Universidad de las Ciencias Informáticas

Facultad 3

Título: Análisis y Diseño del Sistema Integral de Documentación e Información Judicial (SIDIJ) para el Tribunal Supremo Popular.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas.

Autores:

Anais Louit Moreno.

Yordanka Carralero Domínguez.

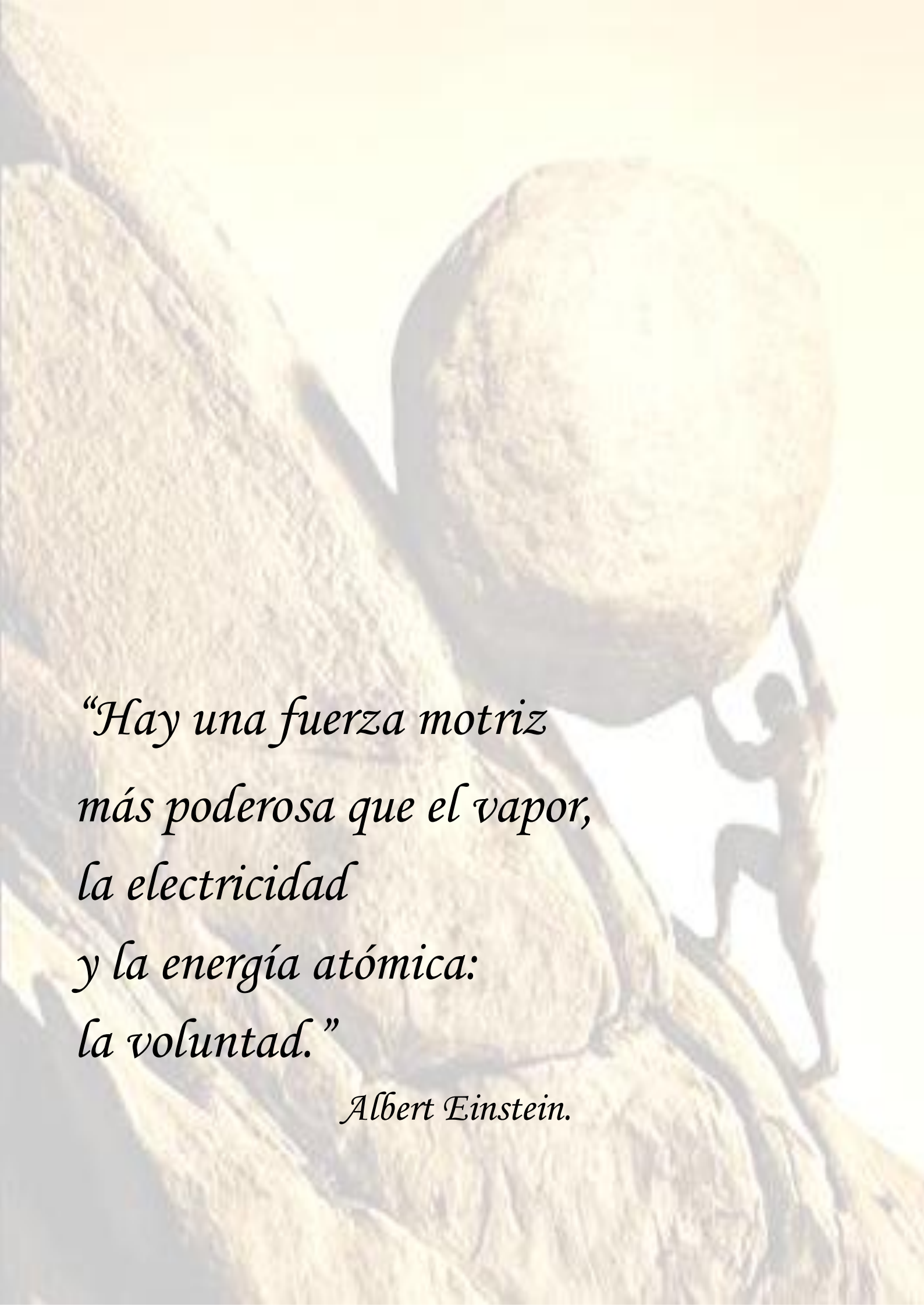
Tutor:

Ing. Yanet Pérez Valcárcel.

Co-tutor:

Ing. Heiler Fabars Corrales.

Ciudad de La Habana, mayo de 2011.
"Año del 52 Aniversario de la Revolución"



"Hay una fuerza motriz
más poderosa que el vapor,
la electricidad
y la energía atómica:
la voluntad."

Albert Einstein.

DECLARACIÓN DE AUTORÍA

Declaramos que somos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste se firma la presente declaración a los __días del mes de _____del año _____.

Autores:

Anais Louit Moreno.

_____

Yordanka Carralero Domínguez.

_____

Tutor:

Ing. Yanet Pérez Valcárcel.

_____

Co-tutor:

Ing. Heiler Fabars Corrales.

_____

Resumen

La información es actualmente el recurso más importante que posee una entidad para su correcto desarrollo y funcionamiento. En las organizaciones o entidades se maneja gran cantidad de documentos, lo que imposibilita gestionarlos de forma manual, por lo que se hace necesario el uso de alguna herramienta informática que permita aumentar el cúmulo de información que se posee, y realizar búsquedas sobre ella en un breve tiempo.

La Universidad de las Ciencias Informáticas (UCI) desde su surgimiento se dio a la tarea de la informatización de la sociedad cubana, así se han ido creando centros en las diferentes facultades que se encargan de áreas específicas de la sociedad; como es el caso de CEGEL que pertenece a la facultad 3, este centro es el encargado de desarrollar sistemas para entidades gubernamentales, lo que da origen a su nombre: Centro de Gobierno Electrónico.

Entre las entidades que informatiza CEGEL se encuentra el Tribunal Supremo Popular que cuenta con el Centro Nacional de Documentación e Información Judicial (CENDIJ), el cual se encarga de proveer servicios de acceso a la información judicial, a la población, al Sistema de Tribunales Populares, y a otros órganos judiciales y afines del país, y que no cuenta con un sistema informático que gestione la información que el centro posee. El presente trabajo desarrolla el análisis y diseño del Sistema Integral de Documentación e Información Judicial (SIDIJ) para el CENDIJ, dicho sistema permitirá almacenar y realizar una rápida búsqueda de la información para una mejor prestación de servicios de la institución facilitando su eficiencia.

Los estudios realizados permitieron seleccionar RUP como la metodología de desarrollo de software, obteniendo finalmente el análisis y diseño validados.

Palabras claves: análisis, diseño.

Índice

Índice

Introducción

El hombre ha necesitado a lo largo de su historia almacenar información, de manera temporal o permanente, en cualquier actividad productiva en la que esté inmerso; por lo cual la información se ha convertido en el mediador principal de todo tipo de actividades y en uno de los recursos más valiosos e importantes con los que cuenta una organización para desarrollar adecuadamente sus procesos, por tanto, es válido decir, que es un recurso clave para cualquier tipo de organización, porque estas deben obtener, procesar, usar y crear información en sus diferentes procesos. A diario se toman decisiones que indican el rumbo de la empresa o proyecto en el que se trabaja, por lo que es necesario basarse en información de calidad.

Para realizar cualquier investigación se hace necesario consultar información variada y completa, por lo que los centros que brindan servicios de acceso a la misma deben nutrirse de grandes volúmenes de documentos, folletos, publicaciones, etc., que permitan realizar análisis entre varias fuentes, con el fin de satisfacer las necesidades del personal que asiste al mismo; pero mientras más amplia sea la colección de escritos que posea la institución, más engorroso será manejar y gestionar de manera manual los mismos, trayendo consigo dificultades para el personal que labora en el centro.

La necesidad de gestionar de forma óptima grandes cantidades de información implica la aplicación de las Tecnologías de la Información y las Comunicaciones (TICs), el surgimiento de estas ha tenido un impacto significativo en prácticamente todas las esferas de la sociedad llegando a convertirse en una herramienta indispensable que brinda la posibilidad de informatizar procesos de cualquier sector, haciendo uso de sistemas que permiten almacenar, y gestionar la información con la que cuenta una organización, ofreciendo la posibilidad de prestar servicios con mayor rapidez.

En Cuba, la Universidad de las Ciencias Informáticas (UCI), que surgió como parte del proyecto de informatización que asume nuestro país, tiene entre sus objetivos principales la formación de ingenieros informáticos altamente calificados, y el desarrollo de la producción de software; para esto vincula el estudio con la investigación y la producción, desarrollando productos informáticos para diferentes entidades nacionales e internacionales a través de proyectos productivos distribuidos entre las diferentes facultades de la universidad. El Centro de Gobierno Electrónico (CEGEL) que pertenece a la facultad 3, se encarga de informatizar los procesos en organizaciones gubernamentales, entre las que se encuentra el Centro Nacional de Documentación e Información Judicial (CENDIJ).

Introducción.

El Tribunal Supremo Popular de la República de Cuba dispuso la creación del CENDIJ, el 3 de marzo del año 2001, institución que almacena un gran cúmulo de información. La creación de este centro fue un paso de gran importancia y utilidad para la administración de justicia, actividad en que el acceso a la información se torna cada día más complejo y por tanto, requiere de una ejecución rápida, oportuna, objetiva y precisa para facilitar la acertada toma de decisiones con la prontitud requerida.

El Centro Nacional de Documentación e Información Judicial tiene entre sus objetivos facilitar el acceso a la información judicial como instrumento de apoyo a la actividad del Sistema de Tribunales Populares e instituciones afines en el país, así como establecer las normas, formatos y herramientas necesarias para la solución, adquisición, análisis y procesamiento de la información.

Un gran cúmulo de información posibilita a los usuarios que acuden al centro, contar con bibliografía variada y de calidad, pero dificulta la rapidez y precisión en la atención a los mismos, al tener que buscar en una inmensa variedad de información, el libro, folleto, revista o documento, que resuelva las necesidad del usuario.

El CENDIJ se encarga de ofrecer información judicial sobre una base sistemática bajo constante actualización, así como prestar servicios de búsqueda sobre la documentación almacenada en el centro; actualmente la realización de estos procesos se hace de forma manual, apoyándose de registros en forma de papel, esto trae problemas para la entidad, representados en: lentitud en la prestación de los servicios definidos por la entidad, la duplicación de la información, un enorme tiempo dedicado a la organización de la documentación y búsqueda de documentos, insatisfacción de los usuarios que en ocasiones no consultan toda la información necesaria porque en el proceso de búsqueda al ser un gran cúmulo de información pueden ser obviados documentos que podrían contener lo que el usuario necesita, así como dificultad para la realización de procesos estadísticos debido al tiempo que se emplea en la organización y búsqueda de la información que se genera diariamente y que se necesita para dichos procesos, además la entidad cuenta con un registro de los fondos documentales existentes en la misma y si esta información sufriera alguna pérdida o daño se afectaría su control. Por lo tanto se hace necesaria la existencia de un Sistema que permita gestionar adecuadamente todos los procesos que se realizan en el CENDIJ.

Partiendo de lo anteriormente planteado se tiene como **problema a resolver:** *¿Cómo obtener los artefactos que permitan a los desarrolladores implementar el Sistema Integral de Documentación e Información Judicial (SIDIJ) para el Tribunal Supremo Popular?*

Para ello se ha definido como **objeto de estudio:** Proceso de desarrollo de software.

Para responder al problema planteado, los autores se trazaron como **objetivo genera**l: Generar los artefactos del análisis y diseño del Sistema Integral de Documentación e Información Judicial (SIDIJ) que permitan el comienzo y continuidad de su implementación.

Mientras que el **campo de acción**, sería:
Análisis y Diseño del Sistema Integral de Documentación e Información Judicial (SIDIJ).

Como **idea a defender** se tiene**:**
Con la realización de los artefactos del análisis y diseño del Sistema Integral de Documentación e Información Judicial (SIDIJ) se podrá dar comienzo y continuidad a la implementación del mismo.

Objetivos Específicos:

- Elaborar el marco teórico de la investigación.
- Realizar el análisis del sistema y su validación.
- Realizar el diseño del sistema y su validación.

Para cumplir con los objetivos y resolver la situación problémica planteada, se proponen las siguientes **tareas de investigación**:

- Selección de la metodología, lenguaje, herramienta y métricas que se utilizarán para construir el sistema.
- Confección del modelo de análisis para definir las clases de interfaz, controladoras y de entidad.
- Confección de los diagramas de interacción.
- Validación del modelo de análisis.
- Confección del modelo de clases del diseño para modelar la interacción entre las clases.
- Confección de los diagramas de diseño web.
- Validación del modelo de diseño.

Los **métodos científicos de investigación** utilizados fueron los siguientes:

Métodos teóricos:

Histórico – lógico: Para el estudio del estado del arte de los sistemas informáticos en los tribunales de justicia, del proceso de desarrollo de software y las técnicas de diseño de software más usadas.

Analítico – sintético: Este método es utilizado para analizar y comprender los procesos de almacenamiento, gestión y búsqueda de información en el CENDIJ para así establecer conclusiones sobre cómo informatizar dichos procesos.

Modelación: Para la creación de modelos y diagramas que reflejen la lógica del Sistema Integral de Documentación e Información Judicial (SIDIJ) en su análisis y diseño.

Métodos empíricos:

Entrevista: Se utiliza para obtener información detallada acerca de cómo ocurren los procesos de almacenamiento, gestión y búsqueda de información en el CENDIJ

Aporte esperado de la investigación:

El sistema propuesto servirá de apoyo a la gestión de la información en el CENDIJ, permitiendo el almacenamiento de la información, su gestión y búsqueda. El aporte del trabajo consiste en la modelación y validación de los artefactos de entrada que permitirán la implementación del Sistema Integral de Documentación e Información Judicial (SIDIJ).

Para el mejor entendimiento del trabajo se decide estructurar el mismo en tres capítulos los cuales se resumen a continuación:

Capítulo 1. Fundamentación teórica.

En este capítulo se tratan aspectos teóricos y técnicos de Análisis y Diseño. Se hace un estudio de los principales patrones existentes para el desarrollo del diseño, así como las metodologías, herramientas y lenguaje más usados en el desarrollo de software, llegando así, a escoger las tecnologías y herramientas que se usarán en el desarrollo del análisis y diseño del SIDIJ.

