Universidad de las Ciencias Informáticas

Facultad 15

Título: Análisis y Diseño del Módulo Índice de Peligrosidad Pre-delictiva del Proyecto Sistema de Gestión Fiscal.

Trabajo de Diploma para optar por el título de Ingeniero Informático.

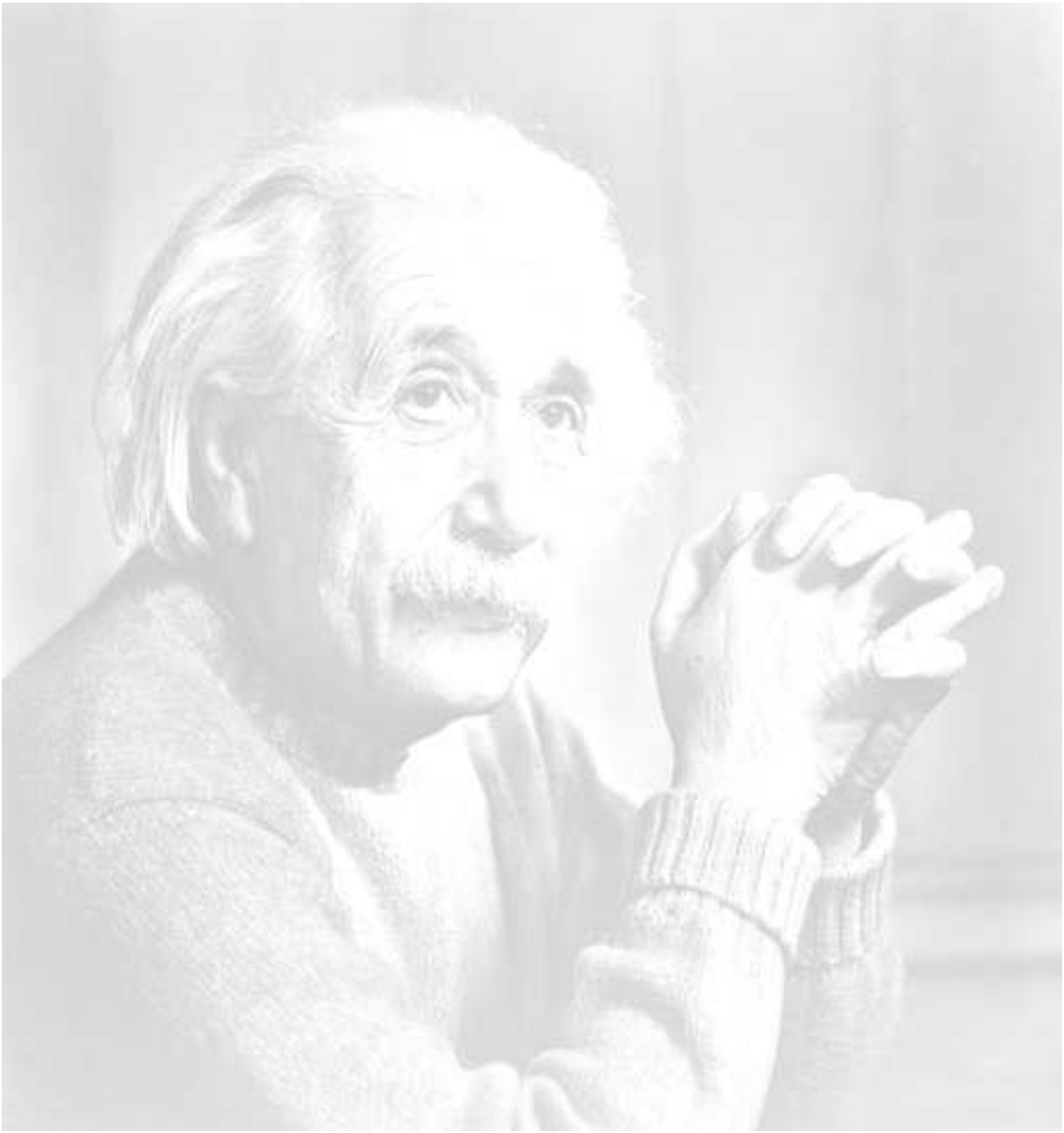
Autor(s): Abraham Michel Parra Parra.

Alexis David Pérez Escobar

Tutor: Ing. Maylin Bacallao Martínez.

Asesor: Esp. Belkis Cabrera Pérez.

Curso 2009-2010



“Nunca consideres el estudio como una obligación sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber”

Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al <nombre área> de la Universidad de las Ciencias Informáticas; así como a dicho centro para que hagan el uso que estimen pertinente con este trabajo.

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

_____ _____

Alexis David Pérez Escobar **Abraham Michel Parra Parra**

_____

Maylin Bacallao Martínez

DATOS DEL CONTACTO

Tutora:

Nombre: Ing. Maylin Bacallao Martínez.

Ingeniero en Ciencias Informáticas, 3 años de experiencia laboral. Se ha desempeñado como jefe de módulo de IPP en el proyecto Sistemas de Gestión Fiscal (SGF) en el curso 2009-2010. Jefa de asignatura de Sistemas Operativos y Seguridad informática. Analista del proyecto Registros y Notarías.

Correo electrónico: mbacallao@uci.cu

Asesor:

Nombre: Esp. Belkis Cabrera Pérez.

Licenciada en Derecho, 18 años de experiencia laboral. Especialista en Derecho Penal de la Dirección de Procesos Penales de la Fiscalía General de la República (FGR).

Correo electrónico: belkis@fgr.cu

Autor:

Nombre: Abraham Michel Parra Parra.

Correo electrónico: amparra@estudiantes.uci.cu

Autor:

Nombre: Alexis David Pérez Escobar.

Correo electrónico: adperez@estudiantes.uci.cu

DEDICATORIA

A mi madre por ser el Sol de mi vida. Te quiero mami.

A mi abuela Enma que sé, le habría encantado estar aquí conmigo, pero me está viendo desde el cielo.

AB.

A mis padres, y tendría que inventar nuevas palabras para poder describir todo lo que siento y lo agradecido que estoy y estaré por tenerlos, a mi familia en general y en especial a mi hermana que sé que le hubiese gustado mucho estar presente.

Alex.

AGRADECIMIENTOS

Agradezco en primer lugar a Dios por el regalo de la vida.
A mi madre por haberme guiado siempre y haber sido madre, padre y amiga. Por darme su amor incondicional, por siempre estar cuando la necesité, porque sin su apoyo no hubiera sido posible este resultado. Gracias mami.
A mis abuelos China y Librado por darme su apoyo siempre y especialmente a mi abuelo por haber sido mi padre en todo momento y haber sido un ejemplo para mí.
A mi esposa por estar en todo momento y darme fuerzas para seguir adelante y conseguir todo lo que me propuse.
A mis hermanos y amigos Rey, Yansel, Yuli, Yali e Ismaray por ser respaldo, ejemplo y un gran orgullo a lo largo de esta travesía en la que siempre han estado ahí. Los quiero Guapos.
A Pedro por estar ahí cada vez que lo necesité y cuando no también.
A mi compañero de tesis por dejarme ser parte de este trabajo tan grande, por prestarme sus conocimientos, paciencia y voluntad.
A Belkis por tener respuesta para cada pregunta, por tener la paciencia suficiente para explicar y volver a explicar, sin ella todo hubiera sido más difícil.
A mi tutora por darme su apoyo.
A todas las personas, familiares o no, que hayan hecho posible que hoy esté realizando mi sueño de ser ingeniero. Muchas gracias.

AB

A mi madre y a mi padre por ser mi razón de ser, por sus continuos consejos y por confiar y creer en mí siempre.
A mi familia por su apoyo incondicional a lo largo de todos estos años.
A nuestro Comandante en Jefe por la gran idea de crear la Universidad de las Ciencias Informáticas.
A mis amigos del Pre universitario, Keyler, Carlos Ramón, Yusdel, Dago, David, Ernesto, Yelenis y Eylis, por ser incondicionales.
A los nuevos amigos que he ido teniendo en el transcurso de estos 5 años, por su apoyo, confianza y consejos, no voy a mencionar nombre porque no quiero que se me quede ninguno.
A todos mis compañeros, de aula, apartamento, recreación, y deporte por hacer más activa mi estancia en la universidad.
A todos los profesores que han participado en mi desarrollo como estudiante, por brindarme sus conocimientos y experiencias.
A mi compañero de tesis por dejarme ser parte de este trabajo tan grande, por prestarme sus conocimientos, amistad, paciencia y voluntad.
A mi tutora por su apoyo y confianza depositada en mí durante el desarrollo de este trabajo.
A todas las personas que han aportado un granito de arena de una u otra forma.
Muchas gracias a todos.

Alex

RESUMEN

Como parte del esfuerzo que lleva adelante el gobierno cubano de informatizar paulatinamente todos los sectores de la sociedad, uno de los que ha sido beneficiado con esta política ha sido el sector jurídico. Formando parte de esta iniciativa se encuentra un proyecto de la Universidad de las Ciencias Informáticas encargado de desarrollar un sistema de gestión para la Fiscalía General de la República que pretende agilizar el proceso de toma de decisiones, eficiencia en el trabajo de los fiscales y elevar los niveles de conformidad por parte de los ciudadanos cubanos.

El propósito fundamental de este trabajo es realizar el modelado del sistema y del diseño del módulo Índices de Peligrosidad Pre-delictiva que es aplicable a personas que incurran en alguna de las siguientes clasificaciones: Narcomanía, Dipsomanía, Conducta antisocial, Enajenados mentales, Desarrollo mental retardado, Embriaguez habitual. Además de la validación de la propuesta presentada como garantías de la calidad del producto en construcción.

El documento consta de tres capítulos. El primero aborda el estudio del estado del arte del contenido teórico que se debe conocer para su realización. El capítulo dos presenta el modelo del negocio y del sistema y por último el tercer capítulo presenta el diseño del sistema así como la evaluación de la calidad de la propuesta.

Con la implementación de esta propuesta se espera que los fiscales tengan a su disposición un sistema que cumpla con sus expectativas, dotándolos de una aplicación que informatice y optimice los procesos Fiscales y el manejo de la información utilizada para su trabajo.

PALABRAS CLAVES

Casos de uso, diseño, métricas, patrones, requerimientos.

ÍNDICE DE CONTENIDO

INTRODUCCIÓN

El sistema Judicial y Legal de la República de Cuba se suscribe en las tradiciones y características del Derecho Continental Europeo del que tomó las correspondientes instituciones judiciales aun cuando, en su elaboración concreta y particular, tuvo en cuenta las condiciones sociales, culturales y jurídicas prevalecientes en la sociedad cubana contemporánea.

La Constitución de la República de Cuba, aprobada en el año 1976 en referéndum popular por el 97% de los ciudadanos de la nación, es la norma jurídica de mayor nivel y es la que determina los órganos con capacidad legislativa y los principios y fundamentos del contenido de las leyes.

Cuba es un Estado Socialista de Trabajadores, independiente y soberano, organizado con todos y para el bien de todos, como república unitaria y democrática, para el disfrute de la libertad política, la justicia social, el bienestar individual y colectivo y la solidaridad humana.

Cuba inició, a partir de 1970, un proceso de perfeccionamiento, modernización y sistematización de su sistema jurídico. Ese proceso complejo y continuo, se dirigió a todas las ramas del Derecho y puede aceptarse hoy que se ha creado un coherente conjunto de normas que son aplicables en la esfera judicial específicamente en la Fiscalía General de la República (FGR).

La FGR es el Órgano del Estado al que corresponde como objetivos fundamentales, el control y la preservación de la legalidad sobre la base de la vigilancia del estricto cumplimiento de la Constitución, las leyes y demás disposiciones legales, por los organismos del Estado, entidades económicas, sociales y por los ciudadanos; así como la promoción y el ejercicio de la acción penal pública en representación del Estado (FGR, 2008). La FGR es una organización de prestigio nacional que desea agilizar el proceso de toma de decisiones, eficiencia en el trabajo de los fiscales y mayor conformidad por parte de los ciudadanos cubanos.

En la Fiscalía General de la República (FGR) actualmente se encuentra un software que permite acelerar los procesos que se llevan a cabo en la misma, pero esta aplicación informática no cumple con sus expectativas, ya que es una aplicación en Access que solo cuenta con un diccionario Jurídico y una leve gestión documental, de ahí la necesidad de contar con una aplicación que informatice los procesos fundamentales que tienen lugar en dicho organismo así como el apoyo en la toma de decisiones.

Unido a lo antes mencionado el déficit de Fiscales trae consigo que cada uno atienda más de un caso a la vez y que esté pendiente de los plazos de vencimiento establecidos, fusionado a la mayor complejidad de los procesos tramitados y al trabajo manual, provoca menor control y supervisión por parte del nivel superior de la tramitación de los procesos en los órganos provinciales.

Para la realización de un producto de software es necesario que los implementadores entiendan las necesidades de informatización, posibilitando la construcción de un sistema exitoso. Actualmente en el proyecto Sistemas de Gestión Fiscal (SGF) no se encuentran traducidas las necesidades del cliente a un lenguaje que sea comprendido por el equipo de desarrollo, lo que imposibilita la construcción del módulo Índices de Peligrosidad Pre-delictiva y por ende la culminación del Subsistema de Procesos Penales.

Luego de analizar la problemática anterior, se plantea la siguiente interrogante como **problema cientifico:** ¿Cómo transformar las necesidades del cliente en un lenguaje comprensible al equipo de desarrollo que facilite la implementación del Módulo Índices de Peligrosidad Pre-delictiva de la solución Informática Sistemas de Gestión Fiscal?

Este problema se enmarca en el **objeto de estudio:** La Ingeniería de Software. Teniendo como **campo de acción:** El Análisis y Diseño.

Para dar solución al problema antes expuesto se plantea el siguiente **Objetivo General:** Realizar el Análisis y el Diseño del módulo de Índices de Peligrosidad Pre-delictiva de la solución informática Sistema de Gestión Fiscal.

La **hipótesis** planteada es la siguiente**:** Si se realiza el Análisis y el Diseño de Software entonces se posibilitará la implementación del módulo IPP de la solución informática Sistema de Gestión Fiscal

Para dar cumplimiento a los objetivos del presente trabajo se desarrollaron las siguientes **Tareas de la investigación:**

Tareas de la investigación:

- Realización del Marco Teórico de la Investigación.
- Realización del estudio y caracterización del Proceso IPP.
- Realización de la Documentación de las necesidades de informatización de la organización.
- Realización del Diseño de Software.
- Aplicación de técnicas para la validación de la propuesta presentada.

Métodos de investigación utilizados: Teórico y Empírico.

Métodos Teóricos:

Análisis - Síntesis: Permite analizar el fenómeno a estudiar, y a partir de este análisis dividirlo permitiendo el procesamiento de la información arribando a conclusiones importantes para la investigación.

Histórico- Lógico: Permite el análisis de la evolución de los patrones de casos de uso, así como del diseño y las metodologías de desarrollo. Además del estado del arte realizado en la investigación.

Modelado: Posibilita la transcripción de los fenómenos de la realidad a un lenguaje entendible para el equipo de desarrollo, ejemplo: modelos de casos de uso del negocio y sistemas, diagramas de actividades entre otros.

Métodos Empíricos:

Entrevista: Este método posibilitó la obtención de la información necesaria para realizar el modelado de los procesos, es de vital importancia en los flujos de trabajo de Modelación del Negocio y Requerimientos.

Resultados Esperados:

- Modelado de negocio del Módulo Índices de Peligrosidad Pre-delictiva que posibilita la comprensión del ambiente en que se desarrollará el sistema.
- La Especificación de los Requisitos de Software que posibilitará la modelación del Sistema.
- El Modelo de Análisis del Módulo Índices de Peligrosidad Pre-delictiva.
- El Modelo de Diseño que permitirá la implementación exitosa del Módulo Índices de Peligrosidad Pre-delictiva.

Organización de la investigación:

Capítulo I “Fundamentación Teórica”: Este capítulo aborda el estudio realizado para la construcción de la investigación, como fueron el estado del arte sobre los sistemas que existen en el mundo para la gestión judicial y el proceso de IPP en Cuba. Se realizó un estudio sobre el proceso de desarrollo de software, entiéndase Ingeniería de requisitos, metodologías de desarrollo, herramientas CASE, lenguajes de modelado, lenguajes de programación, métricas, entre otros.

