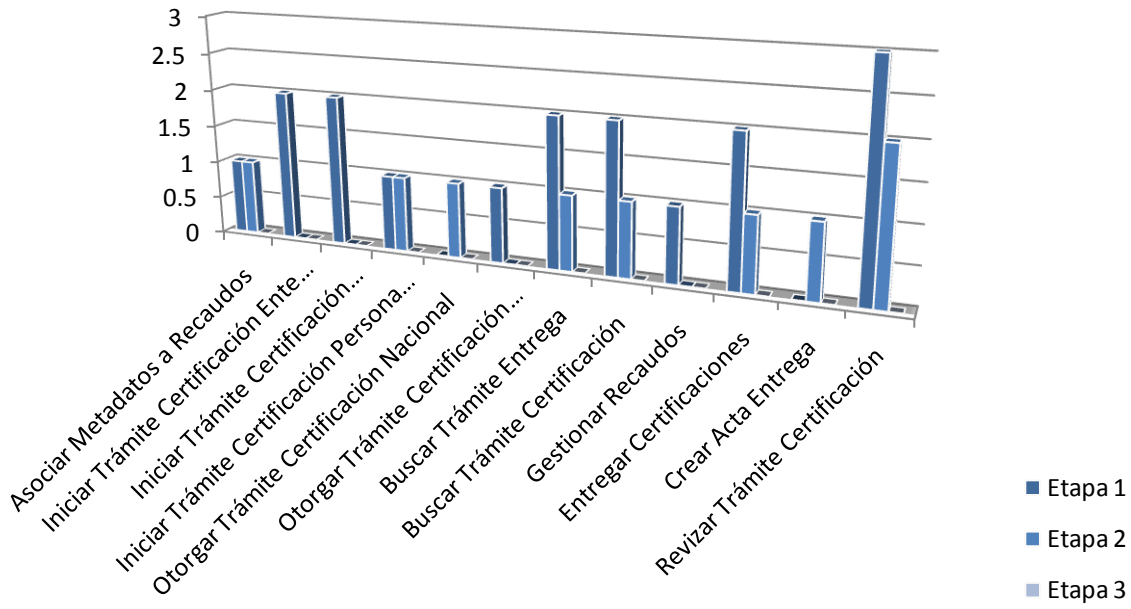

Fig. 2 Representación de no conformidades por etapas

Los artefactos generados serán entregados al equipo de calidad del Centro para su revisión y aprobación con vistas a alcanzar como resultado el acta de liberación de los artefactos del trabajo de investigación.

Capítulo 3: Implementación y Pruebas

3.5. Conclusiones

En el presente capítulo se abordaron los temas que hacen alusión a la implementación de la solución y a las pruebas realizadas al software obtenido. Permitiendo:

- ✓ Mostrar los diagramas utilizados por el equipo de desarrollo para representar las distintas partes del sistema.
- ✓ Demostrar que las pruebas de caja negra son una estrategia fiable para obtener software probado y funcional.
- ✓ Evaluar satisfactoriamente el diseño realizado para el módulo de certificación.

Conclusiones

Al finalizar el presente trabajo de diploma se arriba a las siguientes conclusiones:

- Se obtuvieron los artefactos necesarios para las etapas de diseño de acuerdo a la metodología de desarrollo utilizada en el proyecto aplicando una estrategia para la construcción de un diseño flexible y escalable a partir de la implantación de patrones, arquitectura y estándares.
- Se logró la implementación de todos los casos de uso de acuerdo al diseño elaborado previamente y se generaron el resto de los artefactos definidos para este flujo de trabajo.
- Se comprobó el nivel de calidad que presenta el diseño a partir de la aplicación de métricas para el modelo de diseño orientado a objetos y el análisis de sus resultados.
- A través de las pruebas realizadas al software se obtuvieron resultados satisfactorios que validaron la calidad del sistema.
- Se logró el objetivo general del trabajo de diploma: Diseño e implementación del módulo de certificación de Antecedentes Penales de la Solución Tecnológica Integral de la División de Antecedentes Penales de la República Bolivariana de Venezuela.

Recomendaciones

Recomendaciones

- Migrar próximas versiones a las versiones superiores de Postgre SQL.
- Firmar en línea los trámites en proceso de otorgamiento.
- Aplicar las experiencias obtenidas en el desarrollo de aplicaciones similares.

Bibliografía

1. **Amoroso, Yarina. 2009.** *Apuntes para la modernizacion e informatizacion del Sistema Registral Venezolano.* Venezuela : s.n., 2009.
2. **Autores, Colectivo de. 2011.** java-swing. [En línea] 2011. http://www.scribd.com/doc/50762735/java-swing.
3. **—. 2005.** Terminologia Archivística Ensayos y Documentos. [En línea] 2005. http://www.buenastareas.com/search_results.php?query=documento+legal.
4. **Barriento, Manuel Sánchez. 2008.** BPMN desventajas. [En línea] 2 de Noviembre de 2008. [Citado el: 9 de Febrero de 2010.]
5. **Becerril, José Francisco. 1998.** *Java a su alcance.* México : Mc. Graw-Hill Iteroamericana, 1998.
6. **Caro, Patricio Salinas. 2007.** [En línea] 2007. http://www.dcc.uchile.cl/~psalinas/uml/introduccion.html.
7. **Collector, Garbage. 2004.** www.error500.net. [En línea] 1 de noviembre de 2004. [Citado el: 14 de junio de 2011.] http://www.error500.net/garbagecollector/archives/categorias/bases_de_datos/sistema.
8. **Corporation, Oracle. 2010.** [En línea] 2010. http://www.mysql.com/.
9. **2007.** *El proceso unificado de modelado. Manual de referencia (Segunda edición) .* Madrid : Addison Wesley, 2007.
10. **2010.** *Especificación de requisitos de software. Módulos de Inscripción, Certificación, Archivo, Reportes, Digitalización.* 2010.