Capítulo 2. Solución propuesta. Análisis del sistema y validación.

Se muestra el modelo de análisis de todos los casos de uso del sistema, exponiendo los diagramas de clases del análisis, y los correspondientes diagramas de colaboración. Además se valida el análisis propuesto para dar paso al diseño del sistema.

Capítulo 3. Solución propuesta. Diseño del sistema y validación.

Se muestra el modelo de diseño de todos los casos de uso del sistema, a través de los diagramas de clases del diseño y los correspondientes diagramas de secuencia. Además se validan los resultados obtenidos en la etapa de diseño mediante métricas, para determinar la calidad del mismo.

Capítulo 1. Fundamentación Teórica.

1.1 Introducción

En el presente capítulo se brinda una panorámica general de aspectos relacionados con sistemas informáticos de los tribunales de justicia. Se realiza un estudio de algunas metodologías de desarrollo de software, herramientas CASE, lenguajes de programación, frameworks de desarrollo, patrones de diseño, patrones arquitectónicos y métricas, permitiendo la selección de los adecuados para la concepción del sistema.

1.2 Sistemas informáticos en los tribunales de justicia.

Luego de la investigación realizada se pudo observar que los tribunales de justicia de la mayoría de los países utilizan sistemas informáticos para mostrar a los funcionarios de justicia (abogados, jueces, fiscales) y a la población en general, la información judicial que poseen, entre la que se encuentra la constitución, las leyes vigentes en el país, así como libros y folletos que permitan ampliar el conocimiento sobre los procesos judiciales, aunque en su mayoría se centran en ofrecer noticias e información sobre los juicios que se están llevando a cabo. En Cuba el sitio web del Tribunal Supremo Popular no ofrece información judicial, para esto se cuenta con el Centro Nacional de Información Judicial (CENDIJ) el cual ofrece dicha información, pero este no posee una aplicación informática que permita gestionar los documentos, revistas, folletos y libros que posee, para asegurar la rapidez de los procesos que en él se realizan.

1.3 Tendencias y tecnologías actuales de apoyo en la solución del problema.

Para desempeñar un proyecto es necesario investigar las tecnologías que se pueden utilizar, caracterizar a cada una, para así poder escoger basándose en las ventajas de su aplicación las que se utilizarán, teniendo en cuenta las características del entorno donde se desplegará el sistema a desarrollar. A continuación se define el conjunto de tecnologías a utilizar.

1.3.1 Metodologías de desarrollo de software.

Las metodologías imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más eficiente y predecible. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar. A continuación se relaciona un estudio de un conjunto de ellas con el objetivo de determinar la más adecuada para el desarrollo del sistema en cuestión.

Scrum

Es una metodología de desarrollo ágil, pensada para equipos de desarrollos pequeños (no más de 8 personas) que específica pocos artefactos eliminando el "papeleo" innecesario y dedicando más tiempo a la implementación. Posibilita ajustar las funcionalidades en base a la necesidad de negocio del cliente. Tiene un alcance acotado y viable, por lo que puede ser aplicada en equipos integrados, comprometidos con el proyecto y que se auto administran. No es una metodología de análisis, ni de diseño, sino de gestión del trabajo. (Schwaber, y otros, 2003).

Scrum como metodología de desarrollo de software no es recomendable para algunos contextos y proyectos de software con equipos de trabajo grandes que presenten una estructura de relativa complejidad y donde el cliente no pueda estar en colaboración permanente con el equipo.

eXtreme Programming (XP).

La Programación Extrema (del inglés eXtreme Programming) es una metodología ligera de desarrollo de software que se basa en la simplicidad, la comunicación y la reutilización del código desarrollado. Es utilizada para proyectos de corto plazo y equipos de trabajo pequeño. Utiliza la notación UML como estándar para visualizar sus modelos.

Se basa en la simplicidad, pues su codificación y diseños son simples y claros. Otro aspecto que lo distingue es la comunicación, porque los programadores están en constante intercambio con los clientes para satisfacer sus requisitos y responder rápidamente a los cambios de los mismos. La realimentación o reutilización del código desarrollado, ofrece al cliente la posibilidad de obtener un sistema apto a sus necesidades; se le muestra el producto con tiempo, para ser cambiado si es necesario y poder retroceder a una fase anterior, con el propósito de rediseñarlo a su gusto. (Wesley, 2000).

XP define especificaciones de usuario (del inglés UserStories) como base del software a desarrollar, las cuales son escritas por el cliente y describen escenarios sobre el funcionamiento del software. A partir de las UserStories y de la arquitectura, se crea un plan de entregables entre el equipo de desarrollo y el cliente.

Para cada entregable se definen objetivos, los mismos se discuten con el representante del cliente y se definen las iteraciones necesarias para cumplirlos. El resultado de cada iteración es un programa que se transmite al cliente para que lo juzgue y en base a su opinión se definen las siguientes iteraciones.

En XP sólo se programa la funcionalidad que es requerida para el entregable actual, se sigue un diseño evolutivo con la premisa de conseguir la funcionalidad deseada de la forma más sencilla posible.

Figura 1.1 Fases de la Metodología XP.

La metodología XP se basa en:
Pruebas Unitarias: Pruebas realizadas a los principales procesos, de tal manera que adelantándose en algo hacia el futuro, se puedan hacer pruebas de las fallas que pueden ocurrir.

Refabricación: Se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. (Mendoza Sánchez, 2005).

XP tiene como ventaja la comunicación que existe entre sus desarrolladores, además de la programación en pareja que permite que constantemente se esté revisando el código. Esta metodología integra un cliente al equipo, esta característica permite que las dudas que le surjan a los desarrolladores sean resueltas al momento. Una desventaja de XP es que no existe una documentación que pueda ser utilizada en otro momento.

Proceso Unificado de Desarrollo (RUP).

Es un proceso pesado y flexible, que se puede ajustar a las necesidades del proyecto, brinda facilidades relacionadas a la organización y documentación que genera, lo cual contribuye a un mejor entendimiento del equipo de trabajo. Permite desarrollar aplicaciones sacando el máximo provecho de las nuevas tecnologías, mejorando la calidad, el rendimiento, la reutilización, la seguridad y el mantenimiento del software mediante una gestión sistemática de los riesgos. Proporciona guías explícitas para áreas tales como modelado de negocios, arquitectura Web, pruebas y calidad. Optimiza la productividad de los miembros del equipo al poner al alcance la experiencia derivada de miles de proyectos y muchos líderes de la industria. (Jacobson, y otros, 2000).

Las actividades se organizan en grupos lógicos definiéndose 9 flujos de trabajo. En la Figura 1.2 se representan gráficamente los flujos de trabajo y las 4 fases.

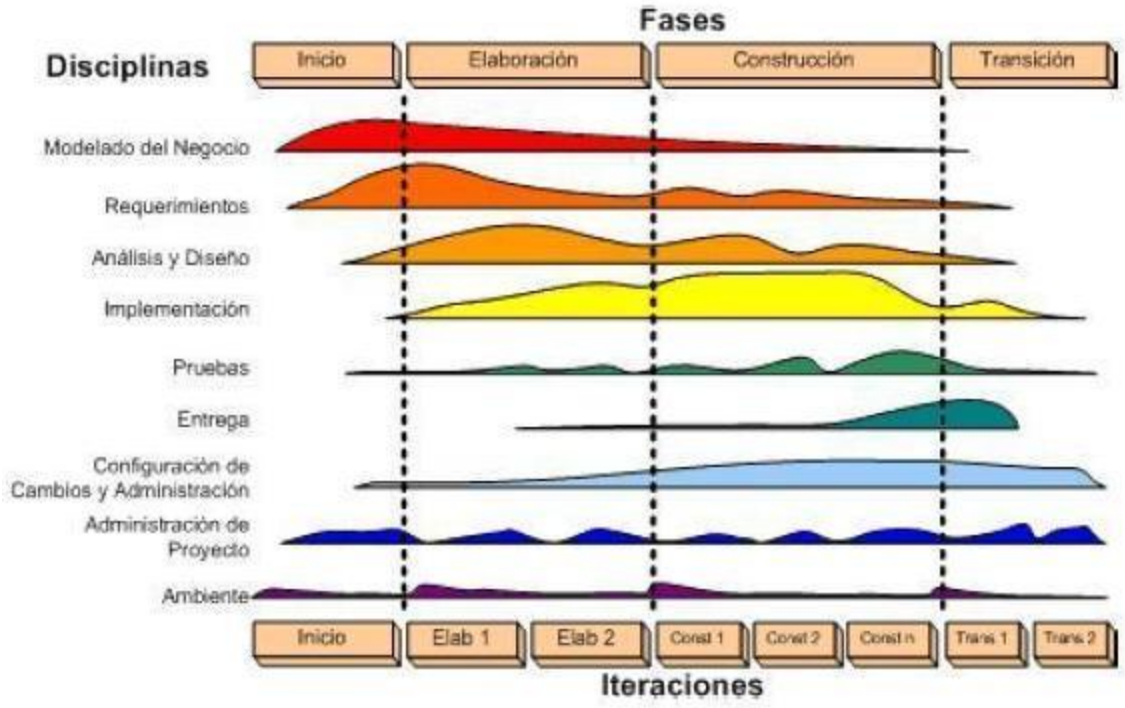


Figura 1.2 Fases y flujos de trabajo de RUP.

El ciclo de vida de RUP se caracteriza por:

1. Dirigido por casos de uso: Estos reflejan las necesidades de los clientes, extraídas en el modelamiento del negocio y representadas después a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo, esto se debe a que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso.

2. Centrado en la arquitectura: La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente.

3. Iterativo e incremental: RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros.

RUP se caracteriza por dividir el ciclo de vida de la producción del software en 4 fases:

1. Inicio: es donde se determina la visión del proyecto, o sea se comprende el entorno y se determina el alcance del producto.

2. Elaboración: en esta etapa se determinan los cimientos de la arquitectura y se analiza el dominio del problema.

3. Construcción: en esta fase se obtiene la capacidad operacional inicial del producto.

4. Transición: se obtiene el release o liberación del producto y se pone en manos de los usuarios finales.

Las fases antes mencionadas se llevan a cabo desarrollando nueve flujos de trabajo principales, los seis primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo:

Flujos de trabajo:

1. Modelado de Negocio

2. Requerimientos

3. Análisis y Diseño

4. Implementación

5. Prueba

6. Despliegue

7. Configuración y Control de Cambios

8. Gestión de Proyectos

9. Entorno

En el Flujo de Análisis y Diseño se traducen los requerimientos a una especificación que describe cómo implementar el sistema.

Incluye artefactos, que son los productos tangibles que conforman el producto final, y roles, papel que desempeña una persona dentro del proceso. Estas características han hecho que junto con el UML (Lenguaje Unificado de Modelado), RUP constituya la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

Metodología de desarrollo de Software escogida: **RUP.**

¿Por qué RUP?

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 del sistema informático que se va a desarrollar.

RUP proporciona calidad, rendimiento, reutilización, seguridad y mantenimiento del software mediante una gestión sistemática de los riesgos. Es flexible, se puede ajustar a las necesidades del proyecto, brinda facilidades relacionadas a la organización y genera gran documentación, lo que contribuye a un mejor entendimiento del equipo de trabajo.

Otras de las razones por las que se elige es por tener el equipo de desarrollo considerable conocimiento en su aplicación y también porque será necesario generar la mayor cantidad de documentación posible del proyecto, además aumenta la productividad del equipo, pues todos los miembros comparten el mismo lenguaje, proceso y visión de cómo desarrollar software.

1.3.2 Herramientas CASE.

Las herramientas CASE son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo

y de dinero. Estas herramientas permiten a los desarrolladores modelar y documentar sus artefactos, cubriendo el ciclo de vida del proceso de desarrollo de software.

Enterprise Architect

Es una herramienta comprensible, de diseño y análisis que utiliza UML, cubriendo el desarrollo de software desde el paso de los requerimientos a través de las etapas del análisis, diseño, pruebas y mantenimiento. 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.

Ayuda a administrar la complejidad con herramientas para rastrear las dependencias, brinda soporte para modelos muy grandes y líneas base por cada punto del tiempo, interfaz intuitiva y de alto rendimiento con vista de proyecto como un "explorador". Ayuda a visualizar sus aplicaciones, soportando ingeniería inversa de un amplio rango de lenguajes de desarrollo de software y esquemas de repositorios de base de datos. El control de versiones es muy lento, por lo menos con svn que es un sistema de control de versiones, que maneja los archivos, las carpetas de un proyecto y sus modificaciones en el transcurso del tiempo, y es complicado hacer combinaciones entre dos importaciones con elementos en común.
Entre sus características se encuentran:

- Realiza transformaciones para DDL, Java, C#.
- Puede generar código fuente C++, Java, C#, VB.Net, Visual Basic, Delphi, PHP, Python y ActionScript.
- Disponible solamente para plataformas Windows.

Enterprise Architect es una herramienta CASE de software propietario. El costo de una licencia para una estación de trabajo es de 194 euros sin contar valores agregados por concepto de mantenimiento.

Rational Rose

Herramienta de modelación visual basada en UML, con plataforma independiente que ayuda a la comunicación entre los miembros de equipo, a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas. Utiliza la notación estándar en la arquitectura de software, la cual permite a los arquitectos y desarrolladores visualizar el sistema completo, utilizando un lenguaje común. Además, los diseñadores pueden modelar sus componentes e interfaces en forma individual, y

luego unirlos con otros componentes del proyecto. Poderosa en el modelado de análisis y diseño de sistemas basados en objetos. Cubre todo ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases. (Kroll, y otros, abril 08, 2003).
El costo de una licencia para un usuario único es de 4,741 euros contando los valores agregados que significa el mantenimiento por 12 meses.