Capítulo II" Análisis del Módulo Índice de Peligrosidad Pre Delictiva": Este capítulo aborda el análisis del módulo IPP. Se trata el modelo de negocio, sistemas, requisitos de software, así como la especificación de los mismos y el uso de los patrones de caso de uso utilizados.

Capítulo III" Diseño y Validación del Módulo Índice de Peligrosidad Pre Delelictiva": Este capítulo aborda el diseño del módulo IPP, haciendo énfasis en los patrones de diseño utilizados. Además se trata la validación de la propuesta presentada mediante la aplicación de diferentes técnicas.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En el presente capítulo se caracterizará el Proceso de Índices de Peligrosidad en Cuba, así como un estudio detallado sobre los sistemas que existen en el mundo para la informatización de los procesos judiciales. Se realiza un estudio sobre el Proceso de desarrollo de software, Ingeniería de Software así como metodologías de desarrollo, herramientas CASE, lenguajes de modelado, lenguajes de programación y Frameworks de desarrollo como bases para la construcción de la presente investigación. Se aborda la Ingeniería de requisitos, patrones de casos de uso, patrones de diseño así como las técnicas de validación de software, como parte de la construcción y validación de propuesta de solución presentada.

1.2. Proceso Índices de Peligrosidad Pre-delictiva en Cuba

El proceso de Índices de Peligrosidad en Cuba más conocido en términos legales como Proceso de IPP ha evolucionado considerablemente desde el triunfo de la revolución hasta nuestros días. Por imperio de la legalidad y justicia la legislación revolucionaria cambió tanto el sentido sustantivo de las llamadas "medidas de seguridad "así como el proceder para su aplicación. (Pereira, 2005)

Para comprender el término IPP se hace indispensable conocer lo que se entiende por Estado Peligroso. En efecto el Estado Peligroso no es más que:

Artículo 72. Se considera estado peligroso la especial proclividad en que se halla una persona para cometer delitos (FGR, 2007).

El Estado Peligroso se aprecia cuando en el sujeto se muestra alguno de los índices de peligrosidad siguiente:

1. Embriaguez Habitual y Dipsomanía (Procedimiento de los enfermos)
2. Narcomanía (Procedimiento de los enfermos)
3. Conducta antisocial

Otros índices de peligrosidad a uno de los trámites anteriores:

1. Enajenados mentales
2. Desarrollo mental retardado (Si por esta causa no posee la facultad de comprender el alcance de sus acciones y dirigir su conducta.)

Existen diferentes medidas en el proceso de IPP para llevar a cabo la reinserción del presunto asegurado a la vida normal, entre las que podemos destacar:

Medidas de Seguridad Pre-delictiva:

1. Para la Embriaguez habitual, Dipsomanía, Narcomanía (narcóticos), Enajenados mentales y Desarrollo mental retardado son medidas terapéuticas. (Art 78 a y 79). En el caso de Dipsomanía, Narcomanía (narcóticos) también se le aplica vigilancia de los órganos de la PNR.
2. Para la conducta antisocial, las medidas son reeducativas y de vigilancia de los órganos de la PNR. (Art 78 b,c, 80. 81)

Medidas terapéuticas:

a) Internamiento en establecimiento asistencial, psiquiátrico o de desintoxicación.
b) Asignación a centro de enseñanza, con o sin internamiento.
c) Tratamiento médico externo. (Art 79.1)

Medidas reeducativas:

a) Internamiento en un establecimiento especializado de trabajo o de estudios.
b) Entrega a un colectivo de trabajo, para el control y la orientación de la conducta del sujeto en estado peligroso.

La ley prevé los actos que los órganos encargados de la realización de este procedimiento especial, deben efectuar sobre la base del reclamo social de la humanidad, garantías esenciales y sobre todo de legalidad. La FGR como órgano que imparte justicia en nuestro país es el encargado de llevar a cabo dicho proceso, pero para la facilitación del mismo se hace indispensable liberarse del trabajo manual así como contar con una aplicación informática capaz de acelerar este proceso y que sirva de apoyo en la toma de decisiones.

1.3. Sistemas para Procesos Jurídicos

A nivel mundial existen diferentes aplicaciones informáticas usadas para el control y la facilitación de los trámites judiciales. En busca de mejoras potenciales se realizó un estudio de estos sistemas entre los que podemos destacar:

1.3.1. Directum

Es una solución tecnológica del área de las TICs que permite manejar de manera fácil, rápida e intuitiva todos los procesos judiciales, coactivos, disciplinarios y administrativos de las entidades, grupos de abogados, departamentos, estudiantes y todas aquellas personas relacionadas con el área legal de diferente índole. (Directum, 2010)

1.3.2. Control de Procesos Judiciales

Permite tener control de citas pendientes por proceso, demandantes y demandados, estado del proceso, documentos pendientes y recibidos, honorarios por proceso, caja menor por cliente, anexos o datos adjuntos por proceso, reportes del estado actual para enviarle al demandante, informe de paz y salvo, informe de procesos por ciudad, por juzgado y otros, gráficos dinámicos para filtrar su información desde varios puntos de vista. (Software para Abogados, 2010)

1.3.3. Software para control de procesos judiciales (SGP)

Presenta un conjunto de herramientas que se necesitan para mantener al día la base de datos de procesos judiciales, como son: agenda de vencimiento de términos con alertas, estimativo de liquidación de las cuantías, listado de procesos por abogado, procesos sin movimiento, informes del estado actual de sus procesos estadísticas y búsquedas avanzadas. (Vigilancia Judicia, 2010)

1.3.4. ABOGest

Es una aplicación diseñada para controlar quiénes y de qué forma se relacionan con su bufete, qué asuntos tiene en trámite y los que han llegado a su término. Permite automatizar el trabajo diario controlando las citas pendientes o los vencimientos, e incluso a cuánto ascienden sus beneficios; y siempre de una forma sencilla y cómoda. (Abogest, 2010)

Del estudio realizado se arribó a la conclusión que de estos sistemas era imposible utilizar sus funcionalidades ya que los mismos son sistemas diseñados para Bufetes de abogados y los procesos que se informatizan no son compatibles con la Fiscalía General de la República. Además están dirigidos a países con condiciones judiciales totalmente diferentes a nuestro país socialista, lo que causa una total discrepancia con la constitución de la república así como leyes y demás disposiciones legales. Lo que evidenció la necesidad de construir un software que satisfaga las necesidades de la FGR.

(Ver anexos 1 para otros Sistemas Judiciales)

1.4. Proceso de Desarrollo de Software

Un proceso de desarrollo de software tiene como propósito la producción eficaz y eficiente de un producto software que reúna los requisitos del cliente. (Ivar Jacobson, 2000). Este proceso es intensamente intelectual, afectado por la creatividad y juicio de las personas involucradas (Sommerville, 2002). Aunque un proyecto de desarrollo de software es equiparable en muchos aspectos a cualquier otro proyecto de ingeniería, en el desarrollo de software hay una serie de desafíos adicionales, relativos esencialmente a la naturaleza del producto obtenido.

Un producto software es intangible y por lo general muy abstracto, esto dificulta la definición del producto y sus requisitos, sobre todo cuando no se tiene precedentes en productos software similares (Ivar Jacobson, 2000) esto hace que los requisitos sean difíciles de consolidar tempranamente. Así, los cambios en los requisitos son inevitables, no solo después de entregado el producto sino también durante el proceso de desarrollo por lo que se hace indispensable la elección correcta de las herramientas, técnicas y métodos para llevar a cabo el proceso.

En el proceso de desarrollo de software intervienen diversas situaciones que pueden afectar la construcción y puesta en funcionamiento del producto. Se hace de suma importancia contar con los métodos y técnicas correctos que satisfagan las necesidades del proceso.

1.5. Ingeniería de Software

La Ingeniería del Software es una disciplina que pertenece a la rama de la informática, que ofrece métodos y técnicas para desarrollar y mantener productos informáticos de buena calidad que ayudan a resolver problemas de todo tipo. En la actualidad es cada vez más frecuente la consideración de la Ingeniería del Software como una nueva área de la ingeniería.

Existen diferentes autores que se refieren a la Ingeniería de Software de la siguiente manera:

La Ingeniería de Software trata del establecimiento de los principios y métodos de la ingeniería a fin de obtener software de modo rentable, que sea fiable y trabaje en máquinas reales (Bauer, 1972).

Ingeniería de software es la aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos. Se conoce también como Desarrollo de Software o Producción de Software (Bohem, 1976).

Es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software; es decir, la aplicación de la ingeniería al software (IEEE, 1993).

Como parte de la Ingeniería de Software la elección de una metodología de desarrollo es un paso muy importante pues esta es la base de la construcción del sistema y posibilita la obtención de un producto software que cumpla con las especificaciones del cliente.

1.6. Metodologías de Desarrollo

En el Proceso de Desarrollo de Software se utilizan metodologías que son de vital importancia durante el proceso de construcción por que imponen un proceso disciplinado sobre su desarrollo con el propósito de hacerlo más predecible y eficiente.

Menéndez se refiere a las metodologías de desarrollo de software como un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software (Menéndez, 2005).

El éxito del producto depende en gran parte de la metodología escogida por el equipo, en la actualidad existen dos tipos de metodologías Las metodologías tradicionales o pesadas y las metodologías ágiles o livianas, la metodología escogida ya sea tradicional o ágil, debe ser capaz de maximizar el potencial del equipo de desarrollo, además de aumentar la calidad del producto con los recursos y tiempos establecidos.

1.6.1. Microsoft Solution Framework (MSF)

MSF es un compendio de las mejores prácticas en cuanto a administración de proyectos se refiere. Más que una metodología rígida de administración de proyectos, es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos.

MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

MSF tiene las siguientes características:

- **Adaptable:** es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- **Escalable:** puede organizar equipos tan pequeños entre tres o cuatro personas, así como también, proyectos que requieren 50 personas a más.

- **Flexible:** es utilizada en el ambiente de desarrollo de cualquier cliente.
- **Tecnología Agnóstica:** porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología. (MSF, 2010)

1.6.2. Rational Unified Process

RUP es un proceso formal: Provee un acercamiento disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad que satisfaga los requerimientos de los usuarios finales (respetando cronograma y presupuesto). Fue desarrollado por Rational Software, y puede ser adaptado y extendido para satisfacer las necesidades de la organización que lo adopte. (Roberth G. Figueroa)

Ciclo de Vida

En lo que se refiere al ciclo de vida RUP, es una implementación del desarrollo en espiral. El ciclo de vida organiza las tareas en fases e iteraciones. RUP divide el proceso de desarrollo en ciclos, teniendo un producto final al culminar cada una de ellos, estos a la vez se dividen en fases donde se debe tomar una decisión importante:

- **Inicio**: se hace un plan de fases, se identifican los principales casos de uso y se identifican los riesgos
- **Elaboración**: se hace un plan de proyecto, se completan los casos de uso y se eliminan los riesgos
- **Construcción**: se concentra en la elaboración de un producto totalmente operativo y eficiente y el manual de usuario
- **Transición**: se instala el producto en el cliente y se entrena a los usuarios. Surgen nuevos requisitos a ser analizados
- **Mantenimiento**: una vez instalado el producto, el usuario realiza requerimientos de ajuste, esto se hace de acuerdo a solicitudes generadas como consecuencia del interactuar con el producto (Scribd, 2010)

RUP presenta dos dimensiones:

- El eje horizontal representa el tiempo y demuestra los aspectos del ciclo de vida del proceso.

- El eje vertical representa las disciplinas, que agrupan actividades definidas lógicamente por la naturaleza

La primera dimensión representa el aspecto dinámico del proceso y se expresa en términos de fases, de iteraciones, y la finalización de las fases. La segunda dimensión representa el aspecto estático del proceso: cómo se describe en términos de componentes de proceso, las disciplinas, las actividades, los flujos de trabajo, los artefactos, y los roles.

En la imagen que se muestra a continuación se encuentra los Flujos de trabajo que tiene definido RUP, los 6 primeros llamados ingenieriles y los 3 últimos de apoyo

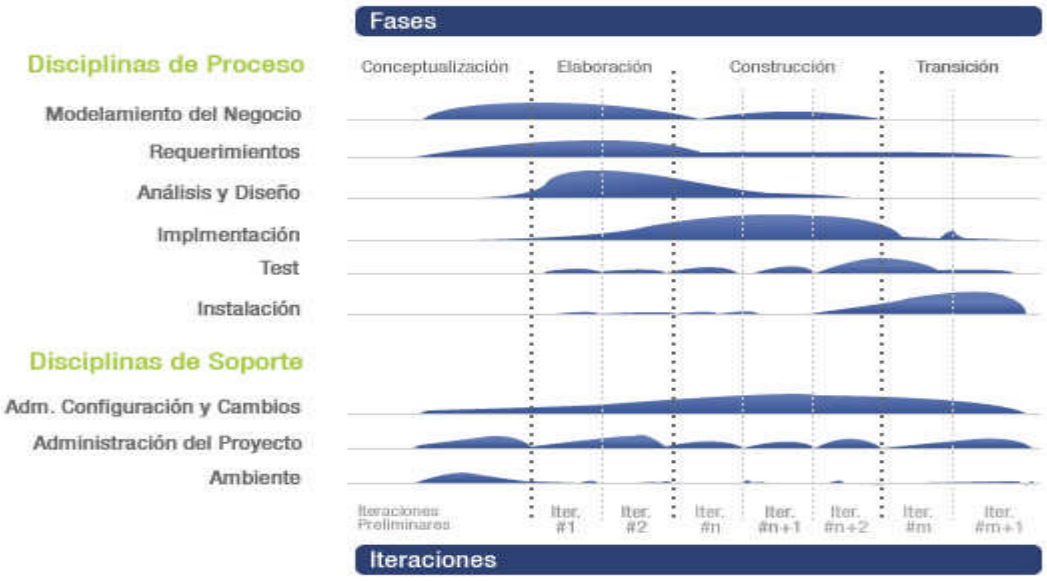


Figura 1. Fases y flujos de Trabajo de RUP.

Características de la Metodología RUP

- Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo)
- Pretende implementar las mejores prácticas en Ingeniería de Software
- Desarrollo iterativo
- Administración de requisitos
- Uso de arquitectura basada en componentes

- Control de cambios
- Modelado visual del software
- Verificación de la calidad del software

RUP además se caracteriza por ser un proceso dirigido o guiado por los casos de usos, iterativo e incremental, y estar centrado en la arquitectura.

- **Proceso Dirigido por los Casos de Uso**: Se refiere a la utilización de los casos de uso para el desenvolvimiento y desarrollo de las disciplinas con los artefactos, roles y actividades necesarias. Los casos de uso son la base para la implementación de las fases y disciplinas de RUP.
- **Proceso Iterativo e Incremental:** Es el modelo utilizado por RUP para el desarrollo de un proyecto de software. Este modelo plantea la implementación del proyecto a realizar en iteraciones, con lo cual se pueden definir objetivos por cumplir en cada iteración y así poder ir completando todo el proyecto iteración por iteración, con lo cual se tienen varias ventajas, entre ellas se puede mencionar la de tener pequeños avances del proyectos que son entregables al cliente, el cual puede probar mientras se está desarrollando otra iteración del proyecto, con lo cual el proyecto va creciendo hasta completarlo en su totalidad.
- **Proceso Centrado en la Arquitectura**: Define la arquitectura de un sistema, y una arquitectura ejecutable construida como un prototipo evolutivo. Arquitectura de un sistema es la organización o estructura de sus partes más relevantes. Una arquitectura ejecutable es una implementación parcial del sistema, construida para demostrar algunas funciones y propiedades. RUP establece refinamientos sucesivos de una arquitectura ejecutable, construida como un prototipo evolutivo. (RUP (2), 2010)

  RUP está basado en 5 principios:

  - **Adaptar el proceso:** El proceso deberá adaptarse a las características propias del proyecto u organización. El tamaño del mismo, así como su tipo o las regulaciones que lo condicionen, influirán en su diseño específico, aunque se debe tener en cuenta el alcance del proyecto.
  - **Balancear prioridades:** Debe encontrarse un balance que satisfaga los deseos de todos.
  - **Demostrar valor iterativamente:** Los proyectos se entregan en etapas iteradas. En cada iteración se analiza la opinión, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados

- **Elevar el nivel de abstracción:** Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes de cuarta generación (SQL, lenguajes de consulta), o esquemas (framework). Esto previene a los ingenieros de software ir directamente de los requisitos a la codificación de software a la medida del cliente. Un nivel alto de abstracción también permite discusiones sobre diversos niveles arquitectónicos. Éstos se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con UML.
- **Enfocarse en la calidad:** El control de calidad debe realizarse en todos los aspectos de la producción. El aseguramiento de la calidad forma parte del proceso de desarrollo y no de un grupo independiente. (RUP (1), 2010)

1.6.3. Programación Extrema (XP)

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en la realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Proceso XP

El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos:

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.