Bibliografía

11. **Félix Prieto, Yania Crespo, José Manuel Marqués, Miguel Angel Laguna. 2000.** *Mecanos y Análisis de Conceptos Formales como soporte para la construcción de Frameworks.* Valladolid y Ciudad de la Habana : s.n., 2000.

12. **Hanrahan, Robert P. 1995.** *The IDEF Process Modeling Methodology.* s.l. : Software Technology Support Center, 1995.

13. **2009.** *Informatizamos para la sociedad cubana.* s.l. : Desoft Division de Servicios Generales, 2009.

14. **Innova, Soluciones. 2007.** www.rational.com.ar. [En línea] 2007. [Citado el: 14 de junio de 2011.] http://www.rational.com.ar/herramientas/roseenterprise.html.

15. **Integrado, Eclipse Entorno. 2010.** Eclipse Entorno Integrado. [En línea] 13 de 3 de 2010. [Citado el: 13 de junio de 2011.] http://www.thehouseofblogs.com/articulo/entorno_integrado_de_desarrollo_ide_modular-30126.html.

16. **ISW, Colectivo DDC. 2010.** Ingeniería de Software II. [En línea] 2010. [Citado el: 27 de Febrero de 2010.]

17. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000.** *El proceso unificado de desarrollo de software.* s.l. : Addison Wesley, 2000.

18. **JDeveloper. 2011.** JDeveloper. [En línea] 10 de 02 de 2011. [Citado el: 14 de junio de 2011.] http://www.scribd.com/doc/49493122/manual-de-ayuda-jdeveloper-3.

19. **2010.** Kernel Doc. [En línea] 2010. http://www.kerneldoc.es/documentos/presentacion.pdf.

20. **Kevin, Loney. 2008.** Oracle Database 11g The Complete Reference (1st ed.). [En línea] 2007 de noviembre de 2008. [Citado el: 15 de abril de 2011.]

21. **Larman, Craig. 2000.** *UML y Patrones.* 2000.

22. **Lockhart, Thomas. 2001.** *Tutorial de PostgreSQL.* 2001.

Bibliografía

23. **Miranda Gonzalez, Francisco Javier, Chamorro Mera, Antonio y Rubio Lacoba, Sergio. 2005.** *Clarificando el concepto de certificació: El caso español .* España : s.n., 2005.

24. **2001.** *Modelo de decisión para soportar la selección de herramientas Case.* 2001.

25. **Mosquera, Juan Diego Diaz. 2008.** *Herramientas CASE.* 2008.

26. **Netbeans. 2011.** Netbeans. [En línea] 03 de 03 de 2011. http://netbeans.org/index_es.html..

27. **Orallo, Enrique Hernández. 2007.** El lenguaje Unificado de Modelado (UML). *El lenguaje Unificado de Modelado (UML).* [En línea] 2007. [Citado el: 12 de Febrero de 2010.]

28. **Pressman, Roger. 2007.** *Ingeniería de Software un enfoque práctico (Quinta edición).* 2007.

29. **Programacion, Lenguaje de. 2010.** Lenguaje de Programacion. [En línea] 2010. [Citado el: 14 de junio de 2011.] http://www.buenastareas.com/ensayos/Lenguaje-De-Programacion/1481199.html.

30. **PRUEBASDESOFTWARE. 2005.** Gestión de Calidad y Pruebas de Software. [En línea] 2005. [Citado el: 31 de 3 de 2011.] http://www.pruebasdesoftware.com/laspruebasdesoftware.htm.

31. **Públicas, Ministerio de Administraciones. 2010.** www.scribd.com. [En línea] 2010. [Citado el: 30 de 3 de 2011.] http://www.scribd.com/doc/49900890/14/Diagrama-de-Paquetes..

32. **RAP, Repositorio del proyecto. 2011.** *Estándares de codificación.* Habana : s.n., 2011.

33. **Rodríguez, José Ernesto Lara. 2010.** *Análisis, diseño e implementación de un IDE Online.* 2010.

34. **Sanchez, Maria A. Mendoza. 2004.** *Metodologia de desarrollo de software.* 2004.

35. **Sierra, Daniel. 2011.** www.slideshare.net. [En línea] 2011. [Citado el: 14 de junio de 2011.] http://www.slideshare.net/vanquishdarkenigma/visual-paradigm-for-uml.

36. **Sparxsystems. 2008.** www.sparxsystems.com. [En línea] 2008. [Citado el: 14 de junio de 2011.] http://www.sparxsystems.com.ar/.

Bibliografía

37. **Urbino, Rafael Jacobo Hidalgo. 2010.** *Análisis, diseño e implementación de una herramienta que permita la centralización de la información gestionada por el módulo Resultados de la Colección Multisaber.* 2010.

38. **Vargas, Alan.** [En línea] http://blogs.sun.com/AlanVargas/entry/qu%C3%A9_es_glassfish.

39. **Velasco Elizondo, Perla Inés. 2001.** *PRUEBA DE COMPONENTES DE SOFTWARE BASADAS EN EL MODELO DE JAVABEANS.* Tlaxcala, México : s.n., 2001.