Visual Paradigm

Visual Paradigm para UML, es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Dicha herramienta ofrece un entorno para crear diagramas UML utilizando un lenguaje estándar para todo el equipo de desarrollo, lo que facilita la comunicación entre estos, además de que permite la ingeniería directa e inversa, y es disponible para múltiples plataformas, esta herramienta soporta aplicaciones web, permite generar imágenes y reportes de muy buena calidad, también admite generar código para java y exportarlo como HTML, es muy fácil de instalar y de usar.

Visual Paradigm es una poderosa herramienta CASE que utiliza UML para el modelado.

Características:

- Licencia: Gratuita y Comercial.
- Producto de calidad.
- Soporta el desarrollo de aplicaciones web.
- Varios idiomas.
- Fácil de instalar y actualizar.
- Compatibilidad entre ediciones.
- Disponibilidad en múltiples plataformas (Windows, Linux, entre otras).
- Ofrece herramientas para la generación de reportes en formatos html, pdf y doc.

- Interoperabilidad e integración. Permite la integración con un conjunto de herramientas (Visio drawing, Rational Rose, ERwin Data Modeler Project, Microsoft Excel y Microsoft Word document) e intercambiar diagramas UML y modelos usando representaciones industriales comunes.
- Modelado de base de datos. Proporciona una mayor documentación de la base de datos y diagramas de mapeo de relación de objetos.
- Integración con herramientas para el control de versiones.
- Diseño de prototipo de Interfaz de Usuario. Permite insertar información adicional a los diagramas mediante notas y comentarios para describir sus elementos lo que facilita la revisión de los prototipos así como el trabajo en equipo.

Herramientas CASE escogida: **Visual Paradigm.**

¿Por qué Visual Paradigm?

Se decide emplear como herramienta para realizar la Ingeniería de Software a Visual Paradigm, pues la Universidad ha incrementado los niveles de aceptación de esta por su robustez, usabilidad y portabilidad. Facilita la organización, visualización, integración, diseño y despliegue de los sistemas a través de la representación gráfica.
Además, se cuenta con una licencia, resultando factible el desarrollo desde el punto de vista económico.

1.3.3 Lenguajes de programación.

Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Son herramientas que permiten crear programas y software. Además, facilitan la tarea de programación, porque disponen de formas adecuadas que permiten ser leídas y escritas por personas, a su vez resultan independientes del modelo de computador a utilizar.

Uno de los ejes fundamentales que diferencian a Internet de otros medios de comunicación es la interacción y personalización de la información con el usuario. Esto se logra por medio de lenguajes para programación web que existen hoy en día, los cuales se encuentran tanto del lado del servidor como del lado del cliente. Entre los lenguajes del lado del servidor se puede encontrar a Python, JAVA, y PHP como los más sobresalientes por el auge que han tenido. Estos se caracterizan por desarrollar la lógica de negocio dentro del servidor, además de ser los encargados del acceso a

bases de datos, tratamiento de la información, entre otras funciones. Del lado del cliente se encuentran principalmente JavaScript (JScript) y HTML.

Lenguajes del lado del servidor:

Python

Python es considerado como la "oposición leal" a Perl, lenguaje con el cual mantiene una rivalidad amistosa. Los usuarios de Python consideran a éste mucho más limpio y elegante para programar. Permite dividir el programa en módulos reutilizables desde otros programas. Viene con una gran colección de módulos estándar que se pueden utilizar como base de los programas. Ahorra un tiempo considerable en el desarrollo, pues no es necesario compilar ni enlazar. Es multiparadigma (orientado a objetos, imperativo, funcional), el tipo de dato con que trabaja es fuertemente tipado y dinámico. (González Duque, 2003).

JAVA

Lenguaje orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. Toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Una de las principales características que favoreció el crecimiento y difusión de este lenguaje que el código fuente funciona sobre cualquier plataforma de software y hardware, es decir, un mismo programa puede ejecutarse en varios sistemas sin tocar el código fuente. Brinda una solución para la programación en todo tipo de plataformas y es considerado por muchos como el lenguaje más potente, puesto que permite trabajar tanto a alto como a bajo nivel.

PHP

PHP (HyperText Preprocessor), es un lenguaje interpretado de alto nivel embebido en páginas HTML (Hyper Text Markup Language) y ejecutado en el servidor.

Una de las características más poderosas del PHP es el soporte nativo incluido para una gran cantidad de bases de datos. Entre las bases de datos soportadas se encuentran: MySQL, Oracle, PostgreSQL, MSSQL Server, DBM, DBase, Frontbase, filePro, SQLite entre otras. PHP ofrece además una potente integración con muchas bibliotecas externas, las cuales permiten que el desarrollador realice prácticamente cualquier cosa desde generar documentos en formato PDF (Portable Document Format) hasta analizar código XML (Extensive Markup Language).

Principales ventajas:

- Es un lenguaje multiplataforma.
- Tiene capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL y PostgreSQL.
- Posee una amplia documentación en su página oficial en la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda.
- Posee una de las más grandes comunidades en Internet.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite las técnicas de Programación Orientada a Objetos.
- No requiere definición de tipos de variables.
- Tiene manejo de excepciones.

Actualmente se encuentra en su versión 5, que utiliza el motor Zend-2 con mayor meditación para cubrir las necesidades de las aplicaciones web actuales. Es completamente expansible y se puede ejecutar con Apache, AOLServer y Roxen.

Lenguaje escogido para la programación del lado del servidor: **PHP**.

¿Por qué PHP?

Se escogió PHP para el desarrollo del sistema debido a que es un lenguaje de programación del lado del servidor gratuito e independiente de la plataforma, rápido, con una gran librería de funciones y mucha documentación. Es un lenguaje interpretado, funciona en 11 tipos de servidores, ofrece soporte sobre unas 20 bases de datos y contiene unas 40 extensiones estables sin contar las que se están experimentando. Entre las capacidades de PHP se encuentra su compatibilidad con las bases de datos más comunes, como: MySQL, MSSQL, mSQL, Oracle, Informix, PostgreSQL y ODBC.

PHP es multiplataforma, puede ser utilizado en cualquiera de los principales sistemas operativos del mercado actual y es soportado por la mayoría de los servidores web. Es software libre, lo que implica menos costo y servidores más baratos, por lo que se puede utilizar en proyectos

comerciales, sin tener que pagar por su licencia. El tiempo de aprendizaje de PHP es muy corto gracias a su simplicidad, al igual que el tiempo de desarrollo. (Coding Standar for PHP, 2005).

El desarrollo con PHP no necesita grandes capacidades de hardware. Relacionado a los servidores, una aplicación en PHP no requiere tanta memoria de máquina como podría requerir una aplicación en Java con sus servidores, que podrían necesitar hasta varios procesadores y gigas de memoria RAM. Es muy rápido y su integración con la base de datos PostgreSQL y el servidor Apache, le permite constituirse como una de las alternativas más atractivas del mercado. Su sintaxis está inspirada en C, ligeramente modificada para adaptarlo al entorno en el que trabaja.

Lenguajes del lado del cliente:

En la programación del lado del cliente se decidió integrar el uso de los lenguajes, HTML, CSS y Java Script, debido a las características que se exponen a continuación sobre los mismos.

Lenguaje de Marcas de Hipertexto (HTML)

HTML es el lenguaje de marcado predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. Puede describir la apariencia de un documento e incluir un script (por ejemplo JavaScript). Es compatible con navegadores reconocidos como Internet Explorer, Opera, Firefox, Netscape y Google Chrome.

CSS

CSS (Cascading Style Sheets, u Hojas de Estilo en Cascada) es la tecnología desarrollada por el World Wide Web Consortium (W3C) con el fin de separar la estructura de la presentación. CSS beneficia la accesibilidad, principalmente por la separación de la estructura y la presentación de un documento. Permite un control preciso, fuera del marcado, del espaciado de caracteres, alineación de texto, posicionamiento de objetos en la página, salidas auditivas y habladas, características de fuentes, entre otros. Mediante la separación del estilo y el marcado, se puede simplificar y limpiar el HTML de sus documentos, haciendo al mismo tiempo más accesibles los documentos. CSS permite controlar los tamaños de fuente, color y estilo, y proporciona un control más preciso sobre la presentación de contenido alternativo que HTML solo.

El modo de funcionamiento de las CSS consiste en definir, mediante una sintaxis especial, la forma de presentación que se le aplicará a:

- Una página web, de modo que se pueda definir la forma de esta de una sola vez.

- Un documento HTML o página, se le puede definir la forma en un pequeño trozo de código en la cabecera o a toda la página.
- Una porción del documento.
- Una etiqueta en concreto, llegando incluso a poder definir varios estilos diferentes para una sola etiqueta. Se puede definir, por ejemplo, varios tipos de párrafos: en rojo, en azul, con márgenes, sin ellos, etc.

JavaScript

JavaScript es un lenguaje de scripts desarrollado por Netscape para incrementar las funcionalidades del lenguaje HTML. Es un lenguaje interpretado, es decir, no requiere compilación, el navegador del usuario se encarga de interpretar las sentencias JavaScript contenidas en una página HTML y ejecutarlas adecuadamente. Es un lenguaje que se enfoca en los eventos, por ejemplo: cuando un usuario pincha sobre un enlace o mueve el puntero sobre una imagen se ejecutan acciones en respuesta a estos eventos.

JavaScript es un lenguaje orientado a objetos, su modelo de objetos está reducido y simplificado, pero incluye los elementos necesarios para que los scripts puedan acceder a la información de una página y actuar sobre la interfaz del navegador.

Es dinámico, responde a los eventos en tiempo real, como presionar un botón, pasar el puntero del mouse sobre un determinado texto o el simple hecho de cargar la página o caducar un tiempo. Con esto se puede cambiar totalmente el aspecto de una página al gusto del usuario, evitándose tener en el servidor una página para cada gusto.

1.3.4 Frameworks de desarrollo.

Un framework simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes, facilita la programación de aplicaciones, encapsula operaciones complejas en instrucciones sencillas y proporciona estructura al código fuente, forzando al desarrollador a crear código legible y más fácil de mantener, además proporciona una estructura definida que ayuda a la creación de aplicaciones con mayor rapidez; son desarrollados con el objetivo de brindarles a los programadores y diseñadores una mejor organización y estructura a los proyectos. A continuación se presentan las características de algunos de los frameworks para PHP que fueron analizados.

Zend Frameworks

Se trata de un framework para desarrollar aplicaciones y servicios web con PHP, brinda soluciones para construir sitios modernos, robustos y seguros. Es simple, no necesita instalación especial, requiere PHP 5 e incorpora el patrón Modelo Vista Controlador (MVC). Cuenta con módulos para manejar archivos PDF, Servicios Web (Amazon, Yahoo), etc. Incluye objetos de las diferentes bases de datos, por lo que es simple consultarlas sin tener que escribir ninguna consulta SQL. Tiene robustas clases para autenticación y filtrado de entrada.

CodeIgniter

Utilizado por una gran comunidad de usuarios, construido para codificadores PHP que necesitan una herramienta de desarrollo fácil para crear aplicaciones web simples y elegantes. Entre sus características se puede encontrar su compatibilidad con PHP 4 y PHP 5, incorpora el patrón modelo-vista-controlador que da soporte para múltiples bases de datos, plantillas, validaciones, no requiere instalación, se puede encontrar una librería con un gran número de clases. CodeIgniter es código abierto, muy pequeño (comparándolo con Zend) y posee un acceso a sus librerías bien estructurado. Es liviano y fácil de instalar, sólo hay que descomprimirlo y copiarlo en una carpeta del servidor. Es bastante fácil de aprender, de familiarizarse con el concepto que propone, permite concentrarse en el problema y no en cómo hacerlo, evitando tener que volver a escribir una y otra vez lo mismo.

Symfony

Diseñado para optimizar el desarrollo de las aplicaciones web, se basa en el patrón MVC y separa la lógica de negocio, la lógica de servidor y la de presentación. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación compleja. Automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. Es fácil de instalar y configurar en la mayoría de plataformas (y con la garantía de que funciona correctamente en los sistemas Windows), independiente del sistema gestor de bases de datos. Sigue la mayoría de las mejores prácticas y patrones de diseño para la web.

Los formularios poseen validación automatizada y relleno automático de datos, lo que asegura la obtención de datos correctos y mejora la experiencia del usuario.

Es lo suficientemente flexible como para adaptarse a los casos más complejos, está basado en la premisa de “convenir en vez de configurar”, en la que el desarrollador solo debe configurar aquello

que no es convencional. Está preparado para aplicaciones empresariales y es adaptable a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo. Permite un mantenimiento muy sencillo y es fácil de extender, lo que permite su integración con librerías desarrolladas por terceros como EXT JS.

Framework de desarrollo escogido: **Symfony.**

¿Por qué Symfony?

Luego de haber elegido a PHP 5 como lenguaje de programación se escoge Symfony como framework; este se encuentra desarrollado completamente con PHP 5 y se basa en el patrón MVC (Modelo Vista Controlador) permitiendo separar el código del programa en tres capas, es compatible con la mayoría de los gestores de base de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft, se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows. Es una herramienta multiplataforma que estructura el proyecto en forma de árbol para su mejor entendimiento, ha sido probado en numerosos proyectos reales y se utiliza en sitios una librería de JavaScript que permite construir aplicaciones con interfaces sencillas y dinámicas y que se destaca por:

- Es rápido.
- Está documentado.
- Posee una licencia comercial y otra de código abierto.
- Funciona en la mayoría de los navegadores.

1.3.5 Servidores web.

Es un programa que implementa el Protocolo de Transferencia de Hipertexto (Hypertext Transfer Protocol, HTTP). Programa que se ejecuta continuamente en un ordenador, manteniéndose a la espera de peticiones por parte de un cliente (un navegador web) y que responde a estas peticiones adecuadamente, mediante una página web que se exhibirá en el navegador o mostrando el respectivo mensaje si se detectó algún error.

AOLServer

AOLServer es un servidor HTTP de tipo multihebra, basado en TCL, que incluye facilidades de uso orientadas a entornos de gran escala y a sitios web con contenido dinámico. Hay que destacar que todos los dominios y servidores de AOL, que son más de 200, soportan miles de usuarios simultáneos y millones de conexiones funcionan con AOLServer. Gracias a su integración con

OpenACS, es un software de gestión de contenidos de código libre muy potente, desarrollado por una empresa llamada ArsDigita y liberado bajo licencia GPL.