El ciclo de vida ideal de XP consiste en seis fases: Exploración, Planificación de la Entrega (Release), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto. (José H. Canós)

1.6.4. Scrum

Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Define un marco para la gestión de proyectos. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo del proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

En Scrum se realizan entregas parciales y regulares del resultado final del proyecto, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad y la productividad son fundamentales. (José H. Canós)

1.6.5. Justificación de la Metodología Seleccionada

Después de realizar el estudio de las principales metodologías de desarrollo de software por parte del equipo de desarrollo se tomó RUP como proceso rector por adaptarse a las características y complejidad de este proyecto de software. Esta metodología hace énfasis en realizar una buena captura de los requisitos y es apropiado para proyectos de larga duración por la generación de documentación, cuestiones estas que no son abordadas de esta manera por las demás metodologías expuestas anteriormente. Posee evaluación en cada iteración que permite cambios de objetivos. Es sencillo, ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el software además del seguimiento detallado en cada una de las iteraciones. Otra de las razones de peso en la elección fue sin lugar a dudas el conocimiento y familiarización del equipo de desarrollo en esta metodología.

Una vez seleccionada la metodología de desarrollo como base para la construcción de un producto software, es imprescindible la selección de un lenguaje de modelado capaz de traducir la información captada por el equipo de desarrollo, a un lenguaje comprensible para el resto del personal del proyecto.

1.7. Lenguajes de Modelado

El lenguaje de modelado de objetos es un conjunto estandarizado de símbolos y de modos de disponerlos para modelar (parte de) un diseño de software orientado a objetos. (Martin, 1995). El lenguaje de modelado en combinación con una metodología de desarrollo de software es usado para avanzar de una especificación inicial a un plan de implementación y para comunicar dicho plan a todo un equipo de desarrolladores en otras palabras son una técnica que permite traducir especificaciones a modelos lo que posibilita el entendimiento por parte del equipo de desarrollo.

1.7.1. Lenguaje Unificado de Modelación (Unified Modeling Language, UML)

UML es un lenguaje estándar para escribir planos de software, es "un lenguaje gráfico para visualizar, especificar, construir, documentar y comunicar los artefactos de un sistema de software. (Jacobson y otros, 2000)"

Su alfabeto está constituido por elementos y relaciones, los cuales al combinarse forman diferentes tipos de diagramas. Figura 7

UML tiene como funciones principales:

- **Visualizar:** Permite expresar de una forma gráfica un sistema de forma que otro lo pueda entender.
- **Especificar:** Permite especificar cuáles son las características de un sistema antes de su construcción.
- **Construir:** A partir de los modelos especificados se pueden construir los sistemas diseñados.
- **Documentar:** Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

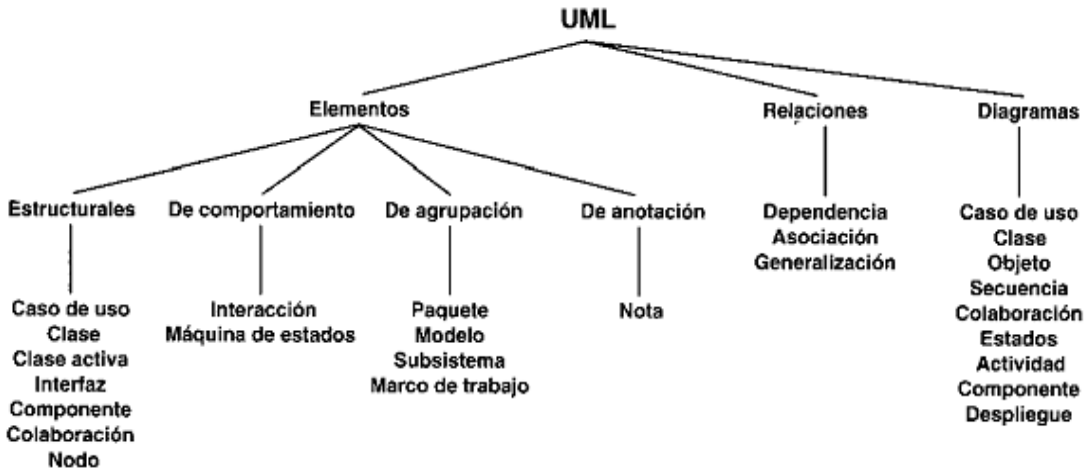


Figura 2. Vocabulario de UML.

1.7.2. Notación de Modelado de Procesos de Negocio (BPMN)

Un Proceso de Negocio es una colección de actividades que, tomando una o varias clases de entradas, crean una salida que tiene valor para un cliente. (Hammer y Champy, 1993).

La Notación de Modelado de Procesos de Negocio (Business Process Modeling Notation, BPMN) define un Diagrama de Procesos de Negocio (BPD), basado en la técnica de "flowcharting" (diagramado de flujos) que ajusta modelos gráficos de operación de procesos de negocio, además es una red de objetos gráficos, correspondientes a actividades y controles de flujo que definen el orden de ejecución de éstas.

Un BPD está estructurado por un grupo de elementos gráficos.

Las cuatro categorías básicas de elementos son:

- Objetos de flujo
- Objetos de conexión
- Calles
- Artefactos

BPMN es capaz de representar semánticas de procesos complejos tiene como objetivo principal servir como soporte para la gestión por procesos, como una notación que pueda ser entendida fácilmente desde los analistas que crean los bocetos iniciales del proceso, los desarrolladores técnicos responsables de implementar la tecnología que ejecutará estos procesos hasta las personas que los ejecutan y aquellas

que llevarán a cabo el monitoreo y supervisión de los procesos, otro objetivo es asegurar que los lenguajes XML diseñados para la ejecución de procesos de negocio puedan ser visualizados con una notación común, dentro de estos lenguajes encontramos por ejemplo BPEL4WS, que es un lenguaje de ejecución de procesos de negocios para servicios web. (White, 2003)

1.7.3. Métodos Integrados de Definición (IDEF)

IDEF0 constituye una técnica de modelación gráfica, especializada en la representación de las relaciones e interdependencias existentes entre los diferentes procesos. (Winnik, 2008)

Su principal característica consiste en su capacidad para diferenciar entre tres tipos posibles de relación entre procesos:

a) relaciones que establecen las guías que debe tener en cuenta el proceso.
b) relaciones que aportan los recursos necesarios para llevar a cabo el proceso.
c) relaciones de encadenamiento lineal entre procesos (entrada – salida). (Lenguajes de Modelado, 2010)

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. La semántica de utilización de estos elementos gráficos es la siguiente:

Actividad: se representa con un cuadro, indica una función, proceso o transformación.
Entrada: se representa con una flecha entrando por el lado izquierdo de la actividad, indica los materiales o informaciones que se transformarán en la actividad para obtener la salida.
Salida: se representa con una flecha saliendo del lado derecho de la actividad, indica los objetos o informaciones producidos por la ocurrencia de la actividad.
Control: se representa con una flecha entrando por la parte superior, indica las regulaciones que determinan si una actividad se realiza o no. Ejemplo: normas, guías, reglas, políticas, entre otros.
Mecanismo: se representa con una flecha entrando por la parte inferior, indica los recursos que ejecutan una actividad. Ejemplo: personas, maquinarias, entre otros. (Lenguajes de Modelado, 2010)

A continuación se muestra la figura que representa los elementos gráficos utilizados para la construcción de los diagramas

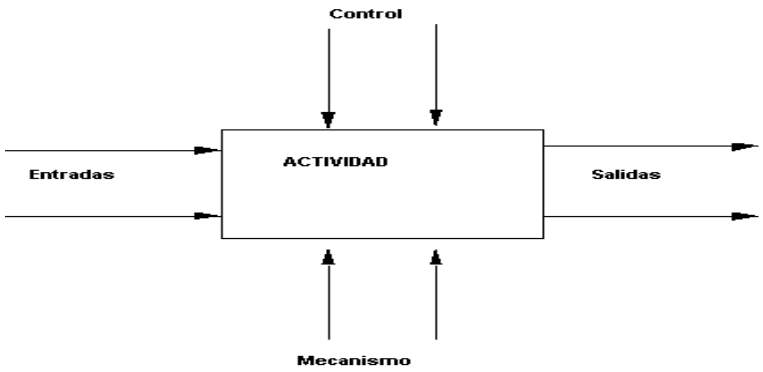


Figura 3. Diagrama de IDEF0.

1.7.4. Justificación de la selección del Lenguaje de Modelado

Después de realizar un estudio de los lenguajes de modelado se adopta UML para la realización de este trabajo. Este es el más utilizado a nivel mundial para modelar los artefactos creados durante el proceso de desarrollo de software, generando una gran cantidad de documentación a lo largo de todos los flujos de trabajo, lo que será de particular importancia en un proyecto de gran envergadura. Teniendo en cuenta además que fue desarrollado junto con la metodología RUP, por lo que responde a todas sus necesidades, se combinan como la elección correcta del equipo de desarrollo.

El lenguaje de modelado proporciona el conjunto de símbolos y modos de disponerlo pero esto no es suficiente, es necesario contar con una herramienta capaz de digitalizar estos símbolos y trasladarlos a imágenes y documentos, imprescindibles en el proceso de desarrollo de software.

1.8. Herramientas CASE

De acuerdo con Kendall y Kendall la Ingeniería de Sistemas Asistida por Ordenador (CASE) es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo, su objetivo es automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas. (Kendall, 1997)

Las herramientas CASE son un conjunto de programas informáticos que ayudan a mejorar la productividad en el desarrollo de software, refiriéndose a productividad como la eficiencia en el desarrollo,

así como la efectividad del sistema implementado. Reducen el coste, tanto en tiempo como en dinero de la aplicación, mejoran la gestión la calidad y dominio sobre el proyecto en cuanto a su planificación, ejecución y control, automatiza la documentación, la generación de código, la detección de errores, y principalmente son una herramienta fundamental para los analistas, diseñadores y desarrolladores de software en el ciclo de vida del proceso de desarrollo de software

1.8.1. Visual Paradigm

Visual Paradigm es una herramienta de diseño que soporta todos los diagramas UML, diagramas de SysML (Lenguajes de Modelado de Sistemas) y el diagrama entidad-relación. Visual Paradigm ofrece amplias características de modelado de casos de uso incluyendo la función completa de UML Diagrama de casos de uso, flujo de eventos editor, de casos de uso / red de actor y la generación de un diagrama de actividad, esta herramienta CASE produce la documentación del sistema en formato PDF, HTML y MS Word. (Visual Paradigm (1), 2010)

Características de esta herramienta:

- Entorno de creación de diagramas para UML 2.0
- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación
- Capacidades de ingeniería directa e inversa
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo Disponibilidad de múltiples versiones, para cada necesidad
- Disponibilidad de integrarse en los principales IDEs (Entornos de Desarrollo Integrados)
- Disponibilidad en múltiples plataformas(Windows, Linux)
- Ofrece un mecanismo general para la organización de los modelos/subsistemas/capas agrupando elementos de modelado y versión gratuita (licencia para Community Edition

1.8.2. Rational Rose

Es una de las herramientas más poderosas de modelado visual para el análisis y diseño de sistemas basados en objetos. Se utiliza para modelar un sistema antes de proceder a construirlo, cubre el ciclo de vida completo de un proyecto (concepción y formalización del modelo, construcción de los componentes, transición a los usuarios, y certificación de las distintas fases)

Características de la Herramienta CASE:

- Soporte para análisis de patrones ANSI C++, Rose J y Visual C++ basado en "Diseño de Patrones:
- Característica de control por separado de componentes modelo que permite una administración más granular y el uso de modelos
- Soporte de ingeniería Forward y/o reversa para algunos de los conceptos más comunes de Java 1.5
- La generación de código Ada, ANSI C ++, C++, CORBA, Java y Visual Basic, con capacidad de sincronización modelo- código configurables
- Soporte Enterprise Java Beans™ 2.0
- Capacidad de análisis de calidad de código
- El Add-In para modelado Web provee visualización, modelado y las herramientas para desarrollar aplicaciones de Web
- Modelado UML para trabajar en diseños de base de datos, con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos
- Capacidad de crear definiciones de tipo de documento XML (DTD) para el uso en la aplicación
- Integración con otras herramientas de desarrollo de Rational
- Capacidad para integrarse con cualquier sistema de control de versiones SCC-compliant, incluyendo a Rational ClearCase
- Publicación web y generación de informes para optimizar la comunicación dentro del equipo. (Rational Rose, 2010)

1.8.3. Enterprise Architect (EA)

Enterprise Architect es una herramienta comprensible de diseño y análisis UML, cubriendo el desarrollo de software desde el paso de los requerimientos a través de las etapas del análisis, modelos de diseño, pruebas y mantenimiento. EA es una herramienta multi-usuario, basada en Windows, diseñada para ayudar a construir software robusto y fácil de mantener. Ofrece salida de documentación flexible y de alta calidad. EA es una herramienta CASE de software propietario

Las bases de Enterprise Architect están construidas sobre la especificación de UML 2.0, usa perfiles UML para extender el dominio de modelado, mientras que la validación del modelo asegura integridad.

Enterprise Architect presenta:

- Soporte para los 13 diagramas de UML y más.
- Generación e ingeniería inversa de código fuente para muchos lenguajes populares, entre los que podemos ver: C++, C#, Java, Delphi, VB.Net, Visual Basic y PHP.
- Add-ins gratis para CORBA y Python disponibles.
- Editor de código fuente con "resaltador de sintaxis" incorporado. (Enterprice Architect, 2010)

1.8.4. Justificación de la selección de la Herramienta CASE

Después de realizar el estudio de las principales herramientas CASE existentes se decidió utilizar Visual Paradigm. Se lleva a cabo esta elección principalmente por ser una herramienta multiplataforma, facilidad que pocas herramientas CASE brindan, pero además es robusta, de fácil uso y que facilita la posibilidad de exportar documentos. También se sustenta la elección de la misma en el hecho de que la universidad cuenta con la licencia para el uso de dicha herramienta.

La herramienta CASE seleccionada proporciona la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo posibilitando contener toda la información en diagramas y documentos lo que facilita la implementación del producto diseñado.

1.9. Lenguajes de Programación

Actualmente existen diferentes lenguajes de programación para desarrollar en la web, los cuales han ido surgiendo debido a las tendencias y necesidades de las plataformas, estos lenguajes pueden encontrarse del lado del servidor, lenguajes que son reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato comprensible para él y por otro lado los del lado del cliente que son aquellos que pueden ser directamente digeridos por el navegador y no necesitan un pre-tratamiento.

En el dominio de la red, los lenguajes de lado servidor más ampliamente utilizados para el desarrollo de páginas dinámicas son el ASP, Java, PHP y PERL.

1.9.1. Perl

Perl es un lenguaje de programación interpretado, al igual que muchos otros lenguajes de Internet como Javascript o ASP. Esto quiere decir que el código de los scripts en Perl no se compila sino que cada vez que se quiere ejecutar se lee el código y se pone en marcha interpretando lo que hay escrito. Además es

extensible a partir de otros lenguajes, ya que desde Perl podremos hacer llamadas a subprogramas escritos en otros lenguajes. También desde otros lenguajes podremos ejecutar código Perl.

El lenguaje Perl no es pre compilado, pero aun así es más rápido que la mayoría de lenguajes interpretados. Esto se debe a que los programas en Perl son analizados, interpretados y compilados por el intérprete Perl antes de su ejecución.

Estas características hacen que el mantenimiento y la depuración de un programa en Perl sean mucho más sencillos que el mismo programa escrito en C.

Por todo esto, Perl es un lenguaje muy utilizado en los dos campos siguientes:

1. **La administración de sistemas operativos.** Debido a sus características Perl es muy potente en la creación de pequeños programas que pueden ser usados como filtros para obtener información de ficheros, realizar búsquedas, entre otros. Además, aunque Perl nació en un entorno Unix, hay versiones para casi todas las plataformas existentes.
2. **La creación de formularios en la Web.** Es decir, se utilizan para la creación de scripts CGI (Common Gateway Interface (Interfaz de entrada común)). Estos scripts realizan el intercambio de información entre aplicaciones externas y servicios de información, es decir, se encargan de tratar y hacer llegar la información que el cliente WWW (World Wide Web) manda al servidor a través de un formulario. (Perl, 2010)

1.9.2. ASP (Active Server Pages)

Es la tecnología desarrollada por Microsoft para la creación de páginas dinámicas del servidor. Las siglas ASP corresponden a las palabras Active Server Pages (Páginas Activas en el Servidor) donde se escribe en la misma página web, utilizando el lenguaje Visual Basic Script o Jscript (Javascript de Microsoft).

ASP es un entorno de secuencias de comandos del servidor que puede utilizar para crear páginas Web dinámicas o para generar eficaces aplicaciones Web. Las páginas ASP son archivos que contienen etiquetas HTML, texto y comandos de secuencias de comandos. Las páginas ASP pueden llamar a componentes ActiveX para que realicen tareas, como la conexión con bases de datos o cálculos comerciales.

ASP nos permite tomar ventaja del Server-Side scripting (scripting que se ejecuta en el servidor). ASP nos provee de una gran cantidad de componentes y objetos con los cuales se puede manejar de manera

sencilla la interacción del navegador y el servidor Web. Lenguajes de scripts como JavaScript o VBScript se usan para manipular estos objetos. ASP no es un lenguaje en sí. Esto quiere decir que no hay código ASP, por eso se puede usar el lenguaje de scripting que le sea más cómodo. (ASP, 2010)

1.9.3. Java

Java es un lenguaje orientado a objetos, eso implica que su concepción es muy próxima a la forma de pensar humana.

Características del lenguaje

- Es un lenguaje que es compilado, generando ficheros de clases compilados, pero estas clases compiladas, son en realidad interpretadas por la máquina virtual de java. Siendo la máquina virtual de java la que mantiene el control sobre las clases que se estén ejecutando.
- Es un lenguaje multiplataforma: El mismo código java que funciona en un sistema operativo, funcionará en cualquier otro sistema operativo que tenga instalada la máquina virtual java.
- Es un lenguaje seguro: La máquina virtual, al ejecutar el código java, realiza comprobaciones de seguridad, además el propio lenguaje carece de características inseguras, como por ejemplo los punteros.
- Gracias al API (Interfaz de Programación de Aplicaciones) de java podemos ampliar el lenguaje para que sea capaz de, comunicarse con equipos mediante red, acceder a bases de datos, crear páginas HTML dinámicas y crear aplicaciones visuales al estilo Windows. (JAVA, 2010)

1.9.4. PHP

PHP es un lenguaje de programación usado frecuentemente para la creación de sitios web dinámicos. PHP es un acrónimo recursivo que significa "Hypertext Pre-processor" (inicialmente PHP Tools, o, Personal Home Page Tools), y se trata de un lenguaje interpretado usado para la creación de aplicaciones para servidores, o creación de contenido dinámico para sitios web. (PHP (2))

PHP usa una mezcla entre interpretación y compilación para intentar ofrecer a los programadores la mejor mezcla entre rendimiento y flexibilidad. (PHP)

Características del Lenguaje

- Soporte para una gran cantidad de bases de datos: Adabas D, dbm, dBase, filePro, Hyperwave, Informix, InterBase, LDAP, Microsoft SQL server, mSQL, MySQL, ODBC, Oracle, PostgreSQL, Solid, Sybase.
- Integración con varias bibliotecas externas, permite generar documentos en PDF (documentos de Acrobat Reader) hasta analizar código XML.
- Ofrece una solución simple y universal para las paginaciones dinámicas del Web de fácil programación.
- Perceptiblemente más fácil de mantener y poner al día que el código desarrollado en otros lenguajes.
- Soportado por una gran comunidad de desarrolladores, como producto de código abierto, PHP goza de la ayuda de un gran grupo de programadores, permitiendo que los fallos de funcionamiento se encuentren y reparen rápidamente.
- El código se pone al día continuamente con mejoras y extensiones de lenguaje para ampliar las capacidades de PHP.
- Con PHP se puede hacer cualquier cosa que podemos realizar con un script CGI, como el procesamiento de información en formularios, foros de discusión, manipulación de cookies y páginas dinámicas. (PHP (1) , 2010)

1.9.5. Justificación de la selección del Lenguaje de Programación

Por decisión de los líderes de la solución informática Sistemas de Gestión Fiscal (SGF) se adoptó el lenguaje PHP como lenguaje de programación a utilizar en el proceso de construcción de la aplicación informática en cuestión al ser un lenguaje libre por lo que se presenta como una alternativa de fácil acceso para todos, tiene soporte para la mayoría de las Base de datos además de ser un lenguaje multiplaforma que cuenta con la ayuda de una amplia comunidad de programadores.

En conjunto con el lenguaje de programación los frameworks ayudan en el desarrollo de software, proporcionando una estructura definida la cual ayuda a crear aplicaciones con mayor rapidez. Ayuda a la hora de realizar el mantenimiento del sitio gracias a la organización durante el desarrollo de la aplicación. Los frameworks son desarrollados con el objetivo de brindarles a los programadores y diseñadores una mejor organización y estructura a sus proyectos.

1.10. Framework de Desarrollo

El framework es una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta. (Framework (1), 2010)

Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

1.10.1. CodeIgniter

Poderoso framework para PHP que facilita la escritura de código repetitivo, y a comparación de otros framework cómo CakePHP, Symfony o Zend Framework, CodeIgniter es más rápido pero menos fácil ya que carece de algunas librerías que los otros framework tienen, pero aun así no deja de ser un buen framework además de que es totalmente extensible y altamente compatible con gran variedad de versiones y configuraciones de PHP. (CodeIgniter, 2010)

1.10.2. Zend

Es creado por los propios creadores de PHP, su filosofía es clara, la ley del mínimo esfuerzo, este framework está formado por una serie de métodos estáticos y componentes. Los componentes son varios y variados. Entre los componentes qué destacaría se encuentran: Zend_Config para temas de configuración de aplicaciones web, Zend_Db para tratar con bases de datos, Zend_Search o Zend_Feed entre otros. La instalación es sencilla, tan solo tendremos que añadir en el fichero de configuración php.ini, el path hasta la carpeta library del framework con la instrucción include_path. (Zend Framework, 2010)

1.10.3. CakePHP

El framework CakePHP proporciona una base robusta para tu aplicación. Puede manejar cualquier aspecto, desde la solicitud inicial del usuario hasta el renderizado final de la página web. Además, como el framework sigue los principios MVC (Modelo, Vista, Controlador), puedes fácilmente personalizar y extender muchos aspectos de la aplicación.

EL framework también proporciona una estructura de organización básica, desde los nombres de los archivos hasta los de las tablas de la base de datos, manteniendo toda tu aplicación consistente y lógica.

Este aspecto es simple pero poderoso. Sigue las convenciones y siempre sabrás exactamente dónde están las cosas y cómo están organizadas. (CakePHP, 2010)

1.10.4. Symfony

Symfony es un framework para construir aplicaciones web con PHP. En otras palabras, Symfony es un enorme conjunto de herramientas y utilidades que simplifican el desarrollo de las aplicaciones web.

Symfony ha tomado las mejores ideas del framework Rails y de muchos otros más, ha incorporado ideas propias y el resultado es un framework elegante, estable, productivo y muy bien documentado. (Symfony (1), 2010)

Symfony emplea el tradicional patrón de diseño MVC para separar las distintas partes que forman una aplicación web. El modelo representa la información con la que trabaja la aplicación y se encarga de acceder a los datos. La vista transforma la información obtenida por el modelo en las páginas web a las que acceden los usuarios. El controlador es el encargado de coordinar todos los demás elementos y transformar las peticiones del usuario en operaciones sobre el modelo y la vista.

Características

- Fácil de instalar y configurar en sistemas Windows, Mac y Linux.
- Funciona con todas las bases de datos comunes (MySQL, PostgreSQL, SQLite, Oracle, MS SQL Server).
- Compatible solamente con PHP 5 desde hace años, para asegurar el mayor rendimiento y acceso a las características más avanzadas de PHP.
- Basado en la premisa de "convenir en vez de configurar", en la que el desarrollador solo debe configurar aquello que no es convencional.
- Preparado para aplicaciones empresariales, ya que se puede adaptar con facilidad a las políticas y arquitecturas propias de cada empresa u organización.
- Flexible hasta cualquier límite y extensible mediante un completo mecanismo de plugins.
- Publicado bajo licencia MIT (Instituto Tecnológico de Massachusetts) de software libre y apoyado por una empresa comprometida con su desarrollo.
- Traducido a más de 40 idiomas y fácilmente traducible a cualquier otro idioma. (Symfony, 2010)

1.10.5. Justificación de la selección del Framework de Desarrollo

El framework seleccionado fue Symfony al ser diseñado con el objetivo de optimizar la creación de las aplicaciones web, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. Posee una librería de clases que permiten reducir el tiempo de desarrollo. Está desarrollado en PHP5, es multiplataforma y libre.

Una vez seleccionados todos los métodos y herramientas se traduce las necesidades del cliente en modelos y diagramas. Para la correcta realización de este complejo proceso de transcripción es de vital importancia llevar a cabo una buena práctica de la ingeniería de requisitos.

1.11. Ingeniería de Requisitos

La Ingeniería de Requisitos del software es un proceso de descubrimiento, refinamiento, modelado y especificación. Se refinan en detalle los requisitos del sistema y el papel asignado al software inicialmente asignado por el ingeniero del sistema. Se crean modelos de los requisitos de datos, flujo de información y control, y del comportamiento operativo. Se analizan soluciones alternativas y el modelo completo del análisis es creado.

Algunos autores definen la Ingeniería de Requisitos como:

La ingeniería de Requerimientos ayuda a los ingenieros de software a entender mejor el problema en cuya solución trabajarán. Incluye el conjunto de tareas que conducen a comprender cuál será el impacto del software sobre el negocio, qué es lo que el cliente quiere y cómo interactuarán los usuarios finales con el software. (Pressman, 2006)

La parte más difícil de construir un sistema es precisamente saber qué construir. Ninguna otra parte del trabajo conceptual es tan difícil como establecer los requerimientos técnicos detallados, incluyendo todas las interfaces con personas, máquinas y otros sistemas. Ninguna otra parte del trabajo afecta tanto el sistema si es hecha mal. Ninguna es tan difícil de corregir más adelante… Entonces, la tarea más importante que el ingeniero de software hace para el cliente es la extracción iterativa y el refinamiento de los requerimientos del producto. (Frederick P. Brooks, 1987).

Como parte del proceso de desarrollar una especificación de software surgen necesidades de informatización por parte del cliente. Estas necesidades son conocidas en el mundo del software como Requisitos o Requerimientos de Software.

1.11.1. Requisitos de Software

En la ingeniería de sistemas, un requerimiento es una necesidad documentada sobre el contenido, forma o funcionalidad de un producto o servicio. Se usa en un sentido formal en la ingeniería de sistemas o la ingeniería de software (IEEE). Según el criterio de varios autores un requerimiento es:

- Condición o capacidad que un usuario necesita para poder resolver un problema o lograr un objetivo (IEEE).
- Condición o capacidad que debe exhibir o poseer un sistema para satisfacer un contrato, estándar, especificación, u otra documentación formalmente impuesta (IEEE).
- Una condición o capacidad que debe ser conformada por el sistema (RUP).
- Algo que el sistema debe hacer o una cualidad que el sistema debe poseer (Robertson, Robertson).

El propósito fundamental de la captura de los requisitos es guiar el desarrollo hacia el sistema correcto. Esto se consigue mediante una descripción de los requisitos del sistema suficientemente buena como para que pueda llegarse a un acuerdo entre el cliente (incluyendo a los usuarios) y los desarrolladores sobre qué debe y qué no debe hacer el sistema. (Rumbaugh, y otros, 2004)

Clasificación de Requisitos

Los requisitos de software los podemos clasificar de dos formas:

Requisitos Funcionales (RF): Los requisitos funcionales describen las funciones que el software va a ejecutar; por ejemplo, ajustarse a un formato de texto o modular una señal. Se conocen también como capacidades o condiciones que el sistema debe hacer.

Requisitos no funcionales (RNF): son requisitos de software que describen no lo que el software hará, sino como lo hará. Estos son difíciles de verificar/testear, y por ello son evaluados subjetivamente (Thayer, 1990).

Entre estos requisitos no funcionales se encuentran:

- **Requerimientos de Software:** se refiere al software que se debe poseer, por ejemplo el sistema operativo que se debe utilizar

- **Requerimientos de Hardware:** hacen referencia a los elementos de hardware que se deben tener para el correcto funcionamiento de un sistema
- **Restricciones en el diseño y la implementación:** son restricciones que no pueden dejar de cumplirse, son imprescindibles para la construcción del sistema en cuestión
- **Requerimientos de apariencia o interfaz externa:** hacen referencia a restricciones en la apariencia e interfaces externas que debe mostrar el producto.
- **Requerimientos de Seguridad:** hacen referencia al manejo de la seguridad. Pueden conducir a riesgos grandes, si no se gestionan correctamente. Abarcan tres aspectos diferentes:
  - **Confidencialidad:** que la información esté protegida de accesos no autorizados.
  - **Integridad:** que la información esté segura de cambios corruptos manejados sin autorización
  - **Disponibilidad:** que la información esté disponible y según el nivel de acceso que tenga cada usuario
- **Requerimientos de Usabilidad:** describen los niveles apropiados de usabilidad, dados los usuarios finales del producto, para ello deben revisarse las especificaciones de los perfiles de usuarios y las clasificaciones de sus niveles de experiencia
- **Requerimientos de Soporte:** abarcan todas las acciones a tomar una vez que se ha terminado el desarrollo del software con motivos de asistir a los clientes de este, así como lograr su mejoramiento progresivo y evolución en el tiempo

Actividades de los requerimientos

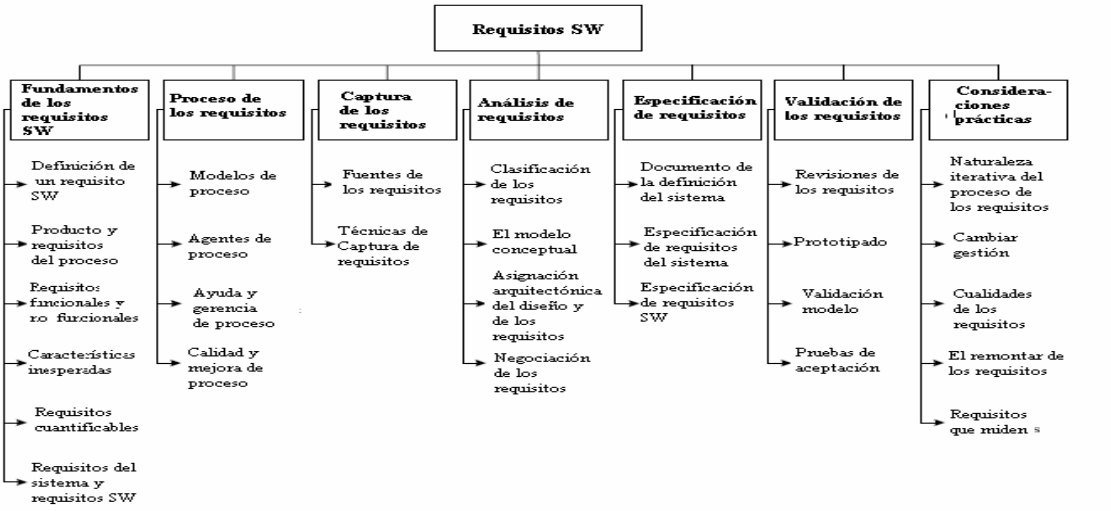


Figura 4. Descomposición de Materias para el Área de Conocimiento (KA) de Requisitos de Software. (Society, 2004)