Roxen

Servidor web de licencia GNU, desarrollado por un grupo sueco que fundó la empresa Roxen Internet Services. Roxen (que antes se llamó Spider y después, Spinner) se destaca por su gran cantidad de funcionalidades. Este servidor, desarrollado en el lenguaje "Pike", ofrece cientos de módulos que permiten el desarrollo sencillo de sitios web muy ricos y dinámicos, sin más herramienta que el servidor Roxen.

Es multiplataforma, puede ejecutarse en Windows, Linux, MAC OS/X y Solaris. Presenta una interfaz de administración basada en web, completa y fácil de usar. Brinda soporte para gráficos integrados que posibilitan, mediante etiquetas de RXML (la extensión de HTML propia de Roxen), la generación de imágenes, títulos y gráficos. El acceso integrado a bases de datos, le permite el acceso a PostgreSQL, Oracle, MySQL, entre otros. Se caracteriza además por un soporte criptográfico fuerte y por su independencia con respecto a la plataforma en los módulos desarrollados por el propio usuario.

Apache

Apache es una de las plataformas de servidores Web más destacadas dentro de la gran cantidad de servidores web que existen en el planeta, de código fuente abierto para plataformas Unix (BSD, GNU/Linux, etc.), Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. Es una tecnología gratuita, multiplataforma, altamente configurable y muy sencilla de utilizar e implementar, está estructurada en módulos, los cuales pueden clasificarse en tres categorías específicas para su utilización con disímiles funcionales.

Módulos Base: Son funciones básicas del Apache. (Van Schermbeek, 2008).

Módulos Multiproceso: Responsables de la unión con los puertos de la máquina, aceptando las peticiones y enviando a los hijos a atender las peticiones. (Van Schermbeek, 2008).

Módulos Adicionales: Cualquier otro módulo que le añada una funcionalidad al servidor se puede añadir sin necesidad de volver a instalar el software. (Van Schermbeek, 2008).

Apache presenta entre otras características bases de datos de autenticación, mensajes de error altamente configurables, negociado de contenido, reescritura de las URL, comprobación de la ortografía de las URL y gran cantidad de manuales on-line.

Servidor web escogido: **Apache.**

¿Por qué Apache?

Se decide seleccionar Apache como servidor Web para la aplicación que se desarrollará, por poseer grandes ventajas como son: sistema modular, código abierto, multi-plataforma, extensible, gratuito y que es muy fácil conseguir ayuda o soporte debido a su gran popularidad lo que hace que millones de servidores reiteren su confianza por sus excelentes prestaciones y las características descritas anteriormente. Además es un servidor Web flexible, rápido y eficiente, continuamente actualizado y adaptado a las últimas versiones de protocolos. Capaz de interpretar diversos lenguajes de programación entre los que se encuentran Perl y PHP, permite personalizar la respuesta ante los posibles errores que pueden surgir en el servidor por ser un servidor robusto, de alta estabilidad y configurabilidad en la creación y gestión de logs.

1.3.6 Sistema Gestor de Bases de Datos.

Se decide utilizar como sistema gestor de Bases de Datos PostgreSQL por ser un sistema de gestión de base de datos relacional, avanzado y orientado a objetos de software. Este gestor posibilita que la aplicación que utilice este sistema, pueda estar en constante actualización y mejora, pudiendo incorporarle todas las facilidades y funcionalidades de su próxima versión. Su distribución es libre, sin necesidad del pago de alguna licencia, lo que constituye una de las ventajas más atractivas. Además aproxima los datos a un modelo objeto-relacional, y es capaz de manejar complejas rutinas y reglas. Ejemplos de su avanzada funcionalidad es el Control de Concurrencia Multi-Versión (Multiversion Concurrency Control (MVCC)), soporte multi-usuario, operadores, métodos de acceso y tipos de datos definidos por el usuario, optimización de consultas, herencia y enlazadores para algunos lenguajes de programación. También tiene soporte para subselects, triggers, vistas y procedimientos almacenados en el servidor.

Además es altamente escalable, tanto en la enorme cantidad de datos que puede manejar y en el número de usuarios concurrentes que puede tolerar. (PostgreSQL, 2008).

Para el estudio de PostgreSQL existe abundante documentación, libros, acceso directo a los desarrolladores, páginas Web que ofrecen información, tutoriales, foros y lo más importante el código fuente.

Se debe señalar que respecto a la selección del Sistema Gestor de Bases de Datos se hace alusión con mayor profundidad en la tesis de los autores: Yamila Zamora Mena y Eugenio Ávila Osoria titulada: “Proceso de migración de datos para el Sistema Integral de Documentación e Información Judicial (SIDIJ) del Tribunal Supremo Popular”.

1.3.7 Arquitectura de software.

La Arquitectura de Software es una disciplina reciente (Szyperski, 2000). Aunque no por ser una normativa joven deja de tener la importancia que requiere.

A medida que crece la complejidad de las aplicaciones, y que se extiende el uso de sistemas distribuidos, los aspectos arquitectónicos del desarrollo de software están recibiendo un interés cada vez mayor, tanto desde la comunidad científica como desde la propia industria del software (Velasco, 2000).

El arquitecto de software contemporáneo, ejecuta una combinación de roles como los de analista de sistemas, diseñador de sistemas e ingeniero de software, pero la arquitectura es más que una recolocación de funciones. El concepto de arquitectura intenta subsumir las actividades de análisis y diseño en un framework de diseño más amplio y más coherente. Es algo más integrado que la suma del análisis por un lado y el diseño por el otro. La integración de metodologías y modelos, es lo que distingue la Arquitectura de Software de la simple yuxtaposición de técnicas de análisis y de diseño. (Albin, 2003).

Bass define Arquitectura de Software de la siguiente forma: “La Arquitectura de Software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos.” (Pressman, 2002).

En la actualidad existen un sin número de definiciones de Arquitectura de Software, pero ninguna que sea adoptada completamente por la totalidad de los arquitectos del mundo. Ninguna definición reprocha a la otra, sino que es otro modo de ver las cosas.

La definición más reconocida internacionalmente por muchos arquitectos tributa a la industria y pertenece a la IEEE 1471, la cual cita así: "La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución". (Reynoso, 2006).

Otra definición muy reconocida es la de Paul Clements: “La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones”. (Paul Clements, 1996).

Existen muchas más definiciones reconocidas para arquitectura de software, entre ellas se encuentra:

La arquitectura concierne a un nivel de abstracción más elevado; se ocupa de componentes y no de procedimientos; de las interacciones entre esos componentes y no de las interfaces; de las restricciones a ejercer sobre los componentes y las interacciones y no de los algoritmos, los procedimientos y los tipos. En cuanto a la composición, la de la arquitectura es de grano grueso, la del diseño es de fina composición procedural; las interacciones entre componentes en arquitectura tienen que ver con un protocolo de alto nivel (en el sentido no técnico de la palabra), mientras que las del diseño conciernen a interacciones de tipo procedural (mensajes, llamadas a rutinas) (Perry, 1997).

Se puede concluir que la Arquitectura de Software, disciplina independiente de cualquier metodología de desarrollo, ocurre en etapas tempranas del desarrollo de un proyecto, es una vista del sistema que se ocupa de los componentes y la conducta de estos en el mismo. Es la organización fundamental del sistema y representa un nivel de abstracción del mismo aportando el más alto nivel de comprensión.

Para la realización del Sistema Integral de Documentación e Información Judicial (SIDIJ) se trabajará con una arquitectura en capas debido a las facilidades que esta ofrece.

Arquitectura en capas

Garlan y Shaw definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior (Reynoso, 2004).

La arquitectura basada en capas se enfoca en la distribución de roles y responsabilidades de forma jerárquica, lo que proporciona una forma muy efectiva de separación de responsabilidades. El rol indica el modo y tipo de interacción con otras capas, y la responsabilidad indica la funcionalidad que está siendo desarrollada. Entre sus principales características se pueden mencionar por ejemplo que describe la descomposición de servicios de forma que la mayoría de la interacción ocurre solamente entre capas vecinas. Además, las capas de una aplicación pueden residir en la misma máquina física (misma capa) o puede estar distribuido sobre diferentes computadores (n-capas). También cabe resaltar que los componentes de cada capa se comunican con otros componentes en otras capas a través de interfaces muy bien definidas; por lo que este modelo ha sido descrito

como una “pirámide invertida de re-uso” donde cada capa agrega responsabilidad y abstracción a la capa directamente sobre ella.

Entre los diseños más empleados se encuentra el de 3 capas, que cuenta con una capa de presentación donde se encuentran las interfaces con las que interactúan los usuarios, captura la información del mismo, muestra los resultados de las peticiones realizadas en un mínimo de proceso y debe tener la característica de ser entendible y fácil de usar para el usuario. La segunda capa se denomina capa de negocio, donde residen los programas que se ejecutan, denominada así porque es aquí donde se establecen todas las reglas que deben cumplirse.

Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación. La última capa se conoce como la capa de datos, donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

La arquitectura en capas y el patrón MVC, pueden relacionarse lógicamente mediante cada uno de sus elementos, donde la capa de presentación podría corresponderse con la Vista, la capa de negocio con el Controlador y la capa de datos con el Modelo.

1.3.8 Patrones de diseño.

Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular (Buschmann, 1996).

Es común que al mismo problema o situación sea aplicable más de un patrón de diseño, ya sea porque son posibles soluciones alternativas, o porque intervienen en partes distintas de la solución. (Ocaña y Sánchez, 2003).

Un sistema bien estructurado está lleno de patrones. (Reynoso, 2005).

Los patrones de diseño tienen como características:

- Son soluciones concretas: Proponen soluciones a problemas concretos, no son teorías genéricas.

- Son soluciones técnicas: Indican soluciones técnicas basadas en Programación Orientada a Objetos (POO). En ocasiones tienen más utilidad con algunos lenguajes de programación y en otras son aplicables a cualquier lenguaje.
- Se utilizan en situaciones frecuentes: se basan en la experiencia acumulada para resolver problemas reiterativos.
- Favorecen la reutilización de código: Ayudan a construir software basado en la reutilización, a construir clases reutilizables. Los propios patrones se reutilizan cada vez que se vuelven a aplicar (Pressman, 2005).
- Los patrones de diseño proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifican y describen formas de solucionar problemas que ocurren de forma frecuente. Están basados en la recopilación del conocimiento de los expertos en el desarrollo de software. Es una experiencia real, probada y que funciona. (Sixto, 2003).

Patrones GRASP (General Responsability Assignment Software Patterns)

Describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades y de cierto modo ayudan a determinar las clases que estarán en el diseño. (Larman, 1999).
A continuación se ofrece una relación de los principales patrones GRASP y el objetivo especifico que cumple cada uno de ellos.

- Experto: La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. De igual manera cada objeto es responsable por mantener su propia información (principio de encapsulamiento) o sea que este conoce y puede informar el valor de sus atributos así como modificarlos.
- Creador: El objeto B tiene la responsabilidad de crear objetos de la clase A si: B agrega objetos A. B contiene objetos A. B registra objetos A. B usa (exhaustivamente) objetos A. B posee la información necesaria para inicializar A.
- Bajo Acoplamiento: El acoplamiento es la medida de cuánto una clase está conectada (tiene conocimiento) de otras clases por lo que éste patrón es evaluativo. Un bajo acoplamiento

permite que el diseño de clases sea más independiente reduciendo el impacto de los cambios y aumentando la reutilización.

- Alta Cohesión: La cohesión funcional dentro de una clase es una medida que indica cuán relacionadas están las responsabilidades de una clase por lo cual éste es un patrón evaluativo. Entre más alta cohesión resulta más fácil de entender, cambiar y reutilizar.
- Controlador: Consiste en asignar la responsabilidad de manejar los mensajes eventos del sistema a una clase facilitando centralizar las actividades (tales como validación, seguridad entre otras).

Patrones GoF

Siguiendo la nomenclatura y organización presentada por GoF (Gamma, 2003), en la clasificación de los distintos tipos de patrones se puede hablar de patrones de creación (Factory Method, Abstract Factory, Builder,…), patrones estructurales (Adapter, Decorator, Bridge,…), y patrones de comportamiento (Strategy, Observer, Template Method, Visitor, Iterator,…).

Patrones de creación

- Abstract Factory: proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.
- Builder: separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.
- Factory Method: 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: especifica los tipos de objetos a crear por medio de una instancia prototípica, y crear nuevos objetos copiando este prototipo.
- Singleton: garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella.

Patrones estructurales

- Adapter: 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: desvincula una abstracción de su implementación, de manera que ambas puedan variar de forma independiente.
- Composite: 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: añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.
- Facade: 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: 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 este.

Patrones de comportamiento

- Chain of Responsibility: evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición.
- Command: 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 las operaciones.
- Interpreter: dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje.
- Iterator: proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.

- Mediator: 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: representa y externaliza el estado interno de un objeto sin violar la encapsulación, de forma que este puede volver a dicho estado más tarde.
- Observer: 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: permite que un objeto modifique su comportamiento cada vez que cambia su estado interno. Parecerá que cambia la clase del objeto.
- Strategy: 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: 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: 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. (Gamma, 2003).

1.3.9 Patrones arquitectónicos.

Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de la solución a ese problema, de tal manera que puedes usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces" (Fowler, 2002).

Los patrones de arquitectura se pueden ver como la descripción de un problema en particular y recurrente de diseño, que aparece en contextos de diseño arquitectónicos específicos, y representa un esquema genérico demostrado con éxito para su solución. (Buschmann, 1996).

Definen la estructura general del software, describen los principios fundamentales de la arquitectura de un sistema de software, identifican los subsistemas, definen sus responsabilidades, y establecen reglas y guías para organizar las relaciones entre ellos.

Modelo-Vista-Controlador

El patrón Modelo-Vista-Controlador es un patrón arquitectónico que separa los datos, las interfaces de usuario y la lógica de control en tres componentes distintos. Cada componente agrupa en módulos toda la aplicación, donde la vista engloba lo referente a las interfaces de usuario, el modelo comprende las clases de acceso a datos, y el controlador toda la lógica de negocio.