Captura de los requisitos

La captura de los requisitos se refiere a de donde vienen los requisitos del software y cómo el ingeniero de software puede recogerlos. Es la primera etapa en la construcción de una comprensión del problema que el software requiere solucionar. Es fundamental una actividad humana, y es donde identifican a los stakeholders y las relaciones se establecen entre el equipo de desarrollo y el cliente. También se conoce como "descubrimiento de los requisitos," y "adquisición de los requisitos." (Society, 2004)

- **Técnicas de Captura de requisitos** se realiza una vez que se hayan identificado las fuentes de los requisitos, ingenieros de software pueden comenzar a sacar requisitos de ellos. Este asunto se concentra en las técnicas para conseguir que los stakeholders articulen sus requisitos. (Society, 2004)

  Las técnicas más habituales en la elicitación de requisitos son las entrevistas, Joint Application Development (JAD) o Desarrollo Conjunto de Aplicaciones, Brainstorming o tormenta de ideas y la utilización de escenarios, más conocidos como casos de uso. Además de estas técnicas existen otras complementarias para el apoyo de estas como la observación in situ, el estudio de

documentación, los cuestionarios, la inmersión en el negocio del cliente o haciendo que los ingenieros de requisitos sean aprendices del cliente. (Jiménez, 2000)

- **Entrevistas:** Las entrevistas son la técnica de elicitación más utilizada, y de hecho son prácticamente inevitables en cualquier desarrollo ya que son una de las formas de comunicación más naturales entre personas. En las entrevistas se pueden identificar tres fases: preparación, realización y análisis.
  - ✓ **Preparación de entrevistas:** Las entrevistas no deben improvisarse, por lo que conviene realizar las siguiente tareas previas:
    - **Estudiar el dominio del problema**: conocer las categorías y conceptos de la comunidad de clientes y usuarios es fundamental para poder entender las necesidades de dicha comunidad y su forma de expresarlas.
    - **Seleccionar a las personas a las que se va a entrevistar**: se debe minimizar el número de entrevistas a realizar, por lo que es fundamental seleccionar a las personas a entrevistar. Normalmente se comienza por los directivos, que pueden ofrecer una visión global, y se continúa con los futuros usuarios.
    - **Determinar el objetivo y contenido de las entrevistas**: para minimizar el tiempo de la entrevista es fundamental fijar el objetivo que se pretende alcanzar y determinar previamente su contenido.
    - **Planificar las entrevistas**: la fecha, hora, lugar y duración de las entrevista deben fijarse teniendo en cuenta siempre la agenda del entrevistado. En general, se deben buscar sitios agradables donde no se produzcan interrupciones y que resulten naturales a los entrevistados.
  - ✓ **Realización de entrevistas:** Dentro de la realización de las entrevistas se distinguen tres etapas:
    - **Apertura**: el entrevistador debe presentarse e informar al entrevistado sobre la razón de la entrevista, qué se espera conseguir, cómo se utilizará la información, entre otros.
    - **Desarrollo**: la entrevista en sí no debería durar más de dos horas, distribuyendo el tiempo en un 20% para el entrevistador y un 80%para el entrevistado.

- **Utilizar palabras apropiadas**: se deben evitar tecnicismos que no conozca el entrevistado y palabras o frases que puedan perturbar emocionalmente la comunicación.
- **Mostrar interés en todo momento**: es fundamental cuidar la comunicación no verbal durante la entrevista: tono de voz, movimiento, expresión facial, entre otros.
- **Terminación**: al terminar la entrevista se debe recapitular para confirmar que no ha habido confusiones en la información recogida, agradecer al entrevistado su colaboración y citarle para una nueva entrevista si fuera necesario, dejando siempre abierta la posibilidad de volver a contactar para aclarar dudas que surjan al estudiar la información o al contrastarla con otros entrevistados.

✓ **Análisis de las entrevistas:** Una vez realizada la entrevista es necesario leer las notas tomadas, pasarlas a limpio, reorganizar la información, contrastarla con otras entrevistas o fuentes de información, entre otros. Una vez elaborada la información, se puede enviar al entrevistado para confirmar los contenidos. También es importante evaluar la propia entrevista para determinar los aspectos mejorables. Del análisis de lo descrito se desprende todo el proceso de seguimiento a los requisitos y por ende la construcción del producto de software.

- **Joint Application Development:** Es una alternativa a las entrevistas individuales que se desarrolla a lo largo de un conjunto de reuniones en grupo durante un período de 2 a 4 días. En estas reuniones se ayuda a los clientes y usuarios a formular problemas y explorar posibles soluciones, involucrándolos y haciéndolos sentirse partícipes del desarrollo.

  JAD tiene dos grandes pasos, el JAD/Plan cuyo objetivo es elicitar y especificar requisitos, y el JAD/Design, en el que se aborda el diseño del Software.

  La aplicación de esta técnica aporta grandes ventajas al proceso de desarrollo entre las que se pueden relacionar: ahorra tiempo al evitar que las opiniones de los clientes se contrasten por separado; todo el grupo, incluyendo los clientes y los futuros usuarios, revisa la documentación generada, no solo los ingenieros de requisitos; implica más a los clientes y usuarios en el desarrollo.
- **Brainstorming:** brainstorming o tormenta de ideas es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios.

Como técnica de elicitación de requisitos, el brainstorming puede ayudar a generar una gran variedad de vistas del problema y a formularlo de diferentes formas, sobre todo al comienzo del proceso de elicitación, cuando los requisitos son todavía muy difusos.

- **Casos de uso:** Los casos de uso son una técnica para la especificación de requisitos funcionales. Los casos de uso presentan ciertas ventajas sobre la descripción meramente textual de los requisitos funcionales, ya que facilitan la elicitación de requisitos y son fácilmente comprensibles por los clientes y usuarios. Además, pueden servir de base a las pruebas del sistema y a la documentación para los usuarios.

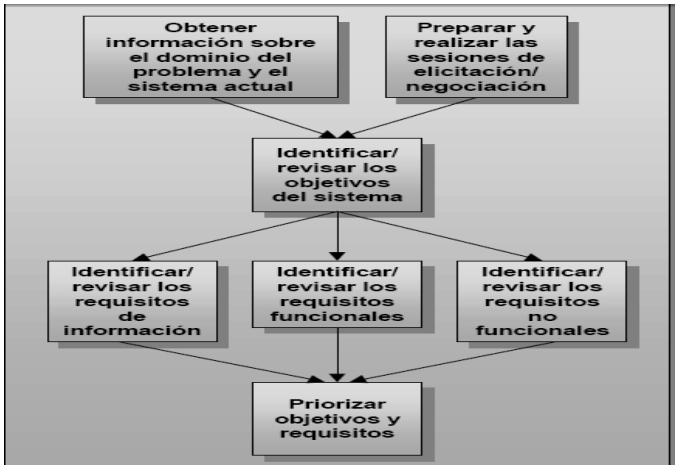


Figura 5 Tareas de Elicitación de requisitos.

Análisis de requisitos

Este asunto se refiere al proceso de analizar requisitos para detectar y resolver los conflictos entre los requisitos, descubrir los límites del software y cómo debe obrar recíprocamente con su ambiente además elaborar los requisitos del sistema para derivar requisitos software. (Society, 2004)

Una vez recopilados los requisitos, se agrupan por categorías y se organizan en subconjuntos, se estudia cada requisito en relación con el resto, se examinan los requisitos en su consistencia, completitud y ambigüedad, y se clasifican en base a las necesidades de los clientes/usuarios. (Pressman, 2005)

Especificación de requisitos

Para la mayoría de las profesiones de la ingeniería, el término “especificación” se refiere a la asignación de valores o límites numéricos para metas del diseño del producto. (Society, 2004)

La especificación de requisitos del software permite un riguroso gravamen de requisitos antes de que el diseño pueda comenzar y reducir un reajuste final. Debe también proporcionar una base realista para estimar costes y riesgos del producto. Las organizaciones pueden también utilizar un documento de especificación de requisitos software para desarrollar su propia validación y que la verificación sea más productiva. (Society, 2004)

Validación de los requisitos

Los documentos de los requisitos pueden estar conformes a la validación y procedimientos de verificación. Los requisitos pueden ser validados para asegurarse de que el ingeniero del software entiende los requisitos, y es también importante para verificar que un documento de requisitos se conforma con la compañía de los estándares, y éste es comprensible, constante, y finito. (Society, 2004)

El resultado del trabajo realizado es una consecuencia de la ingeniería de requisitos (especificación del sistema e información relacionada) y es evaluada su calidad en la fase de validación. La validación de requisitos examina las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados hayan sido corregidos, y que el resultado del trabajo se ajusta a los estándares establecidos para el proceso, el proyecto y el producto. (Pressman, 2005)

Gestión de los Requisitos:

Como parte de las consideraciones prácticas de este proceso se encuentra **Cambiar a gestión**. La gestión del cambio es central en el análisis de requisitos. Este asunto describe el papel de la gestión del cambio, los procedimientos que necesitan estar preparados, y el análisis que se debe aplicar a los cambios propuestos.

Como salidas del proceso de análisis se encuentran diferentes artefactos que son la entrada a la fase de Diseño. En la cual se transforma la información aportada por el análisis a un modelo más cercano al producto en elaboración.

1.12. Diseño

El Diseño del software se encuentra en el núcleo técnico de la ingeniería del software y se aplica independientemente del modelo de diseño de software que se utilice. Una vez que se analizan y especifican los requisitos del software, el diseño del software es la primera de las tres actividades

técnicas-diseño, generación de código y pruebas que se requieren para construir y verificar el software. Cada actividad transforma la información de manera que dé lugar por último a un software de computadora validado.

El diseño es tanto un proceso como un modelo. El proceso de diseño es una secuencia de pasos que hacen posible que el diseñador describa todos los aspectos del software que se va construir. Sin embargo, es importante destacar que el proceso de diseño simplemente no es un recetario. Un conocimiento creativo, experiencia en el tema, un sentido de lo que hace que un software sea bueno, y un compromiso general con la calidad son factores críticos de éxito para un diseño competente.

Los principios básicos de diseño hacen posible que el ingeniero del software navegue por el proceso de diseño. Davis sugiere un conjunto de principios para el diseño del software, los cuales han sido adaptados y ampliados en la lista siguiente: (DAV95)

1. El diseño deberá poderse rastrear hasta el modelo de análisis
2. El diseño no debe inventar nada que ya esté inventado
3. El diseño deberá "minimizar la distancia intelectual entre el software y el problema como si de la misma vida real se tratara
4. El diseño deberá presentar uniformidad e integración
5. El diseño deberá estructurarse para admitir cambios
6. El diseño deberá estructurarse para degradarse poco a poco, incluso cuando se enfrenta con datos, sucesos o condiciones de operación aberrantes
7. El diseño no es escribir código y escribir código no es diseñar
8. El diseño deberá evaluarse en función de la calidad mientras se va creando, no después de terminarlo
9. El diseño deberá revisarse para minimizar los errores conceptuales (semánticos).

Cuando estos principios se aplican correctamente el ingeniero de software crea un diseño que muestra los factores de calidad tanto externos como internos. Como parte de años de estudio en la industria del software han surgido soluciones simples capaces de facilitar todo el proceso de elaboración. Estas soluciones simples son conocidas como patrones.

1.13. Patrones de Casos de Uso y patrones de Diseño

Los patrones de software describen un problema que ocurre repetidas veces en algún contexto determinado del proceso de desarrollo de software, y entregan una buena solución ya probada. Esto ayuda a diseñar correctamente en menos tiempo, ayuda a construir problemas reutilizables y extensibles, y facilita la documentación y la comunicación con otros miembros del equipo de desarrollo.

Los patrones son soluciones simples compuestos por una pareja problema/solución, fundamentadas en la experiencia para problemas específicos y comunes y que se ha demostrado que funcionan y pueden emplearse en diferentes contextos. Algunos de estos son (Övergaard, 2004):

1.13.1. Patrones de Casos de Uso

Los patrones permiten y han permitido en diferentes áreas del conocimiento humano reusar la esencia de la solución de un problema al enfrentar nuevos problemas similares. Es así que los patrones constituyen una especie de mecanismo de registro y concentración de experticia. Uno de los problemas que trae aparejado el uso de patrones en la práctica, es la identificación del patrón más apropiado para el problema en cuestión.

Existen diferentes patrones aplicados a casos de uso entre los que podemos destacar(Övergaard, y otros, 2004):

Concordancia (Commonality) Este patrón toma una subsecuencia de acciones que estén en diferentes partes del flujo de casos de uso y la define por separado.

Concordancia: Re-uso (Commonality: Reuse) El patrón Re-uso es un patrón de estructura que consiste en tres casos de uso. El primero llamado “Sub-secuencia Común”, modela la secuencia de acciones que aparecen en múltiples casos de uso del modelo.

Los otros dos casos de uso comparten de esta sub-secuencia común de acciones (dos es la menor cantidad que puede existir).

La sub-secuencia tiene que estar en un fragmento, es decir, todo lo que requiere estar incluido tiene que estar en un único fragmento completo. Además no se puede hacer referencia desde la sub-secuencia a donde esta es utilizada, porque el caso de uso incluido tiene que ser independiente del caso de uso base.

Concordancia: Adición (Commonality: Addition) En el caso de este patrón alternativo, la subsecuencia común de casos de uso, extiende los casos de uso compartiendo la subsecuencia de acciones. Los otros

casos de uso modelan el flujo que será expandido con la subsecuencia. Este patrón es preferible usarlo cuando otros casos de uso se encuentran propiamente completos, o sea, que no requieren de una subsecuencia común de acciones para modelar los usos completos del sistema.

Extensión Concreta (Concrete Extension or Inclusion: Extension) Extensión Concreta es un patrón de estructura. Patrón que consiste en dos casos de uso y una relación de extensión. El caso de uso extendido es concreto, es decir, puede ser instanciado por su cuenta como por el caso de uso base. Este patrón es aplicable cuando un flujo puede extender el flujo de otro caso de uso, lo que significa que puede ocurrir el proceso del caso de uso base, o puede ocurrir el del caso de uso base con su caso de uso extendido.

Inclusión Concreta (Concrete Extension or Inclusion: Inclusion) Inclusión Concreta es un patrón de estructura. Consiste en dos casos de uso y una relación de inclusión entre el caso de uso base y el caso de uso incluido. Este último puede ser instanciado por sí solo. El caso de uso base puede ser concreto o abstracto. Se utiliza este patrón cuando un flujo de datos puede ser incluido en el flujo de otro caso de uso y también puede ejecutarse por sí solo.

CRUD (Creating, Reading, Updating, Deleting) Es un patrón de estructura que se basa en la fusión de casos de uso simples para formar una unidad conceptual.

Múltiples actores (Multiple Actors) Captura la concordancia entre actores manteniendo roles separados.

Roles diferentes (Distinct Roles) Consiste en un caso de uso y por lo menos dos actores. Este patrón se usa cuando los dos actores juegan papeles diferentes hacia el caso de uso es decir, ellos interactúan de forma diferentemente con el caso del uso.

Roles comunes (Commons Roles) Puede suceder que los dos actores jueguen el mismo rol sobre el CU. Este rol es representado por otro actor, heredado por los actores que comparten este rol. Es aplicable cuando, desde el punto de vista del caso de uso, solo exista una entidad externa interactuando con cada una de las instancias del caso de uso.

1.13.2. Patrones de Diseño

El patrón es una descripción de un problema y su solución que recibe un nombre y que puede emplearse en otros contextos; en teoría, indica la manera de utilizarlo en circunstancias diversas. Muchos patrones

ofrecen orientación sobre como asignar las responsabilidades a los objetos ante determinada categoría de problemas.

Los patrones de diseño son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia.

1.13.2.1. Patrones GRASP

Los Patrones de Software para la asignación General de Responsabilidad (GRASP) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

Booch y Rumbaugh definen la responsabilidad como "un contrato u obligación de un tipo o clase" (BJR97). Las responsabilidades se relacionan con las obligaciones de un objeto respecto a su comportamiento.