Modelo: Es el componente encargado del acceso a datos, representando las estructuras de datos del sistema. Típicamente el modelo de clases contendrá funciones para consultar, insertar y actualizar información de la base de datos.

Vista: Define la interfaz de usuario, es la información presentada al usuario. Una vista puede ser una página web o una parte de una página.

Controlador: Responde a eventos y actúa como intermediario entre el modelo, la vista y cualquier otro recurso necesario para generar una página.

Ventajas de la utilización del patrón MVC:

Facilidad para realizar cambios en la aplicación, puesto que al realizarse un cambio de bases de datos, programación o interfaz de usuario, solo se maneja uno de los componentes. Se puede modificar uno de los componentes sin conocer cómo funcionan los otros.

1.3.10 Métricas.

Desde los albores de la informática se han tratado de medir de una manera u otra, distintos atributos del software, como son: tamaño, complejidad, frecuencia esperada de aparición de errores, cobertura de pruebas, o incluso atributos del proceso de desarrollo de software como la productividad.

El IEEE define las métricas como “medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado”. (Pressman, 2005).

El modelo de métricas tiene por objetivo medir la calidad de los productos intermedios generados en un proyecto de software. Las métricas de calidad proporcionan una indicación de cómo se ajusta el software a los requisitos implícitos y explícitos del cliente. Es decir, cómo se va a medir para que el sistema se adapte a los requisitos que pide el cliente. (Navarro, 2006).

En el mundo de hoy un factor fundamental para efectuar la evaluación de los productos y procesos de software en los diferentes dominios de aplicación lo constituyen las métricas, las cuales abarcan

un gran escenario en cuanto a medidas de software computacional se refieren. Las métricas suelen ser aplicadas a muchas organizaciones, procesos y productos que estén relacionados y afecten a la estimación de costo.

En general, la medición persigue tres objetivos fundamentales:

- Entender qué ocurre durante el desarrollo y el mantenimiento.
- Controlar qué es lo que ocurre en los proyectos.
- Mejorar los procesos y los productos.

Las métricas juegan un papel fundamental en la construcción de productos de software. Estas proporcionan una indicación de la calidad de los mismos en función de sus atributos y características, permitiendo obtener una visión más profunda de los artefactos que se crean durante el ciclo de vida de un proyecto y de esta forma corregirlos para que alcancen un nivel superior de calidad. (Pressman, 2005).

Métricas orientadas a clases

Se utilizan para evaluar la calidad de diseños orientados a objetos, teniendo como base las características de las clases.

Árbol de Profundidad de Herencia (APH).

La métrica Árbol de Profundidad de Herencia fue propuesta por Chidamber y Kemerer (CK). Se define como la longitud máxima desde el nodo hasta la raíz del árbol, el nodo es una clase hija que hereda de una clase, y así respectivamente hasta llegar a la raíz. A medida que esa longitud va creciendo, entonces se van heredando más operaciones y atributos por las clases hijas y la complejidad del módulo aumenta haciéndose más difícil predecir el comportamiento de las clases que ocupan niveles inferiores.

Definición: mide el máximo nivel en la jerarquía de herencia. Es la cuenta directa de los niveles en la jerarquía de herencia. En el nivel cero de la jerarquía se encuentra la clase raíz. (Chidamber y Kemerer, 1994).

Propósito: medida de la complejidad de una clase, esto es debido a que cuanto más profunda se encuentra una clase en la jerarquía, mayor será la probabilidad de heredar un mayor número de métodos. (Chidamber y Kemerer, 1994).

Si los valores de APH son grandes, entonces se garantiza que se reutilice gran cantidad de código; pero al mismo tiempo hace que el diseño sea más complejo, esto provoca un mayor acoplamiento entre las clases.

Tamaño de clase (TC).

La métrica de tamaño de clase pertenece a la familia de métricas propuesta por Lorenz y Kidd. El tamaño de una clase está basado en el número de sus operaciones y atributos. Valores elevados de TC, indican que la clase posee demasiada responsabilidad, por lo que disminuye la reusabilidad de la misma y se dificulta su mantenimiento. (Pressman, 2005).

El tamaño general de una clase se puede determinar teniendo en cuenta:

- El número total de operaciones (tanto operaciones heredadas como operaciones privadas de la instancia) que están encapsuladas dentro de la clase.
- El número de atributos (tanto atributos heredados como atributos privados de la instancia) que están encapsulados en la clase.
- Promedio general de los dos anteriores para el sistema completo.

Si existen valores grandes de TC éstos estarán demostrando que una clase puede tener demasiada responsabilidad, lo cual reduciría la reutilización de la clase y hará complicada la implementación y la prueba. De forma contraria sucede si los valores TC son de menor valor.

Es necesaria una evaluación concreta de las métricas mediante los umbrales:

Clasificación	Valores de los umbrales
Pequeño	<=20
Medio	>20 <=30
Grande	>30

Figura 1.3 Umbrales para la clasificación de las clases.

1.4 Conclusiones

En este capítulo se profundizó en el estudio de las metodologías de desarrollo de software más usadas, se estudiaron las características fundamentales de los lenguajes de programación tanto del lado del cliente como del lado del servidor para el desarrollo de aplicaciones web, y se analizaron herramientas que dan soporte al proceso de desarrollo de software. Después de tener estos elementos se arribó a las siguientes conclusiones:

Se utilizará la metodología de desarrollo RUP por ser una metodología robusta que genera un gran volumen de información e implementa un conjunto de mejoras prácticas recomendadas para desarrollar software exitosamente. Se usará como herramienta CASE Visual Paradigm 6.4 para la especificación y el diseño de la aplicación. Los lenguajes de programación del lado del cliente que se usarán son JavaScript, CSS y HTML, y del lado del servidor PHP 5, como framework de desarrollo se empleará Symfony 1.3 utilizando la librería EXT JS y como servidor web se usará Apache 2.2.6. Además se trabajará con una arquitectura en capas.

Capítulo 2. Análisis del sistema y validación.

2.1 Introducción

En este capítulo se realizará el análisis del Sistema Integral de Documentación e Información Judicial (SIDIJ), generando así los diagramas de clases de análisis y los diagramas de colaboración.

Se elaborará un diagrama de clase de análisis por cada caso de uso y un diagrama de colaboración por cada escenario de estos. Después de la realización de estos diagramas se obtiene como resultado la entrada principal para el Modelo de Diseño, y luego se procede a la validación del modelo de análisis propuesto.

Se debe señalar que en este capítulo se hace referencia a los casos de uso que fueron definidos en la tesis de los autores: Yoanna Ibáñez Fernández y Yebel Fornaris Licea, titulada: "Modelamiento del Negocio e Ingeniería de Requisitos del Sistema Integral de Documentación e Información Judicial (SIDIJ)", debido a que en el análisis se debe conseguir una comprensión más precisa de los requisitos, refinarlos y estructurarlos. La realización del modelo de análisis está basada en los casos de uso del sistema representados en el siguiente diagrama; la descripción de cada caso de uso se encuentra en la tesis a la que se hizo alusión.

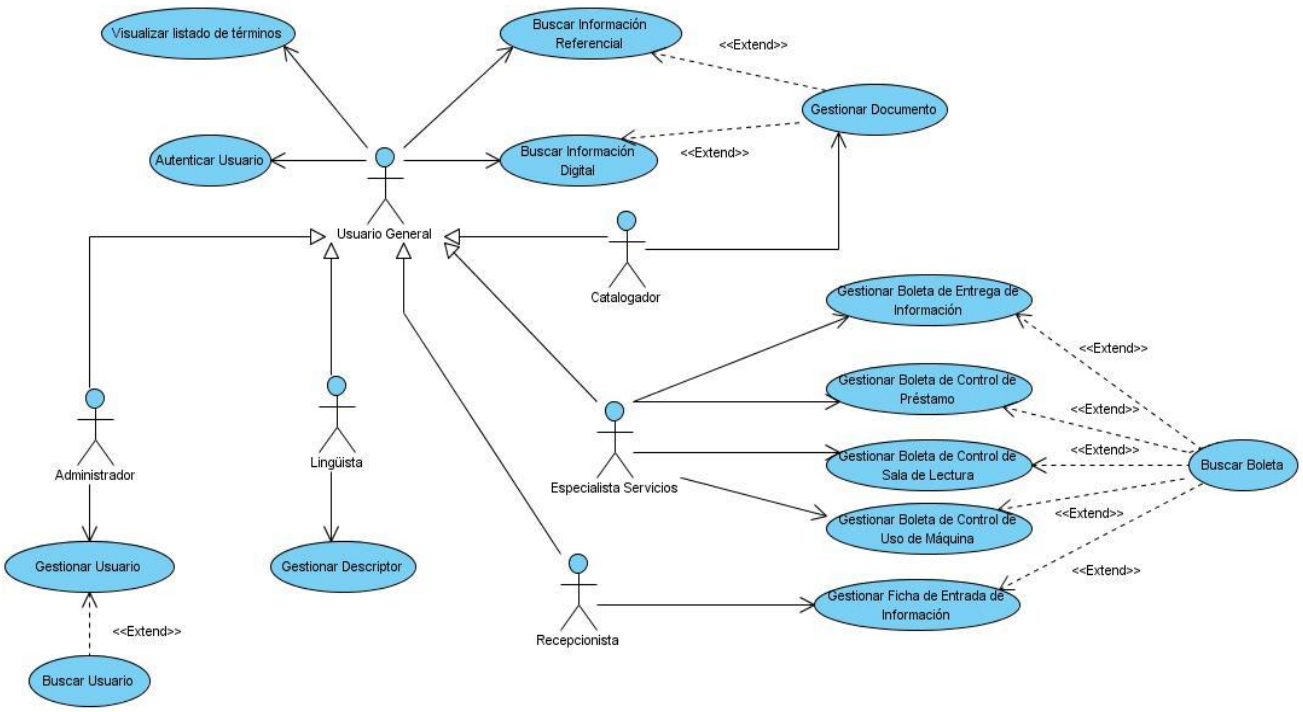


Figura 2.1 Diagrama de Casos de Uso del Sistema.

2.2 Análisis.

El análisis consiste en obtener una visión general del sistema, se preocupa de ver qué hace, de modo que sólo se interesa por los requisitos funcionales.

El análisis orientado a objetos se puede definir como un método de análisis que examina los requisitos desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del dominio del problema (Booch, 1996).

En la construcción del modelo de análisis se identifican las clases que describen la realización de los casos de uso y las relaciones entre ellas.

Las clases que se utilizan para modelar el análisis, se centran en el tratamiento de los requerimientos funcionales y posponen los no funcionales, y se estereotipan de la siguiente manera:

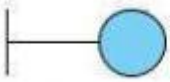
NOMBRE	CARATERÍSTICAS	REPRESENTACIÓN
Entidad	Modelan información que posee larga vida y que es a menudo persistente.	 Clase de entidad
Interfaz	Modelan la interacción entre el sistema y sus actores.	 Clase de interfaz
Control	Coordinan la realización de uno o unos pocos casos de uso coordinando las actividades de los objetos que implementan la funcionalidad del caso de uso.	 Clase de control

Figura 2.2 Estereotipos de las clases del análisis.

2.3 Modelo de clases del análisis.

Como parte del Modelo de Análisis se identifican las clases del análisis y las relaciones que existen entre ellas, basándose en los conceptos significativos del dominio del problema. Con esta información se realizan los diagramas de clases del análisis, en este se refinan los requisitos, pero no se toma en cuenta el lenguaje de programación a usar en la construcción, ni se define la plataforma en la que se ejecutará la aplicación. Sirve como una primera aproximación al diseño constituyendo sus artefactos la entrada principal de este último.

A continuación se muestra los diagramas de las clases del análisis de algunas de las funcionalidades significativas del sistema, los restantes pueden encontrase en los anexos.

Figura 2.3 CU_ Autenticar_Usuario.

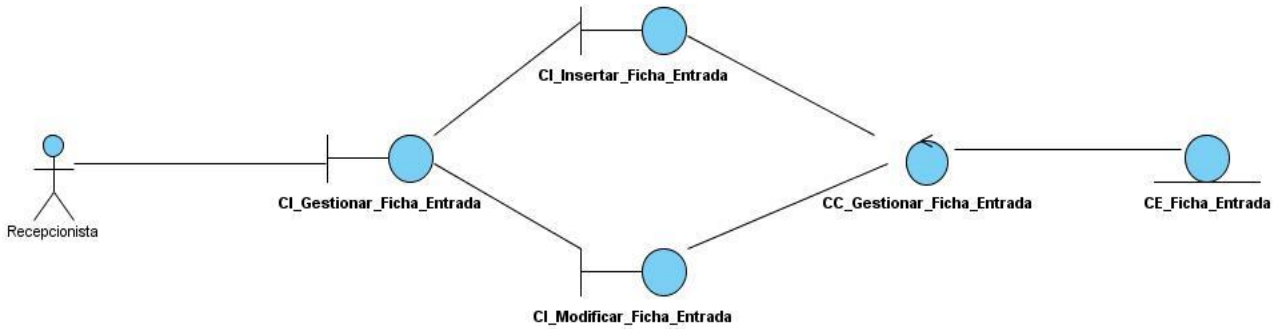


Figura 2.4 CU_ Gestionar_Ficha_Entrada.

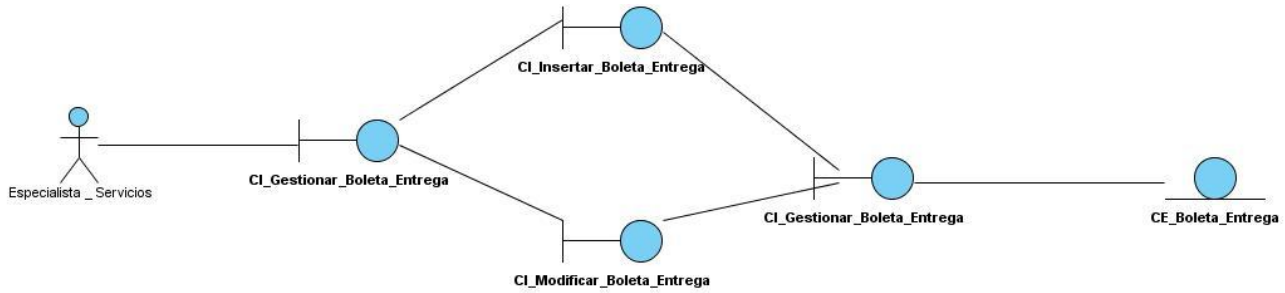


Figura 2.5 CU_ Gestionar_Boleta_Entrega.