Ejemplo de Patrones GRASP

1. Experto
2. Creador
3. Controlador
4. Bajo Acoplamiento
5. Alta Cohesión
6. Polimorfismo
7. Fabricación Pura
8. Indirección
9. No Hables con Extraños

Ver especificación de los cinco primeros patrones en el (Anexo 2)**.**

1.13.2.2. Patrones GOF

Los patrones GOF son los patrones de diseño orientado a objetos más habituales publicados en el libro "Design Patterns" o Diseño de Patrones, escrito por un colectivo de autores que comúnmente se conoce como GoF (Gang of Four, “Pandilla de los Cuatro”).

Este colectivo de autores reúne a los patrones según su propósito en tres clasificaciones, la cual describe la función que cada patrón cumple:

Los patrones según su propósito se pueden clasificar en:

Patrones de Creación: Creación de instancias de objetos.

Patrones Estructurales: Relaciones entre clases, combinación y formación de estructuras mayores.

Patrones de Comportamiento: Interacción y cooperación entre clases.

Patrones de creación

- Abstract Factory (Fábrica Abstracta). Proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas
- Builder (Constructor Virtual). Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones
- Factory Method (Método de Fábrica). Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos
- Prototype (Prototipo). Especifica los tipos de objetos a crear por medio de una instancia prototípica, y crear nuevos objetos copiando este prototipo
- Singleton (Instancia Única). Garantiza que una clase solo tenga una instancia, y proporciona un punto de acceso global a ella

Patrones estructurales

- Adapter (Adaptador). Convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permiten que cooperen clases que de otra manera no podrían por tener interfaces incompatibles
- Bridge (Puente). Desvincula una abstracción de su implementación, de manera que ambas puedan variar de forma independiente
- Composite (Objeto Compuesto). Combina objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos
- Decorator (Envoltorio). Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad

- Facade (Fachada). Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar
- Flyweight (Peso Ligero). Usa el compartimiento para permitir un gran número de objetos de grano fino de forma eficiente
- Proxy. Proporciona un sustituto o representante de otro objeto para controlar el acceso a éste

Patrones de comportamiento

- Chain of Responsibility (Cadena de Responsabilidad). Evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición. Crea una cadena con los objetos receptores y pasa la petición a través de la cadena hasta que esta sea tratada por algún objeto
- Command (Orden). Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer la operaciones.
- Interpreter (Intérprete). Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje
- Iterator (Iterador). Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna
- Mediator (Mediador). Define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente
- Memento (Recuerdo). Representa y externaliza el estado interno de un objeto sin violar la encapsulación, de forma que éste puede volver a dicho estado más tarde.
- Observer (Observador). Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos
- State (Estado). Permite que un objeto modifique su comportamiento cada vez que cambia su estado interno. Parecerá que cambia la clase del objeto
- Strategy (Estrategia). Define una familia de algoritmos, encapsula uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan

- Template Method (Método Plantilla). Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura.
- Visitor (Visitante). Representa una operación sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera.

Un producto analizado y diseñado satisfactoriamente es apto para satisfacer las necesidades del cliente y/o usuarios finales. Pero solo la certeza de una fuerte y hábil elaboración no es suficiente por lo que se hace importante validar técnicamente la calidad de la propuesta presentada. Existen diferentes técnicas para la validación entre las que podemos destacar las métricas.

1.14. Métricas para la validación de la propuesta presentada

Las métricas son la maduración de una disciplina, que, según Pressman van a ayudar a la evaluación de los modelos de análisis y de diseño, en donde proporcionarán una indicación de la complejidad de diseños procedimentales y de código fuente, y ayudarán en el diseño de pruebas más efectivas; Es por eso que propone un proceso de medición, el cual se puede caracterizar por cinco actividades:

- Formulación: La obtención de medidas y métricas del software apropiadas para la representación de software en cuestión.
- Colección: El mecanismo empleado para acumular datos necesarios para obtener las métricas formuladas.
- Análisis: El cálculo de las métricas y la aplicación de herramientas matemáticas.
- Interpretación: La evaluación de los resultados de las métricas en un esfuerzo por conseguir una visión interna de la calidad de la representación.
- Realimentación: Recomendaciones obtenidas de la interpretación de métricas técnicas trasmitidas al equipo de software.

Métricas para Sistemas Orientados a Objetos (Olmedilla Arregui, 2005)**:**

Métricas propuestas por Lorenz y Kidd.

Las métricas orientadas al tamaño se centran en contar los atributos y operaciones de cada clase y los valores para el sistema como un todo son:

- Número de Métodos de Instancia Públicos (PIM).

- Número de Métodos de Instancia (NIM).
- Número de Variables de Instancia (NIV).
- Tamaño de clase (TC).
- Tamaño medio de operación (TMO).
- Número de escenarios (NE).
- Número de clases claves (NCC).
- Número de subsistemas (NSUB).
- Las que se basan en la herencia para ver en la forma que las operaciones se reutilizan en la jerarquía de clases son:
- Número de operaciones redefinidas para una subclase (NOR).
- Número de operaciones añadidas por una subclase (NOA).
- Índice de especialización (IES).
- Número de Métodos Heredados (NMI).

Métricas propuestas por Chidamber y Kemerer (CK).

Es uno de los conjuntos de métricas más difundidos y conocidas como las CK, también se les llama MOOSE. Adoptan tres criterios a la hora de definirlas: capacidad de satisfacer propiedades analíticas, aspecto intuitivo a los profesionales y facilidad para su recogida automática. Definen los siguientes conceptos:

- Métodos ponderados por clase (MPC).
- Profundidad del árbol de herencia (PAH).
- Número de hijos (NDH).
- Acoplamiento entre clases objeto (AEC).
- Respuesta para una clase (RPC).
- Carencia de cohesión en los métodos (CCM).

Se realizó un estudio sobre las métricas existentes y se adoptó por el equipo de desarrollo como técnicas a utilizar en la validación, las métricas para requisitos y casos de uso. Y para el diseño el tamaño de clases TC y el árbol de profundidad de herencia PAH.

1.15. Conclusiones Parciales

En este capítulo se realizó un estudio de las características de los sistemas de gestión fiscal existentes en el mundo. Se realizó un estudio que permitió fundamentar el uso de:

- El Proceso Unificado de Desarrollo como metodología de desarrollo de software con UML como lenguaje de modelado.
- El Visual Paradigm como herramienta para modelar el sistema.
- Las técnicas para capturar los requisitos del sistema, de manera que se obtengan los conocimientos necesarios y se puedan elegir cuales deben ejecutarse para la realización de este trabajo de investigación.
- Los patrones de caso de uso y de diseño, como mecanismos de ayuda para los desarrolladores, ya que facilitan una solución ya probada a un problema que se pueda presentar.

CAPÍTULO 2 ANÁLISIS DEL MÓDULO ÍNDICE DE PELIGROSIDAD PRE DELICTIVA.

2.1. Introducción

En este capítulo se aborda el modelado de Negocio y Sistema. Utilizando la metodología RUP como metodología de desarrollo, permite la realización de varias actividades y la generación de artefactos posibilitando así la solución del problema planteado. Para la descripción de la solución se realiza el modelo de negocio, soportándose la misma en el modelo de casos de uso y en el modelo de objetos. Se realiza la especificación de requisitos funcionales y no funcionales y la modelación del sistema en producción.

2.2. Modelo del Negocio

El modelo de negocio es un Flujo de Trabajo que permite caracterizar el ambiente en que se desarrollará el sistema en construcción, conociendo los procesos y actividades que tienen lugar en la institución. Los objetivos que se persiguen en el mencionado Flujo de Trabajo son : comprender la estructura y la dinámica de la organización, entender los problemas actuales e identificar mejoras potenciales, asegurarse de que los clientes, usuarios finales y desarrolladores tengan una idea común de la organización y derivar los requerimientos del sistema a partir del modelo de negocio que se obtenga (Pressman 2007).

Como parte de la caracterización del negocio se usaron disimiles técnicas para lograr una correcta elicitación de las necesidades de informatización.

2.3. Técnicas usadas para la Captura de Requisitos.

La definición de lo que el sistema debe hacer y como lo debe hacer es un proceso complejo y de gran importancia, es vital que se identifiquen las necesidades de informatización de forma correcta para lograr la mayor satisfacción posible de los clientes y/o usuarios finales.

El proceso de captura de requisitos es un proceso de comunicación entre equipo de desarrollo y clientes. La realización satisfactoria del mismo permite disminuir costos y tardanzas en las entregas de los productos, mejorar la calidad de software y maximizar la satisfacción de los clientes. En la realización de este proceso se utilizaron como técnicas en la captura de requisitos la entrevista y la tormenta de ideas.

Estas técnicas posibilitaron la identificación de varios procesos de negocio.

2.4. Procesos de Negocio

Un proceso de negocio es una colección de actividades estructurales relacionadas, que producen un valor para la organización, sus inversores o sus clientes. Es, por ejemplo, el proceso a través del que una organización ofrece sus servicios a sus clientes (ISO, 2000; pp. 6).

Hay tres tipos de procesos de negocio:

1. Procesos estratégicos - Estos procesos dan orientación al negocio. Por ejemplo, "Planificar estrategia", "Establecer objetivos y metas".
2. Procesos centrales – Estos procesos dan el valor al cliente, son la parte principal del negocio. Por ejemplo, “Repartir mercancías”
3. Procesos de soporte – Estos procesos dan soporte a los procesos centrales. Por ejemplo, “contabilidad”, “Servicio técnico”.

Se identificaron varios procesos básicos dentro del proceso Índices de Peligrosidad Pre-delictiva, que se relacionan a continuación.

Recepcionar Expediente de Índice de Peligrosidad: El proceso tiene lugar cuando el Órgano Instructor hace entrega de un expediente de IPP para la toma de decisiones por parte del fiscal. La secretaria de la Fiscalía distribuye estos expedientes a los fiscales respectivos indicando su prioridad.

Tomar decisión IPP con Enfermos: Proceso en el cual luego de haber revisado las actuaciones el fiscal actuante formaliza la solicitud de medida del pretenso asegurado al tribunal correspondiente.

Tomar decisiones por Conducta Antisocial: Proceso en el cual el fiscal luego de haber recepcionado el expediente de Índices de Peligrosidad Pre-delictiva, se manifiesta con una propuesta de medida para el pretenso asegurado.

Comparecer en el tribunal: Proceso en el cual una vez notificado, el fiscal asiste a la comparecencia en el tribunal, donde se impone una decisión para el pretenso asegurado.

Apelar al tribunal: El proceso tiene lugar una vez que se haya realizado la comparecencia si el fiscal está en desacuerdo con la decisión tomada en dicho evento. Posee un límite de tiempo para proyectarse en desacuerdo.

Devolver a la Fiscalía: Proceso en el cual el tribunal luego de haber recepcionado el expediente de IPP determina que no está completo y lo devuelve a la Fiscalía para que esta complete las actuaciones correspondientes.

Cada uno de los procesos identificados posee una característica en común y es que son inicializados y llevados a cabo por una persona o rol, que comúnmente llamamos actor y trabajador respectivamente.

2.5. Actores del Negocio

Un actor expresa un rol, no una persona. Puede ser un individuo, sistema, entidad, que interactúa con el negocio y que se beneficia de esto. (Rational, 2003)

(Ver Modelo de Casos de Usos del Negocio).

2.6. Trabajadores del Negocio

El trabajador del negocio es una abstracción de un humano o un sistema de software que representa un rol que realiza las actividades de los casos de uso. (Rational, 2003).

(Ver Modelo de Casos de Usos del Negocio).

2.7. Modelo de casos de uso del negocio

(Ver Modelo de Casos de Usos del Negocio).

2.8. Descripción de los casos de uso del negocio

(Ver Modelo de Casos de Usos del Negocio).

2.9. Diagrama de actividades

En UML, un **diagrama de actividades** representa los flujos de trabajo paso a paso de negocio y operacionales de los componentes en un sistema. Un diagrama de actividades muestra el flujo de control general.

Se define un diagrama de actividad como: "… una variación de una máquina de estados, en el cual los estados representan el rendimiento de las acciones o subactividades y las transiciones que provocan por la realización de las acciones o subactividades. (BELLOWS, Jeannie, Castek ,2000).

(Ver Modelo de Casos de Usos del Negocio).

2.10. Modelo de objetos del negocio

(Ver Modelo de Objetos del Negocio).

2.11. Reglas del negocio

Las reglas de negocio describen políticas que deben cumplirse o condiciones que deben satisfacerse.

(Ver Reglas del Negocio).

2.12. Especificación de requisitos

En este apartado se registran los requisitos capturados hasta el momento, y se especifican las capacidades operacionales y funcionales que el sistema deberá tener con el mayor detalle posible. A continuación se especifican tanto los requisitos funcionales como los no funcionales.

2.12.1. Requisitos funcionales

Un **Requisito Funcional** define el comportamiento interno del software: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que muestran cómo los casos de uso serán llevados a la práctica (ISBN 0-7356-1879-8, 2006). Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir.

(Ver Especificación de Requisitos).

2.12.2. Requisitos no funcionales

Un requisito no funcional (NFR) especifica los criterios que se deben usar para juzgar el funcionamiento de un sistema, en lugar de un comportamiento específico. Los requisitos no funcionales verifican cómo un sistema debería de ser. Los Requisitos no funcionales son a menudo llamados las “cualidades de un sistema”.

(Ver Especificación de Requisitos).

2.13. Modelado del sistema

Un caso de uso es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema. Este artefacto contiene actores, casos de uso y sus relaciones y sirve como entrada fundamental para el análisis, diseño y pruebas.

Los casos de uso no son parte del diseño (cómo), sino parte del análisis (qué). De forma que al ser parte del análisis nos ayudan a describir qué es lo que es sistema debe hacer. Los casos de uso son qué hace el sistema desde el punto de vista del usuario. Es decir, describen un uso del sistema y cómo este interactúa con el usuario.

2.14. Actores del sistema

Los actores del sistema suelen ser los trabajadores del negocio y representan a los que interactúan con el sistema.

Los actores modelan cualquier entidad externa que necesite intercambiar información con el sistema. No están restringidos a ser personas físicas por lo que pueden representar otros sistemas externos al actual. Lo esencial es que los actores representan entidades externas al sistema. Además cada uno de estos actores podrá ejecutar una o más tareas en el sistema.

***(Ver Modelo de Casos de Uso del Sistema)*.**

2.15. Patrones de casos de uso

Luego de analizar los requisitos y definir los casos de uso que tendrá el sistema, se representan los actores y casos de uso en el diagrama de casos de uso, que se muestra en el epígrafe siguiente. En este diagrama se usan varios patrones de casos de uso. Estos son:

Concordancia, Adición: En este caso la subsecuencia común de casos de uso, extiende los casos de uso compartiendo la subsecuencia de acciones. Los otros casos de uso modelan el flujo que será expandido con la subsecuencia.

Extensión concreta: Este patrón consiste en dos casos de uso y una relación de extensión entre ellos. El caso de uso extendido es concreto, es decir, este puede ser instanciado por sí solo, así como ser una extensión del caso de uso base. El caso de uso base puede ser concreto o abstracto.

Inclusión concreta: En este patrón existe una relación de inclusión entre el caso de uso base y el caso de uso incluido. Este último puede ser instanciado por sí solo. El caso de uso base puede ser concreto o abstracto.

Múltiples actores, específicamente el patrón Rol común: Pues se destacan actividades comunes para varios actores, por lo que se define un actor del cual heredan los actores con estas funcionalidades comunes.

2.16. Modelo de casos de uso del sistema

(Ver Modelo de Casos de Uso del Sistema).

2.17. Descripción de los casos de uso del sistema

(Ver Especificación de Casos de Uso del Sistema).

2.18. Conclusiones Parciales

Durante la realización del capítulo se logró:

- Modelar los procesos del negocio, permitiendo tener una mejor comprensión de los procesos IPP.
- Capturar los requisitos funcionales y no funcionales del sistema, aplicando técnicas como la tormenta de ideas y la entrevista.
- Aplicar patrones de casos de uso de forma que se pudiera modelar más claro el sistema.
- Generar el modelo del sistema como un artefacto muy importante que sirve de entrada para la realización del diseño.