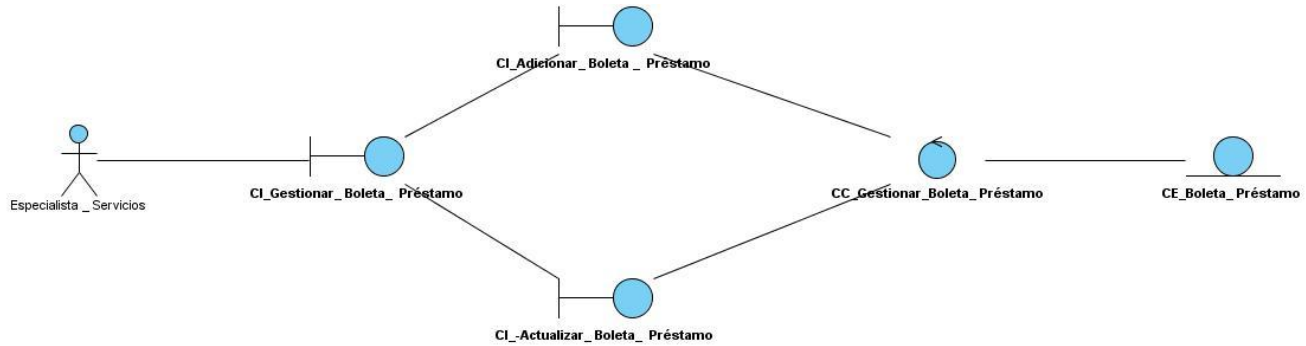


Figura 2.6 CU_ Gestionar_Boleta_Préstamo.

2.4 Diagramas de interacción.

Los diagramas de interacción se utilizan para modelar los aspectos dinámicos de un sistema, lo que conlleva a modelar instancias concretas o prototípicas de clases interfaces, componentes y nodos, junto con los mensajes enviados entre ellos, todo en el contexto de un escenario que ilustra un comportamiento.

UML define dos tipos de diagramas de interacción, los cuales expresan interacciones semejantes o idénticas del mensaje:

Diagrama de secuencia: Muestra las interacciones expresadas en función de secuencias temporales destacando la ordenación temporal de los mensajes.

Diagrama de colaboración: Muestra las relaciones entre los objetos y los mensajes que intercambian. Es un diagrama que destaca la organización estructural de los objetos que envían y reciben mensajes.

Es recomendable utilizar los diagramas de colaboración en el análisis y los de secuencia para el diseño.

2.4.1 Diagramas de colaboración.

Los diagramas de colaboración, además de destacar la organización estructural de los objetos que envían y reciben mensajes, son usados para modelar la dinámica del sistema a través de los objetos de las diferentes clases del análisis, sus relaciones y los mensajes que se puedan enviar entre ellos.

A continuación se muestran diagramas de colaboración del análisis de algunas de las funcionalidades significativas del sistema, los restantes pueden encontrase en los anexos.

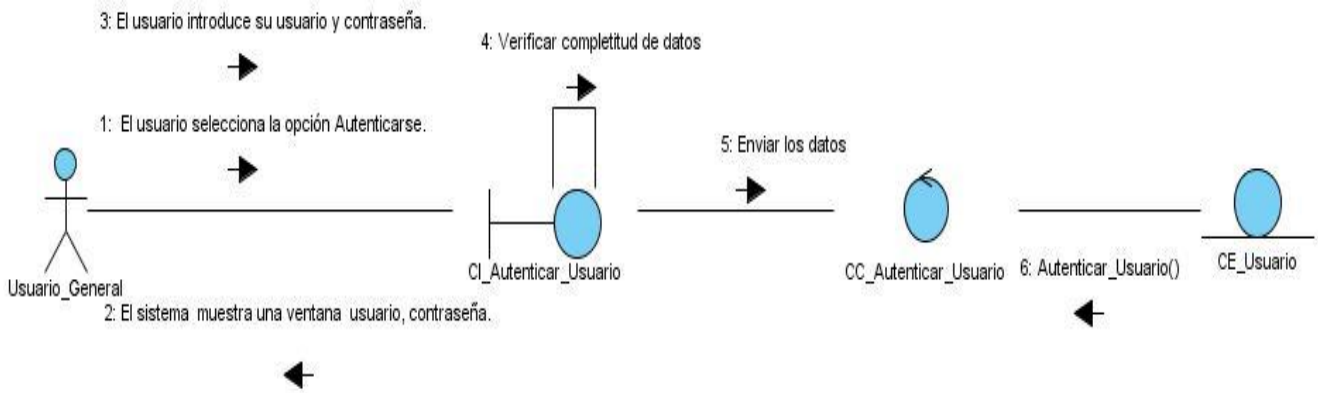


Figura 2.7 CU_ Autenticar_Usuario.

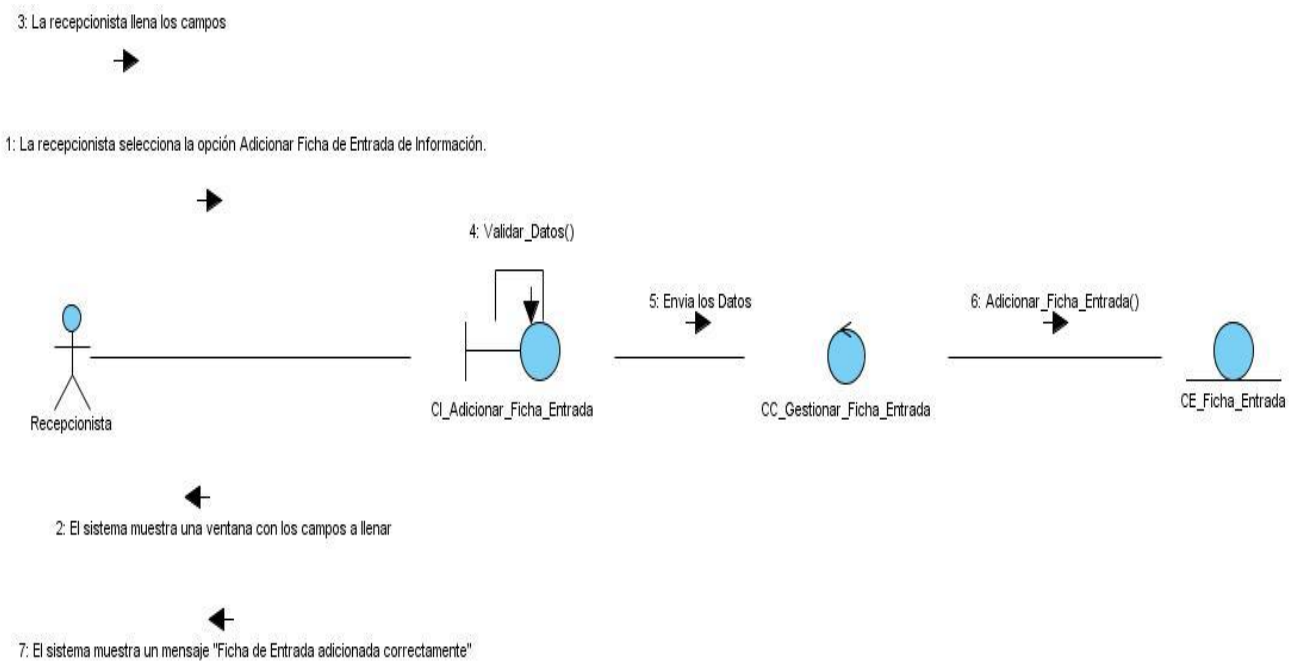


Figura 2.8 CU_ Gestionar_Ficha_Entrada Escenario: Adicionar.

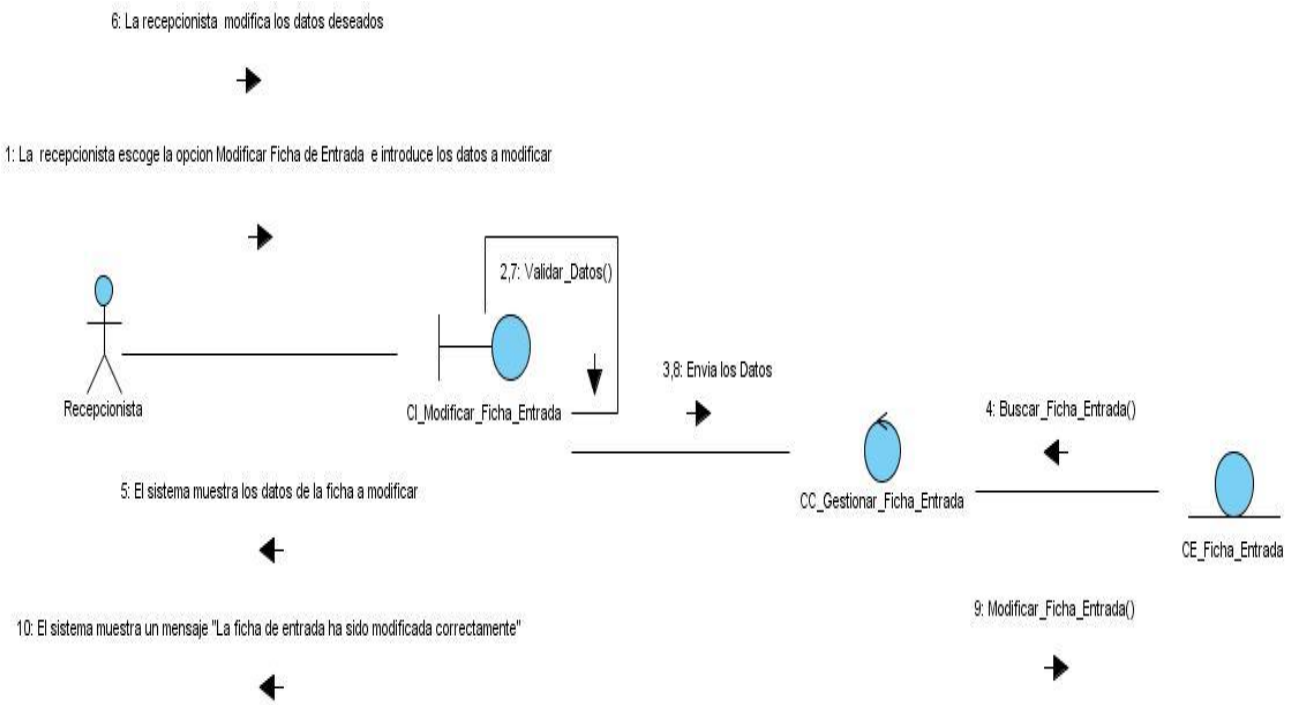


Figura 2.9 CU_ Gestionar_Ficha_Entrada Escenario: Modificar.

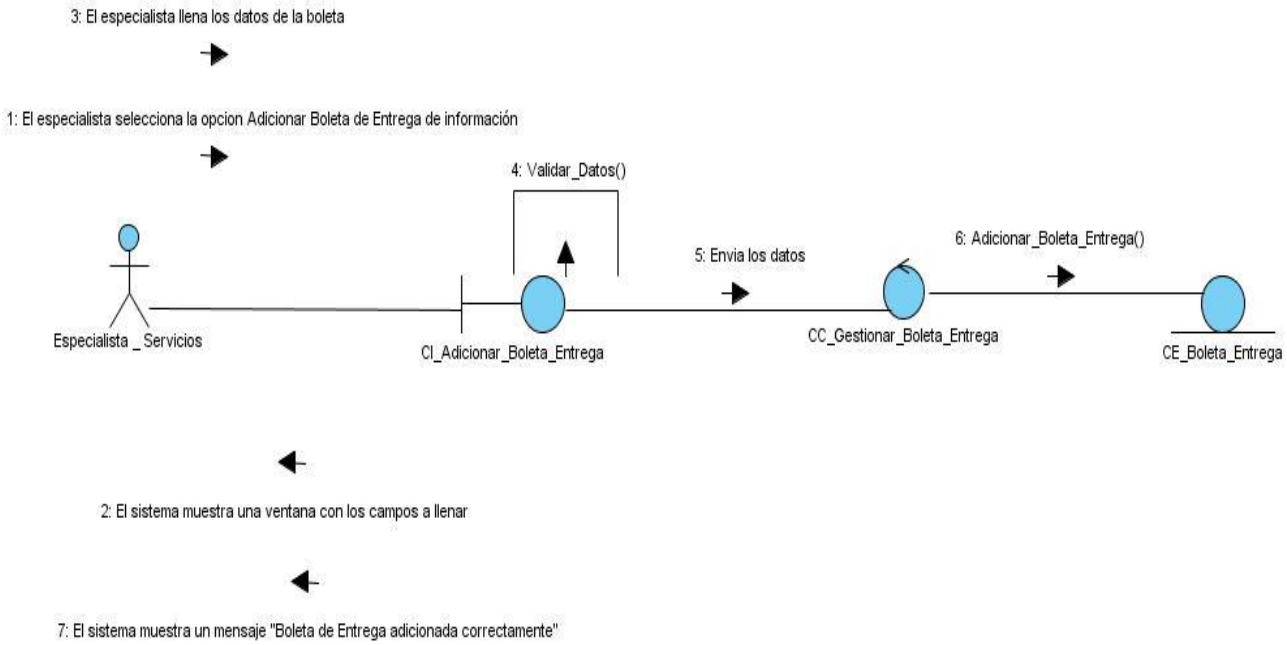


Figura 2.10 CU_ Gestionar_Boleta_Entrega Escenario: Adicionar.

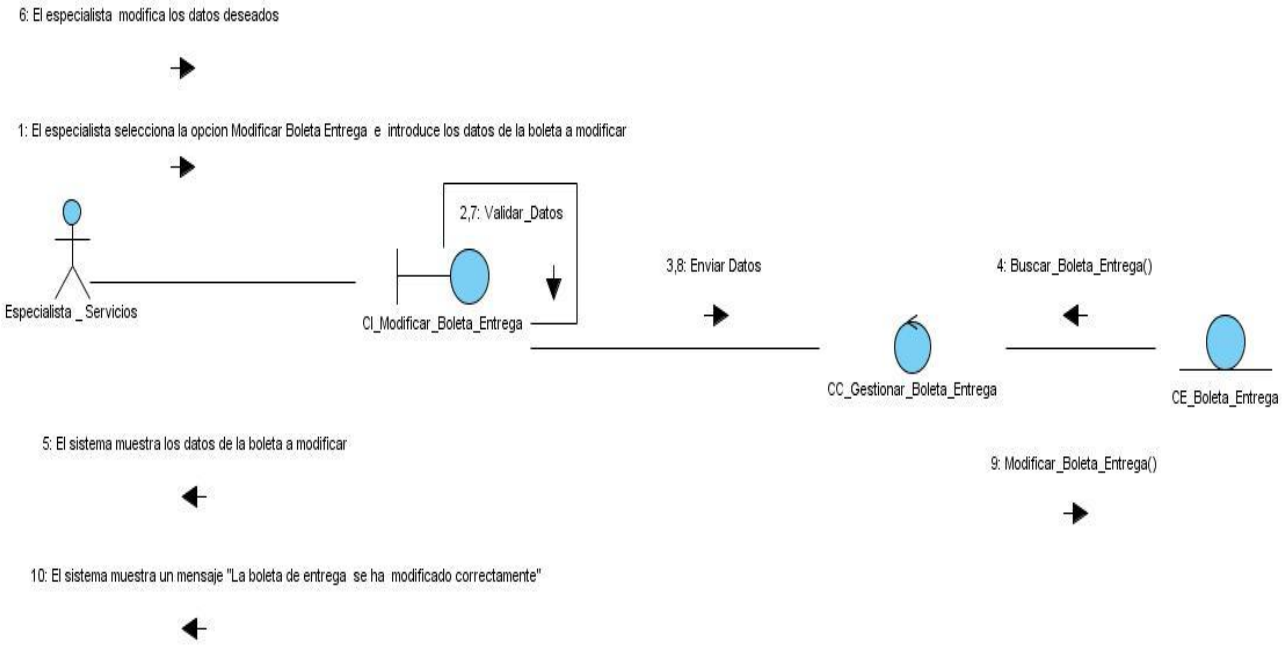


Figura 2.11 CU_ Gestionar_Boleta_Entrega Escenario: Modificar.

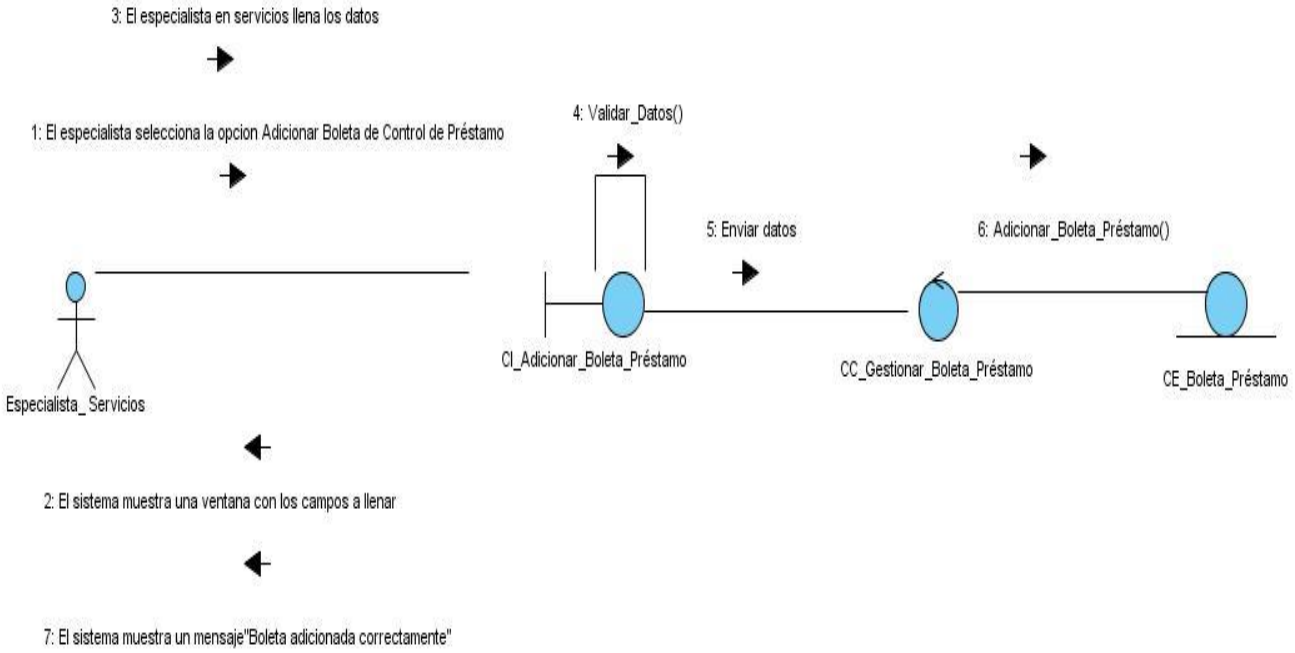


Figura 2.12 CU_ Gestionar_Boleta_Préstamo Escenario: Adicionar.

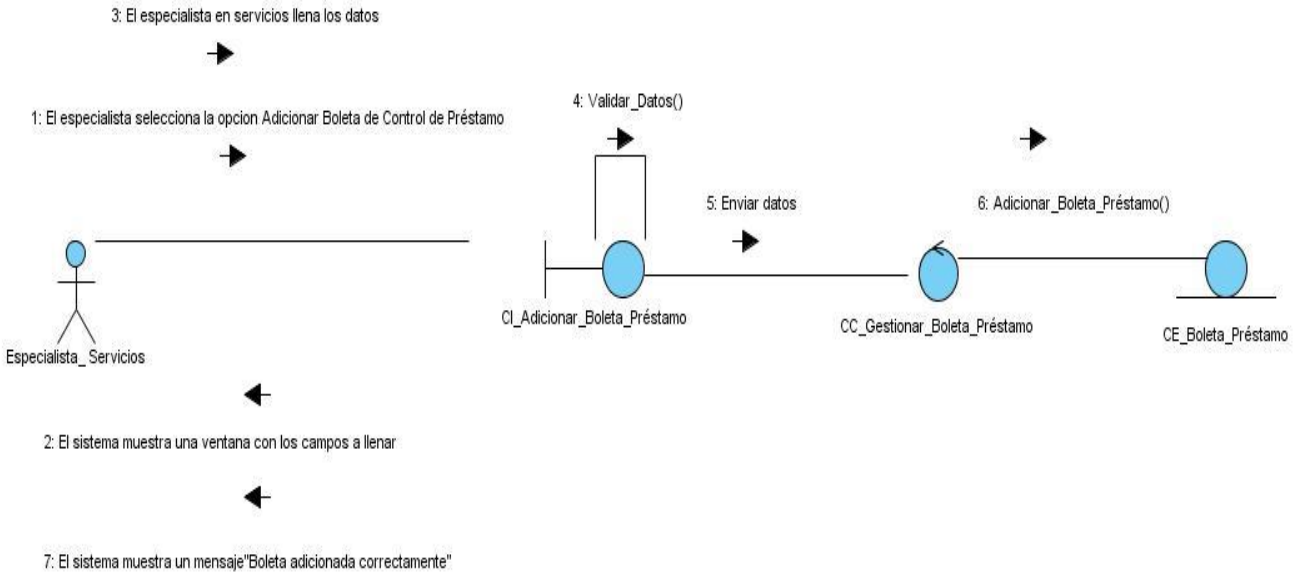


Figura 2.13 CU_ Gestionar_Boleta_Préstamo Escenario: Actualizar.

2.5 Validación del análisis.

Al culminar el modelo de análisis quedaron definidos 15 diagramas de clases del análisis, y 72 diagramas de colaboración, estos últimos se realizan para cada escenario de los casos de uso.

El modelo de análisis realizado es fiable y permite dar paso al diseño de la aplicación teniendo en cuenta que cumple con:

Por diagramas:

- Cada diagrama muestra el flujo normal de los eventos descritos para los casos de uso.
- Se realizaron diagramas de colaboración para representar la interacción entre las clases a través de mensajes, indicando que necesita cada clase para responder una petición y que responde ante esta.

Por clases:

- Las clases tienen las relaciones necesarias para propagar los mensajes y cumplir con la tarea que indica el caso de uso al que representan.
- Las clases involucradas en cada diagrama contienen la información necesaria para la realización de la acción que indica el caso de uso correspondiente.

Por mensajes:

- Los mensajes obligan a que se cumplan todos los pasos necesarios para la realización de las acciones, incluyendo la validación de los datos.
- Todo mensaje está definido en su correspondiente clase.
- La secuencia de los mensajes se encuentra en correspondencia con el flujo de eventos que debe realizarse para dar cumplimiento a la funcionalidad del sistema representada.

Por entidades:

- Se tiene en cuenta en cada diagrama los objetos o entidades que se encuentran involucrados en la acción que indica el caso de uso.

Por actores:

- Los actores involucrados en cada diagrama son los encargados de inicializar la acción que indica el caso de uso que se representa.

Matrices de trazabilidad.

Se realizaron matrices de trazabilidad para determinar las relaciones entre los casos de uso y los diagramas de clases del análisis; y entre ellos y los diagramas de colaboración del análisis, facilitando el seguimiento de los casos de uso; y permitiendo reconocer que diagrama es afectado por cada uno de ellos; así cuando un caso de uso sufre algún cambio se conoce rápidamente cuáles diagramas deben ser modificados.

Casos de uso:

1. Autenticar Usuario.
2. Gestionar Usuario.
3. Buscar Usuario.
4. Gestionar Descriptor.
5. Visualizar Listado Descriptores.
6. Gestionar Ficha Entrada.
7. Gestionar Boleta Entrega.
8. Gestionar Boleta Préstamo.
9. Gestionar Boleta Control Sala Lectura.
10. Gestionar Boleta de Control de Uso de Máquina.
11. Buscar Boleta (Recepcionista).
12. Buscar Boleta (Especialista).
13. Gestionar Documento.
14. Buscar Información Digital.
15. Buscar Información Referencial.

El caso de uso Buscar Boleta fue necesario dividirlo en dos casos de uso para su representación en el análisis: Buscar Boleta (Recepcionista) y Buscar Boleta (Especialista) debido a que lo inicializan diferentes actores y se manipulan entidades distintas.

	Casos de Uso													
Diagramas de clases análisis	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
DC_Autenticar_Usuario	x														
DC_Gestionar_Usuario --- Escenario: Adicionar		x													
DC_Gestionar_Usuario --- Escenario: Eliminar		x													
DC_Gestionar_Usuario --- Escenario: Modificar		x													
DC_Buscar_Usuario			x												
DC_Gestionar Descriptor --- Escenario: Adicionar				x											
DC_Gestionar Descriptor --- Escenario: Eliminar				x											
DC_Gestionar Descriptor --- Escenario: Modificar				x											
DC_Visualizar_Listado_Descriptores					x										
DC_Gestionar_Ficha_Entrada ---Escenario: Adicionar						x									
DC_Gestionar_Ficha_Entrada ---Escenario: Modificar						x									
DC_Gestionar_ Boleta _ Entrega ---Escenario: Adicionar							x								
DC_Gestionar_ Boleta _ Entrega ---Escenario: Modificar							x								
DC_Gestionar_ Boleta _ Préstamo ---Escenario: Adicionar								x							
DC_Gestionar_ Boleta _ Préstamo ---Escenario: Actualizar								x							
DC_Gestionar_ Boleta_Control_ Sala_ Lectura ---Escenario: Adicionar									x						
DC_Gestionar_ Boleta_Control_ Sala_ Lectura ---Escenario: Modificar									x						
DC_Gestionar_Boleta_Control_Uso_Maquina ---Escenario: Adicionar										x					
DC_Gestionar_Boleta_Control_Uso_Maquina ---Escenario: Modificar										x					
DC_Buscar_Boleta(Ficha)											x				
DC_Buscar_Boleta(Entrega)												x			
DC_Buscar_Boleta(Prestamo)												x			
DC_Buscar_Boleta(Sala_Lectura)												x			
DC_Buscar_Boleta(Uso_Maquina)												x			
DC_Gestionar_Documento(CD) --- Escenario: Insertar													x		
DC_Gestionar_Documento(CD) --- Escenario: Modificar													x		
DC_Gestionar_Documento(CD) --- Escenario: Eliminar													x		
DC_Gestionar_Documento(Investigación) --- Escenario: Insertar													x		
DC_Gestionar_Documento(Investigación) --- Escenario: Modificar													x		
DC_Gestionar_Documento(Investigación) --- Escenario: Eliminar													x		
DC_Gestionar_Documento(Doctrina) --- Escenario: Insertar													x		
DC_Gestionar_Documento(Doctrina) --- Escenario: Modificar													x		
DC_Gestionar_Documento(Doctrina) --- Escenario: Eliminar													x		
DC_Gestionar_Documento(DispCon_Gob_Digital) --- Escenario: Insertar													x		
DC_Gestionar_Documento(DispCon_Gob_Digital) --- Escenario: Modificar													x		
DC_Gestionar_Documento(DispCon_Gob_Digital) --- Escenario: Eliminar													x		
DC_Gestionar_Documento(DispCon_Gob_Referencial) --- Escenario: Insertar													x		
DC_Gestionar_Documento(DispCon_Gob_Referencial) --- Escenario: Modificar													x		
DC_Gestionar_Documento(DispCon_Gob_Referencial) --- Escenario: Eliminar													x		
DC_Gestionar_Documento(Sentencia) --- Escenario: Insertar													x		
DC_Gestionar_Documento(Sentencia) --- Escenario: Modificar													x		
DC_Gestionar_Documento(Sentencia) --- Escenario: Eliminar													x		
DC_Gestionar_Documento(Legislación) --- Escenario: Insertar													x		
DC_Gestionar_Documento(Legislación) --- Escenario: Modificar													x		
DC_Gestionar_Documento(Legislación) --- Escenario: Eliminar													x		
DC_Gestionar_Documento(Publicación_Seriada) --- Escenario: Insertar													x		
DC_Gestionar_Documento(Publicación_Seriada) --- Escenario: Modificar													x		
DC_Gestionar_Documento(Publicación_Seriada) --- Escenario: Eliminar													x		
DC_Gestionar_Documento(Flujo Ascendente) --- Escenario: Insertar													x		
DC_Gestionar_Documento(Flujo Ascendente) --- Escenario: Modificar													x		
DC_Gestionar_Documento(Flujo Ascendente) --- Escenario: Eliminar													x		
DC_Gestionar_Documento(Referencia_Fondo_General) --- Escenario: Insertar													x		
DC_Gestionar_Documento(Referencia_Fondo_General) --- Escenario: Modificar													x		
DC_Gestionar_Documento(Referencia_Fondo_General) --- Escenario: Eliminar													x		
DC_Gestionar_Documento(Referencia_Libro_Folleto) --- Escenario: Insertar													x		
DC_Gestionar_Documento(Referencia_Libro_Folleto) --- Escenario: Modificar													x		
DC_Gestionar_Documento(Referencia_Libro_Folleto) --- Escenario: Eliminar													x		
DC_Gestionar_Documento(Referencia_Referencia) --- Escenario: Insertar													x		
DC_Gestionar_Documento(Referencia_Referencia) --- Escenario: Modificar													x		
DC_Gestionar_Documento(Referencia_Referencia) --- Escenario: Eliminar													x		
DC_Buscar_Información_Digital(Legislación)														x	
DC_Buscar_Información_Digital(DisCGob)														x	
DC_Buscar_Información_Digital(Doctrina)														x	
DC_Buscar_Información_Digital(Investigacion)														x	
DC_Buscar_Información_Digital(Sentencia)														x	
DC_Buscar_ Información _Referencial(DisCGob)															x
DC_Buscar_ Información _Referencial(Flujo_Ascendente)															x
DC_Buscar_ Información _Referencial(Fondo_General)															x
DC_Buscar_ Información _Referencial(Libro_y_Folleto)															x
DC_Buscar_ Información _Referencial(Publicación_Seriada)															x
DC_Buscar_ Información _Referencial(Referencia)															x
DC_Buscar_ Información _Referencial(CD)															x