CAPÍTULO 3 DISEÑO Y VALIDACIÓN DEL MÓDULO ÍNDICE DE PELIGROSIDAD PRE DELELICTIVA.

3.1. Introducción

En este capítulo se realiza el diseño de software del módulo Índices de Peligrosidad Pre-delictiva, se dan a conocer los artefactos generados como son el diagrama de clases, el diagrama de Secuencia y la arquitectura. Se explican los patrones de diseño utilizados en el mismo, también se valida el sistema a través de métricas.

3.2. Modelo del Diseño

El modelo de diseño es un modelo de objetos que describe la realización de los casos de usos centrándose en como los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Además, el modelo de diseño sirve de abstracción de la implementación del sistema y es, de ese modo, utilizado como una entrada fundamental de las actividades de implementación. (Jacobson, Booch y Rumbaugh, 2000)

3.3. Arquitectura definida para el Sistema

En el proyecto SGF la arquitectura seleccionada fue el estilo arquitectónico Modelo Vista Controlador (MVC) el cual es utilizado por el framework Symfony. Este estilo permite la reutilización y la independencia entre las capas, se pueden realizar cambios en capas sin tener que modificar las otras, facilita la estandarización y la utilización de los recursos. Las capas que componen el sistema se listan a continuación:

Symfony está basado en el patrón de arquitectura MVC, que está formado por 3 niveles:

- **El modelo** representa la información con la que trabaja la aplicación, es decir, su lógica de negocio.
- **La vista** transforma el modelo en una página web que permite al usuario interactuar con ella.
- **El controlador** se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista. La Figura 3.1 ilustra el funcionamiento del patrón MVC.

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

controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email, entre otros.). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación.

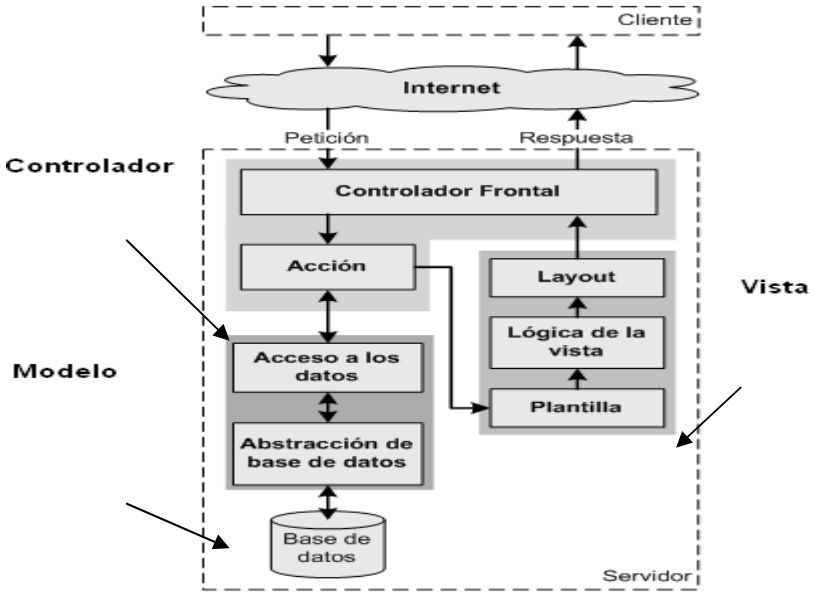


Figura 6 Patrón MVC

3.4. Patrones del Diseño utilizados

Symfony contiene patrones de diseño comunes como son singleton, abstract factory, decorator, así como patrones GRASP a continuación se describen estos patrones que contiene el framework.

Singleton (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia desde cualquier parte del framework.

Abstract Factory (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando. Cuando el framework necesita por ejemplo crear un nuevo objeto para una petición, busca en la definición de la factoría el nombre de la clase que se debe utilizar para esta tarea.

Decorator (Envoltorio): Añade funcionalidad a una clase, dinámicamente. El archivo layout.php, que también se denomina plantilla global, almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el layout.

Patrones GRASP Entre los patrones de asignación de responsabilidad que presenta Symfony están:

Creador En las clases Actions se encuentran las acciones definidas para el Sistema de Gestión Fiscal y se ejecutan cada una de ellas. En las acciones se crean los objetos de las clases que representan las entidades, evidenciando de este modo que la clase Actions es "creador" de dichas entidades.

Experto Este es uno de los más utilizados, puesto que Propel es la librería externa que utiliza Symfony para realizar su capa de abstracción en el modelo, encapsula toda la lógica de los datos y son generadas las clases con todas las funcionalidades comunes de las entidades.

Alta Cohesión Symfony permite asignar responsabilidades con una alta cohesión, por ejemplo la clase actions tiene la responsabilidad de definir las acciones para las plantillas y colabora con otras para realizar diferentes operaciones, instanciar objetos y acceder a las propiedades, es decir, está formada por diferentes funcionalidades que se encuentran estrechamente relacionadas proporcionando que el software sea flexible frente a grandes cambios.

Controlador Todas las peticiones Web son manejadas por un solo controlador frontal (sfActions), que es el punto de entrada único de toda la aplicación en un entorno determinado. Cuando el controlador frontal recibe una petición utiliza el sistema de enrutamiento para asociar el nombre de una acción y el nombre de un módulo con la URL (Localizador Uniforme de Recursos) entrada por el usuario. sfController es la clase del controlador. Se encarga de decodificar la petición y transferirla a la acción correspondiente.

Bajo Acoplamiento Las clases Actions heredan solamente de sfActions para lograr un bajo acoplamiento de clases.

3.5. Descripciones de las Clases de la capa de Negocio

*(*Ver Modelo de Diseño*)*

3.6. Diagramas de clases de diseño

*(*Ver Modelo de Diseño*)*

3.7. Diagramas de Secuencia

*(*Ver Modelo de Diseño*)*

3.8. Validación de la solución propuesta

En este epígrafe se valida el trabajo realizado durante la elaboración del documento mediante métricas. Las métricas tienen por objetivo medir la calidad de los productos intermedios generados en un proyecto de software. Además de las métricas se cuenta con una carta de aceptación por parte del cliente donde especifica su conformidad con el trabajo realizado así como un aval de calidad de la facultad número 15.

(Para ver aval ir a Anexos).

3.8.1. Métricas para evaluar los requisitos

El personal de calidad realizó revisiones técnicas formales durante el levantamiento de los requisitos y aplicó métricas para evaluar la calidad de los mismos. Para determinar la especificidad, ausencia de ambigüedad, de los requisitos se aplica la métrica basada en la consistencia de la interpretación de los revisores para cada requisito:

Q1 = Nu1 / NR, NR = NF + NNF

NR: Es el número de Requisitos que hay en una especificación.

NF: Es el número de Requisitos Funcionales.

NNF: Es el número de Requisitos No Funcionales.

Q1: Consistencia de la interpretación de los revisores

Nu1: Es el número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas. Cuanto más cerca de uno este el valor de Q1 menor será la ambigüedad de la especificación.

NF = 84

NNF = 46

NR = 84 + 46 = 130

Q1 = 128 / 130

Q1 = 0.9846

Después de haber aplicado esta métrica se demostró, gracias a que la mayoría de las interpretaciones de los revisores coincidían, que las especificaciones de los requisitos presentan un alto grado de claridad, ya que los requisitos responden a una única interpretación, pues el valor obtenido Q1 es bastante cercano a 1 y entre más cercano esté este valor a 1 más consistentes están los requisitos.

3.8.2. Métricas para evaluar los casos de uso

Una forma de evaluar la calidad de los casos de uso es mediante métricas. Se tuvieron en cuenta para la validación de los mismos las siguientes propiedades de calidad: consistencia, correctitud, completitud y complejidad que cuentan con un conjunto de factores. Cada uno de estos factores tendrá asociada una o más métricas, que establecen una medida cuantitativa del grado en que los factores indiquen una mala calidad (EAFIT, 2007).

- **Completitud:** Grado en que se ha logrado detallar todos los casos de uso relevantes.
- **Consistencia:** Grado en que los casos de uso del sistema describen las interacciones adecuadas entre el usuario y el sistema.
- **Correctitud:** Grado en que las interacciones actor / sistema soportan adecuadamente el proceso del negocio.
- **Complejidad:** Grado de claridad en la presentación de los elementos que describen el contexto y la claridad del sistema.

Atributo	Factor	Métrica Asociada	Valor
Completitud	Factor 1. ¿Han sido definidos todos los roles relevantes de usuario encargados de generar/ modificar o consultar información?	Métrica 1: Número de roles relevantes omitidos.	Número de roles relevantes omitidos: 0 Se presenta un 100%.
	Factor 2. ¿Se presenta una descripción resumida de todos los casos de uso?	Métrica 2. Número de casos de uso que no tienen descripción resumida.	Número de casos de uso que no tienen descripción resumida 0. Se presenta un 100%
	Factor 3. ¿Están definidos todos los requisitos que dan funcionalidad al caso de Uso?	Métrica 3. Número de requisitos omitidos por caso de Uso. Métrica 4. Número de casos de uso que	Número de requisitos omitidos por caso de uso. 0 Se presenta un 100 %. Número de casos de uso

		tienen requisitos omitidos.	que tienen requisitos omitidos. 0 Se presenta un 100%
	Factor 4. Todos los casos de uso han sido clasificados en Primario, Secundario o auxiliar de acuerdo a su relevancia.	Métrica 5. Número de casos de Uso que no han sido clasificados.	Número de casos de uso que no han sido clasificados 0. Se presenta un 100%
			Se presenta un 100%
Consistencia	Factor 5. ¿Cada caso de uso posee un nombre que describe la funcionalidad que posee en el contexto del usuario?	Métrica 6. Número de Casos de Uso que tienen un nombre incorrecto.	Número de casos de uso con nombre incorrecto 0. Se presenta un 100%.
	Factor 6. ¿Esta adecuadamente redactado el caso de uso en el lenguaje del usuario todo el flujo de eventos?	Métrica 7. Grado de adecuación de la descripción del flujo básico con respecto al lenguaje del usuario.	Todos los casos de uso están descritos en el lenguaje de usuario. Se presenta un 100%
	Factor 7. ¿La descripción del flujo de eventos es iniciada por la acción externa de un usuario o por invocación del sistema?	Métrica 8. Número de casos de uso que no inician por una acción externa o por invocación del sistema.	Número de casos de uso que no son iniciados por una acción externa o invocación del sistema 0. Se presenta un 100 %.
	Factor 8. ¿Existe una separación adecuada entre el flujo normal de eventos y los	Métrica 9. Número de casos de uso que no tienen una separación adecuada entre el flujo	Número de casos de uso que no tienen una separación adecuada entre el flujo normal de

	flujos alternos?	normal de eventos y los flujos alternos.	eventos y los flujos alternos 0. Se presenta un 100%.
			Se presenta un 100 %.
Correctitud	Factor 9. ¿Presenta en caso de uso requisitos comprensibles al usuario?	Métrica 10. Número de casos de Uso que poseen requisitos no comprendidos por el usuario.	Número de casos de uso que presentan requisitos no comprendidos por el usuario 0. Se presenta un 100%
	Factor 10. ¿Las interacciones definidas describen la funcionalidad requerida del sistema?	Métrica 11. Número de casos de uso que poseen interacciones que no describen la funcionalidad requerida del sistema.	Número de casos de Uso que deben ser modificados para adaptarlos a la funcionalidad del sistema 2. Se presenta un 86.66 %
	Factor 11. ¿El diagrama de casos de uso se ajusta a la representación normada en la metodología utilizada?	Métrica 12. Grado en que se ajusta el diagrama de casos de uso a la metodología utilizada en la investigación.	Grado en que se ajusta el diagrama de casos de uso a la metodología. Se presenta en un 100%.
	Factor 12. ¿Las interacciones definidas introducen mejoras al proceso actual?	Métrica 13. Número de casos de uso que deben ser modificados para mejorar el proceso actual.	Número de casos de uso que deben ser modificados para mejorar el proceso actual 0. Se presenta un 100%
			Se presenta un 96.66 %

Complejidad	Factor 13. ¿Los elementos del diagrama están adecuadamente ubicados que facilitan su interpretación?	Métrica 14. Número de elementos del diagrama que requieren reubicación.	Número de elementos del diagrama que requieren reubicación 0 Se presenta un 100%.
			Se presenta un 100 %.

Tabla 3.1 Evaluación de los casos de uso.

El equipo de calidad del proyecto SGF realizó la aplicación de estas técnicas en tres ocasiones que se relacionan a continuación en la Tabla 3.2. Por cada revisión de calidad del proyecto SGF se arrojaron diferentes resultados que fueron tratados para lograr un producto altamente competitivo y con calidad desde el punto de vista técnico y del cliente. Cada propiedad de la métrica tuvo asociada diferentes factores que responden a una descripción del sistema con la calidad requerida. De forma general las propiedades estuvieron por encima del 85 % en la primera revisión y de un 90 % en la segunda, lo que demuestra la calidad del sistema, por otra parte la última revisión demostró valores por encima del 96 % lo que corrobora una acertada calidad en la descripción del sistema en construcción y por ende una garantía del éxito de la implementación del Módulo Índices de Peligrosidad Pre-delictiva.

Nro. de Revisiones	Fecha	Complejidad	Correctitud	Consistencia	Completitud
1	**20/1/2010**	95.12 %	86.80%	100%	100%
2	**12/3/2010**	100%	93.25%	100%	100%
3	**20/5/2010**	100%	96.66%	100%	100%

Tabla3.2 Valores de las métricas por revisión

3.8.3. Métricas para evaluar el diseño

3.8.3.1. Métrica Tamaño de Clase

El tamaño general de una clase se puede determinar empleando métricas para saber el número total de operaciones (tanto operaciones heredadas como privadas de la instancia) que están encapsuladas dentro de la clase y encontrando el número de atributos (tanto atributos heredados como atributos privados de la Instancia) que están encapsulados en la clase. Las medidas o umbrales para los parámetros de calidad

han sido una polémica a nivel mundial en el diseño de sistemas. Algunos especialistas plantean umbrales para estas métricas según como se muestra a continuación, los cuales fueron aplicados al diseño.

Número de Operaciones y/o Atributos	
Métrica TC	**Umbral**
Pequeño	**<= 20**
Medio	**> 20 y <= 30**
Grande	**> 30**

Tabla 3.2 Valores de los umbrales según su clasificación.

Los valores grandes de TC demuestran que una clase puede tener demasiada responsabilidad, lo cual reducirá la reutilizabilidad de la clase y complicará la implementación y la comprobación, por otra parte cuanto menor sea el valor medio para el tamaño, más probable es que las clases existentes dentro del sistema se puedan reutilizar ampliamente.

Resultados. Al aplicar la métrica TC en las clases presentes en la capa del controlador quedaron recogidos los siguientes datos:

Clases	Atributos	Operaciones	Umbral
CrearExpediente/action	**0**	**26**	**Medio**
CrearLibroControl/action	**0**	**1**	**Pequeño**
DatosPretensosAsegurados/action	**0**	**1**	**Pequeño**
ManejarLibroControl/action	**0**	**2**	**Pequeño**
ReasignarTrabajo/action	**0**	**4**	**Pequeño**
ManejarFactura/action	**0**	**3**	**Pequeño**
ManejarReportes/action	**0**	**12**	**Pequeño**
CrearDocumento/action	**0**	**8**	**Pequeño**
BuscarExpediente/action	**0**	**2**	**Pequeño**
ConsultarProceso/action	**0**	**1**	**Pequeño**
RealizarIndicación/action	**0**	**2**	**Pequeño**
AgregarActuación/action	**0**	**1**	**Pequeño**

Tabla 3.3 Clases de la Capa Controlador

La mayoría de las clases que conforman esta capa están dentro de la categoría de pequeñas, solo una presenta umbral medio lo que demuestra que el diseño del sistema no es complejo, quedando demostrada la calidad del diseño.

3.8.3.2. Árbol de Profundidad de Herencia (APH)

Esta métrica se define como la longitud máxima desde el nodo hasta la raíz del árbol. A medida que crece el APH, es más probable que las clases de niveles inferiores hereden muchos métodos. Esto da lugar a posibles dificultades cuando se intenta predecir el comportamiento de una clase. Una jerarquía de clases profunda con un valor grande de APH, lleva también a una mayor complejidad del diseño. Por el lado positivo, los valores grandes de APH implican que se pueden reutilizar muchos métodos. Por su parte, algunos autores como Lorenz y Kidd sugieren que un umbral de 6 niveles como indicador es un abuso en la herencia en distintos lenguajes de programación.

Resultado. A partir de los datos obtenidos al aplicar la métrica APH se obtuvo que el nivel de profundidad de herencia más alto entre las clases del diseño es dos, por lo que queda demostrado que el diseño no es complejo, ya que existe un bajo acoplamiento entre las clases y no es difícil su mantenimiento, pues el nivel de profundidad de la herencia es el mínimo que puede existir en la relación entre clases, este resultado conlleva a que el diseño realizado tiene calidad.

3.9. Conclusiones Parciales

En el presente capítulo se realizó el diseño del módulo IPP, generándose los diagramas de clases del diseño y los diagramas de secuencia de cada uno de los casos de uso del sistema. Se aplicaron patrones de diseño, patrones presentes en el frameworks utilizado para el desarrollo del sistema, que permiten el desarrollo de forma ágil y eficaz. Se aplicaron métricas para evaluar la especificación de los requisitos, el modelo del sistema y el diseño realizado posibilitando que el diseño presente un bajo acoplamiento y una gran robustez, lo que garantiza una mayor confiabilidad y eficiencia al implementar el sistema.

CONCLUSIONES GENERALES

- Se realizó un estudio de las características de los sistemas de gestión fiscal existentes en el mundo, las herramientas CASE, el lenguaje de modelado y de programación, la metodología a tener en cuenta, los patrones de casos de uso y de diseño, y las métricas para evaluar la calidad. Esto permitió la preparación teórica que sustenta el desarrollo del trabajo.
- Se aplicaron técnicas para la captura de los requisitos, tanto funcionales como no funcionales, del sistema. Se tuvieron en cuenta además, patrones para modelar tanto el sistema como el diseño, lo que permitió generar artefactos empleando buenas prácticas necesarias para desarrollar con mayor robustez.
- La calidad de los principales artefactos generados, requisitos, casos de uso del sistema y el diseño fueron evaluadas mediante las métricas definidas en este trabajo, lo que permitió confirmar la realización de artefactos confiables y con calidad.

RECOMENDACIONES

Se recomienda realizar la implementación del módulo Índices de Peligrosidad Pre-delictiva del Sistema Gestión Fiscal, dada la propuesta que se presenta en este trabajo, con el objetivo de obtener una versión del producto.

BIBLIOGRAFÍA

Addison Wesley. Pressman, Roger. *Ingeniería del software: Un enfoque práctico.* 2005.

Ivar Jacobson, Grady Booch, Addison Wesley. *El Proceso Unificado de Desarrollo de Software.* Madrid : s.n., 2000.

Sommerville, I., Ingeniería de Software, Pearson Educación, 2002.

Society, IEEE Computer. *Software Engineering Body of Knowledge(SWebok).* 2004.

Pereira, L. J. (2005). *Algunos Procedimientos Especiales.*

Martin, R. C. (1995). *Designing Object-Oriented C++ Applications using the Booch Method. Prentice-Hal.*

SCHMULLER, **J**. *Aprendiendo UML en 24 horas*. México, 2000.

Fabien Potencier, François Zaninotto. *Symfony, la guía definitiva.* 2007.

LARMAN, C. *UML y Patrones. Introducción al análisis y diseño orientado a objetos*.

REFERENCIAS BIBLIOGRÁFICAS

Abogest. 2010. [En línea] 2010. [Citado el: 8 de Febrero de 2010.] http://www.abogest.com/..

Addison Wesley, Pressman, Roger. 2005.. *Ingeniería del software: Un enfoque práctico.* 2005.

ASP. 2010. [En línea] 2010. [Citado el: 10 de Febrero de 2010.] http://www.fpdv2006.bligoo.com/content/view/276896/ASP-VENTAJAS-Y-DESVENTAJAS.html..

Boehm, B. W. 1976. *Software Engineering.* 1976.

CakePHP. 2010. [En línea] 2010. [Citado el: 11 de Febrero de 2010.] http://book.cakephp.org/es/compare/13/Basic-Principles-of-CakePHP..

CodeIgniter. 2010. [En línea] 2010. [Citado el: 11 de Febrero de 2010.] http://techtastico.com/post/manual-codeigniter-castellano/.

Directum. 2010. [En línea] 2010. [Citado el: 16 de Febrero de 2010.] http://procesosjuridicos.com/.

Enterprice Architect. 2010. [En línea] 2010. [Citado el: 10 de Febrero de 2010.] http://www.sparxsystems.com.ar/products/ea.html.. 10.

Fabien Potencier, François Zaninotto. 2007. *Symfony, la guía definitiva.* 2007.

Framework. 2010. [En línea] 2010. [Citado el: 11 de Febrero de 2010.] http://dokeoslatinoamerica.wordpress.com/2009/01/14/algunos-frameworks-para-php-mas-usados/..

Framework & PHP. 2010. [En línea] 2010. [Citado el: 15 de Abril de 2009.] http://www.maestrosdelweb.com/editorial/los-frameworks-de-php-agilizan-tu-trabajo/..

Framework (1). 2010. [En línea] 2010. [Citado el: 18 de Abril de 2010.] http://euphoriait.com/articulos/framework_web.

GEDEX. 2010. [En línea] 2010. [Citado el: 8 de Febrero de 2010.] http://gedex.net/..

Herramientas CASE. 2010. [En línea] 2010. [Citado el: 13 de Febrero de 2010.] http://www.cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro/c5/c5.htm..

IEEE. 1993. *Standards Collection: Software Engineering.* s.l. : IEEE Standard, 1993.

IURISIS. 2010. [En línea] 2010. [Citado el: 8 de Febrero de 2010.] http://iurisis.es.tripod.com/iurisis/id2.html..

Ivar Jacobson, Grady Booch, Addison Wesley. 2000. *El Proceso Unificado de Desarrollo de Software. .* Madrid : s.n., 2000.

JAVA. 2010. [En línea] 2010. [Citado el: 27 de Febrero de 2010.] http://www.exes.es/ManJava/index.asp?Pg=java_inicial_4_1.html.

Jiménez, Beatriz Bernárdez, Toro, Amador Durán. 2000. Metodología para la Elicitación de Requisitos de Sistemas Software. Octubre de 2000.

José H. Canós, Patricio Letelier yMª Carmen Penadés. *Métodologías Ágiles en el Desarrollo de Software.* DSIC -Universidad Politécnica de Valencia.

Kendall, Kendall. 1997.. *Analisis y diseño de sistemas.* 1997.

Lenguaje de Programación. 2010. [En línea] 2010. [Citado el: 11 de Febrero de 2010.] http://www.buenastareas.com/ensayos/Lenguajes-De-Programacion/104842.html.

Lenguajes de Modelado. 2010. [En línea] 2010. [Citado el: 16 de Febrero de 2010.] http://www.gestiopolis.com/administracion-estrategia/lenguajes-notaciones-y-herramientas-en-analisis-de-procesos.htm..

Martin, R. C. 1995. *Designing Object-Oriented C++ Applications using the Booch Method.* s.l. : Prentice-Hal., 1995.

MSF. 2010. [En línea] 2010. [Citado el: 18 de Febrero de 2010.] http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html..

Olmedilla Arregui, Juan José. 2005. *Revisión Sistemática de Métricas de Diseño Orientado a Objetos.* s.l. : Universidad Politécnica de Madrid, Facultad.de Informática : s.n., 2005.

Övergaard, Gunnar y Palmkvist, Karin. 2004. *Use Cases Patterns and Blueprints.* s.l. : Addison Wesley Professional, 2004.

Penadés, José H. Canós Patricio Letelier yMª Carmen. *Métodologías Ágiles en el Desarrollo de Software.* s.l. : Universidad Politécnica de Valencia. pág. Universidad Politécnica de Valencia.

Pereira, L. J. 2005. *Algunos Procedimientos Especiales.* 2005.

Perl. 2010. [En línea] 2010. [Citado el: 8 de Febrero de 2010.] http://www.ulpgc.es/otros/tutoriales/perl/cap1.htm.

PHP. 2010. [En línea] 2010. [Citado el: 14 de Febrero de 2010.] http://www.cursosypostgrados.com/noticias/los-lenguajes-de-programacion-con-mas-demanda-13588.html..

PHP (1). 2010. [En línea] 2010. [Citado el: 14 de Febrero de 2010.] http://www.linuxcentro.net/linux/staticpages/index.php?page=CaracteristicasPHP..

PHP (2). 2010. [En línea] 2010. [Citado el: 14 de Febrero de 2010.] http://www.silicontower.net/alojamiento-web/caracteristicas-detalladas/lenguajes-programacion..

Procesos Jurídicos. 2010. [En línea] 2010. [Citado el: 7 de Febrero de 2010.] http://procesosjuridicos.com..

Rational Rose. 2010. [En línea] 2010. [Citado el: 10 de Febrero de 2010.] http://www.rational.com.ar/herramientas/roseenterprise.html.

Roberth G. Figueroa, Camilo J. Solís. *METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES.* Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación.

RUP (1). 2010. [En línea] 2010. [Citado el: 2010 de febrero de 24.] http://metodologiaxpvsmetodologiarup.blogspot.com/.

RUP (2). 2010. [En línea] 2010. [Citado el: 2010 de febrero de 24.] http://jeolcr.blogspot.com/2009/06/metodologias-rup.html.

Scribd. 2010. [En línea] 2010. [Citado el: 19 de Febrero de 2010.] http://www.scribd.com/doc/31440864/Metodologia-RUP.

Scrum. 2010. [En línea] 2010. [Citado el: 12 de Febrero de 2010.] http://www.controlchaos.com..

Society, IEEE Computer. 2004. *Software Engineering Body of Knowledge(SWebok).* 2004.

Software para Abogados. 2010. [En línea] 2010. [Citado el: 9 de Febrero de 2010.] http://www.softwarepractico.com/AbogadosProcesosJuridicos.htm..

Sommerville, I. 2002. *Ingeniería de Software.* s.l. : Pearson Educación, 2002.

Symfony. 2010. [En línea] 2010. [Citado el: 20 de Febrero de 2010.] http://www.symfony.es/que-es-symfony.

Symfony (1). 2010. [En línea] 2010. [Citado el: 11 de Febrero de 2010.] http://www.maestrosdelweb.com/editorial/el-framework-symfony-una-introduccion-practica-i-parte/.

Vigilancia Judicial. 2010. [En línea] 2010. [Citado el: 8 de Febrero de 2010.] http://www.vigilanciajudicial.com/..

Visual Paradigm. 2010. [En línea] 2010. [Citado el: Febrero de 10 de 2010.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_(MÍ)_14720_p/..

Visual Paradigm (1). 2010. [En línea] 2010. [Citado el: 12 de Febrero de 2010.] http://www.visual-paradigm.com/product/vpuml.

XP. 2010. [En línea] 2010. [Citado el: 12 de Febrero de 2010.] http://www.extremeprogramming.org..

Zend Framework. 2010. [En línea] 2010. [Citado el: 11 de Febrero de 2010.] http://sentidoweb.com/2006/07/13/introduccion-al-zend-framework.php.

ANEXOS

Anexo 1. Aplicaciones Informáticas para Procesos Jurídicos.

IuriSis

Es un programa desarrollado especialmente para la administración de despachos judiciales, oficinas de abogados y todas aquellas dependencias que tengan que ver con los diferentes asuntos del mundo jurídico, tales como juzgados, tribunales, bufetes jurídicos, oficinas de conciliación y arbitraje, facultades de derecho.

GEDEX

Realiza el seguimiento completo de los expedientes jurídicos de su despacho, bufete o departamento jurídico. Es el sistema más rápido para buscar y localizar sus casos, con diferencia, ayudará a incrementar beneficios y ahorrar en costes de gestión. Evalúa cuánto puede beneficiarse (en tiempo y dinero) al informatizar sus expedientes.

Anexo 2. Patrones GRASP

Experto
Problema: ¿Quién asumirá la responsabilidad en el caso general?
Solución: Asignar una responsabilidad al experto en información: la clase que posee la información necesaria para cumplir con la responsabilidad.
Beneficios: Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento. El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases "sencillas" y más cohesivas que son más fáciles de comprender y de mantener. Así se brinda soporte a una alta cohesión.
Creador
Problema: ¿Quién crea?
Solución: Asignar a la clase B la responsabilidad de crear una instancia de clase A, si se cumple una de las siguientes condiciones:

1. B contiene A
2. B agrega A
3. B tiene los datos de inicialización de A
4. B registra A
5. B utiliza A muy cerca.

Beneficio: Se brinda soporte a un bajo acoplamiento (que describiremos más adelante), lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Es probable que el acoplamiento no aumente, pues la clase creada tiende a ser visible a la clase creador, debido a las asociaciones actuales que nos llevaron a elegirla como el parámetro adecuado.

Controlador

Problema: ¿Quién administra un evento del sistema?

Solución: Asignar la responsabilidad de administrar un mensaje de eventos del sistema a una clase que represente una de las siguientes opciones:

- ✓ El negocio o la organización global (un controlador de fachada)
- ✓ El "sistema" global (un controlador de fachada)
- ✓ Un ser animado del dominio que realice el trabajo (un controlador de papeles)
- ✓ Una clase artificial (Fabricación Pura) que represente el caso de uso (un controlador de casos de uso)

Beneficios:

- ✓ Mayor potencial de los componentes reutilizables
- ✓ Reflexionar sobre el estado del caso de uso

Bajo Acoplamiento

Problema: ¿Cómo dar soporte a poca dependencia y a una mayor reutilización?

Solución: Asignar las responsabilidades de modo que se mantenga bajo acoplamiento.

Beneficios:

- ✓ No se afectan por cambios de otros componentes.
- ✓ Fáciles de entender por separado.
- ✓ Fáciles de reutilizar.

Alta Cohesión
Problema: ¿Cómo mantener controlable la complejidad?
Solución: Asignar las responsabilidades de modo que se mantenga una alta cohesión.

Beneficios:

- ✓ Mejoran la claridad y la facilidad con que se entiende el diseño
- ✓ Se simplifican el mantenimiento y las mejoras en funcionalidad
- ✓ A menudo se genera un bajo acoplamiento
- ✓ La ventaja de una gran funcionalidad soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico

GLOSARIO DE TÉRMINOS

IPP: Índices de Peligrosidad Pre-delictiva.

Expediente de Índices de Peligrosidad Pre-delictiva (IPP)*:* Instrumento oficial escrito en el que la autoridad hace constar como cierta alguna cosa. De manera especial se aplica este nombre a las diligencias que se realizan para la averiguación de un índice.

Presunto asegurado: Persona que está siendo tramitada en el proceso de IPP, de la cual todavía no se tiene una certeza final de la medida a aplicar.

Día hábil: día laborable que incluye desde el lunes hasta el viernes y los sábados laborables. No incluye los sábados no laborables, los domingos ni los días feriados del año.

Día natural: son todos los días de la semana incluyendo el sábado y el domingo.

Fiscal: Lo perteneciente al físico, y en este sentido se dice leyes fiscales y derechos fiscales, acción fiscal, entre otros. Encargado de promover los intereses del fisco. El funcionario generalmente letrado, que representa y ejerce el ministerio público en los tribunales, defendiendo judicialmente los intereses de la sociedad y del estado.

Fiscal actuante: Es aquel que a los efectos de la tramitación de las impugnaciones tiene la capacidad legal para emitir el pronunciamiento que corresponda.

LPP: Ley de Procedimiento Penal.

PNR: Policía Nacional Revolucionaria.

FGR: Fiscalía general de la República.

Tribunal: Entiéndase por tribunal no solo el lugar en que se administra justicia, sino también por los jueces que componen el órgano colegiado.

Caso de Uso: es una secuencia de pasos a seguir para la realización de un fin o propósito, y se relaciona directamente con los requerimientos, un Caso de Uso es la secuencia de pasos que conlleva la realización e implementación de un Requerimiento planteado por el Cliente.

Lenguajes de Programación: son un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. (Lenguaje de Programación, 2010)

Métrica: como “una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado*”.*