Figura 2.14 Matriz de Trazabilidad de Casos de Uso vs. Diagramas de Colaboración del análisis.

	Casos de Uso														
Diagramas de clases del análisis	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CU_Autenticar_Usuario	x														
CU_Gestionar_Usuario		x													
CU_Buscar_Usuario			x												
CU_Gestionar Descriptor				x											
CU_Visualizar_Listado_Descriptores					x										
CU_Gestionar_Ficha_Entrada						x									
CU_Gestionar_ Boleta _ Entrega							x								
CU_Gestionar_ Boleta _ Préstamo								x							
CU_Gestionar_ Boleta_Control_ Sala_ Lectura									x						
CU_Gestionar_Boleta_Control_Uso_Maquina										x					
CU_Buscar_Boleta(Recepcionista)											x				
CU_Buscar_Boleta(Especialista)												x			
CU_Gestionar_Documento													x		
CU_Buscar_Información_Digital														x	
CU_Buscar_ Información _Referencial															x

Figura 2.15 Matriz de Trazabilidad de Casos de uso vs. Diagramas de Clases del Análisis.

Leyenda:

DC: Diagrama de colaboración.

CU: Caso de uso.

DiscGob: Disposición del Consejo de Gobierno.

2.6 Conclusiones

Al culminar la modelación de las clases del análisis quedan definidos los diagramas de clases y los diagramas de colaboración.

- Se logró describir lo que requiere el cliente, y establecer una base para la creación de un diseño de software.
- La validación del análisis permite garantizar la fiabilidad y calidad de los artefactos generados durante el análisis.
- Con el desarrollo de los diferentes artefactos se ha logrado un modelo de análisis preparado para el diseño del software.

.

Capítulo 3. Diseño del sistema y validación.

3.1 Introducción

En este capítulo se realizará el diseño del sistema SIDIJ donde se elaborarán los diagramas de clases e interacción del diseño haciendo uso de UML como lenguaje de modelado.

Teniendo en cuenta que se diseñará sobre la base de Symfony como framework de trabajo, se hará necesario centrar las decisiones que se tomen y los artefactos que se generen en la arquitectura que este propone y en sus características principales.

3.2 Diseño.

El diseño es un refinamiento del análisis que tiene en cuenta los requisitos no funcionales, este debe ser suficiente para que el sistema pueda ser implementado sin ambigüedades. El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en como los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. En este modelo, los casos de uso son realizados por las clases del diseño y sus objetos.

A través del flujo de trabajo de Diseño, RUP permite utilizar las ventajas de la Programación Orientada a Objetos (POO). La POO es un paradigma de la programación que define los programas en términos de clases de objetos, estos objetos son entidades que combinan estado (datos), comportamiento (procedimientos o métodos) e identidad (propiedades de un objeto que lo diferencia de los otros). La POO expresa un programa como un conjunto de estos objetos que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

3.3 Symfony como framework.

Symfony es un framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web; separando la lógica de negocio, del servidor y de la presentación de la aplicación.

- El Modelo representa la información con la que trabaja la aplicación, 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.

Cada nivel está compuesto en varias partes:

- La capa del Modelo
  - ✓ Abstracción de la base de datos
  - ✓ Acceso a los datos
- La capa de la Vista
  - ✓ Vista
  - ✓ Plantilla
  - ✓ Layout
- La capa del Controlador
  - ✓ Controlador frontal
  - ✓ Acción

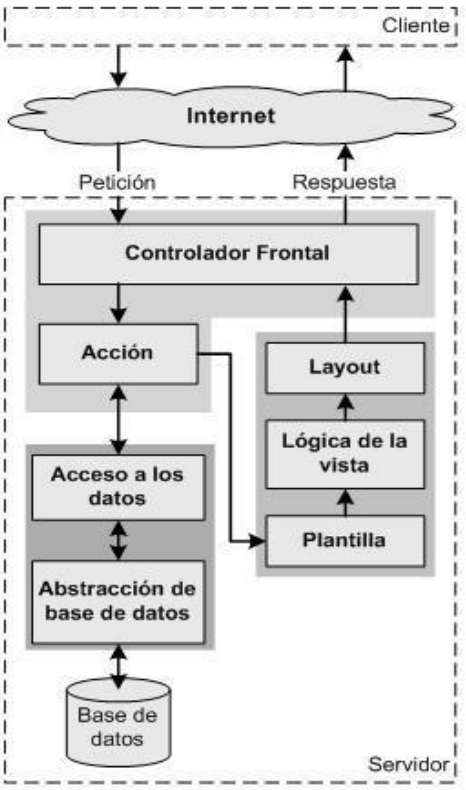


Figura 3.1 Modelo Vista Controlador en Symfony.

El controlador contiene el código que liga la lógica de negocio con la presentación, está compuesto por el controlador frontal que es el único punto de entrada a la aplicación y las acciones que se encargan de verificar la validez de las peticiones y preparar los datos para la vista. Esta última está formada por el layout que posee el código común a todas las acciones, las plantillas que contienen el código asociado a cada acción en particular y la lógica que se puede manipular a través de un fichero de configuración sencillo sin necesidad de programarla. Por último, el modelo está formado por la capa de acceso a datos y por la capa de abstracción de la base de datos que es completamente transparente al programador.

3.4 Subsistemas de diseño.

Los subsistemas de diseño constituyen una forma de estructurar los artefactos que conforman el modelo de diseño en estructuras más independientes. Los elementos del subsistema deben tener un alto grado de cohesión. El subsistema en su conjunto debe tener bajo acoplamiento con respecto a otros subsistemas, lo cual facilita su reutilización.

Con el fin de separar los aspectos del diseño y lograr una mayor independencia en cuanto a desarrollo y a reutilización de funcionalidades, se identificaron los subsistemas relacionados con el sistema SIDIJ. En el subsistema Common se encuentran las clases del diseño que soportan las funcionalidades que puede realizar cualquier usuario, y en el subsistema administrativo todas las que son desarrolladas por usuarios administrativos del sistema, los cuales también pueden utilizar las funcionalidades de las clases del subsistema Common.

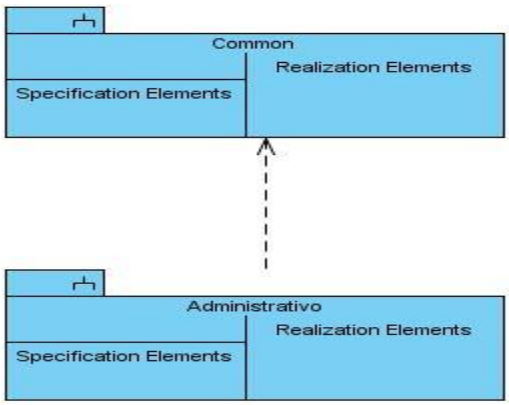


Figura 3.2 Subsistemas de diseño.

3.5 Paquetes de diseño.

El diseño de paquetes facilita la agrupación de clases, relaciones, realizaciones de casos de uso, diagramas y otros paquetes relacionados de alguna manera. Constituye una división del subsistema en partes más manejables. Para lograr una mejor organización y especialización de los elementos del diseño se identificaron y definieron los siguientes paquetes del diseño, siguiendo el patrón arquitectónico MVC, que es además el que utiliza el framework Symfony.

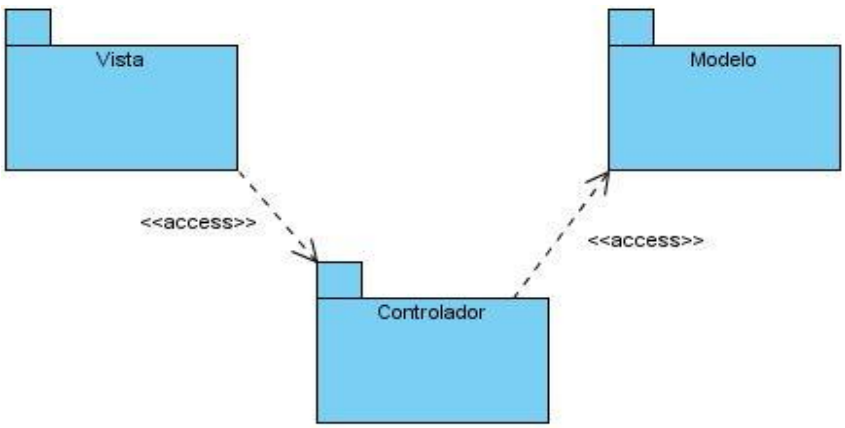


Figura 3.3 Paquetes de diseño.

Posteriormente se podrá observar que en cada diagrama de clase del diseño se encuentra representada cada clase dentro del paquete al que pertenece.

3.6 Patrones aplicados al diseño del SIDIJ.

Generalmente para la elaboración del diseño, se utilizan un grupo de patrones o modelos para lograr objetivos específicos, dentro de los que se encuentran los GRASP, GoF, y los Arquitectónicos.

Del grupo de patrones GRASP se utilizaron Controlador, Bajo Acoplamiento, Experto, este último para que cada clase experta en algún tipo de información fuese la encargada de realizar las operaciones con la misma, logrando con eso un mejor funcionamiento del sistema.

Teniendo en cuenta que se utilizó Symfony como framework de trabajo, se aprovecharon las ventajas que este brinda con el objetivo de utilizar los patrones que soporta por defecto; como es el caso de:

En la categoría Creacionales:

- 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. En el controlador frontal hay una llamada a sfContext: getInstance (). El método getContext (), un objeto muy útil que guarda una referencia a todos los objetos del núcleo de Symfony.

- Abstract Factory (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las estas 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.

En la categoría Estructurales:

- Decorator (Envoltorio): Este patrón añade de manera dinámica funcionalidad a una clase. 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, o si se mira desde el otro punto de vista, el layout decora la plantilla. Todas las páginas mostradas a los usuarios se conforman utilizando el patrón antes descrito. Como se muestra en la figura:

Figura 3.4 Patrón decorador.

3.7 Diagramas de clases del diseño.

Para modelar la vista de diseño estático del sistema SIDIJ, se confeccionaron los diagramas de clases del diseño. Los mismos muestran las relaciones entre clases, interfaces, así como la colaboración entre ellas. Los diagramas de clases, son importantes, no sólo para visualizar, especificar y documentar modelos estructurales, sino también para construir sistemas ejecutables, aplicando ingeniería directa e inversa.

Para la realización de los diagramas de clases de diseño fue necesario el uso de estereotipos web ya que el sistema SIDIJ por sus fines se definió que debía ser una aplicación web. Dichos estereotipos se muestran a continuación:

 Server Page (página servidora): representa la página Web que tiene código, que se ejecuta en el servidor.

 Client Page (página cliente): una instancia de Página Cliente es una página Web, con formato HTML.

 Form (formulario): colección de elementos de entrada que son parte de una página cliente.

Figura 3.5 Estereotipos web.

Se utiliza la extensión de UML para el modelado de aplicaciones Web. Esta extensión presenta como elementos más significativos a tres clases estereotipadas “Server Page”, “Client Page” y “Form” empleadas para el código servidor, código cliente y formularios respectivamente. Y las relaciones entre ellas pueden ser:

<<Build>>: representa una asociación especial que relaciona las páginas cliente con las páginas servidor.

<<Link>>: expresa las asociaciones más comunes entre las páginas, en este caso la del hipervínculo.

<<Submit>>: es la relación que se crea siempre entre una página servidor y un formulario.

A continuación se muestran los diagramas de clases del diseño de algunas de las funcionalidades significativas del sistema, los restantes pueden encontrarse en los anexos.

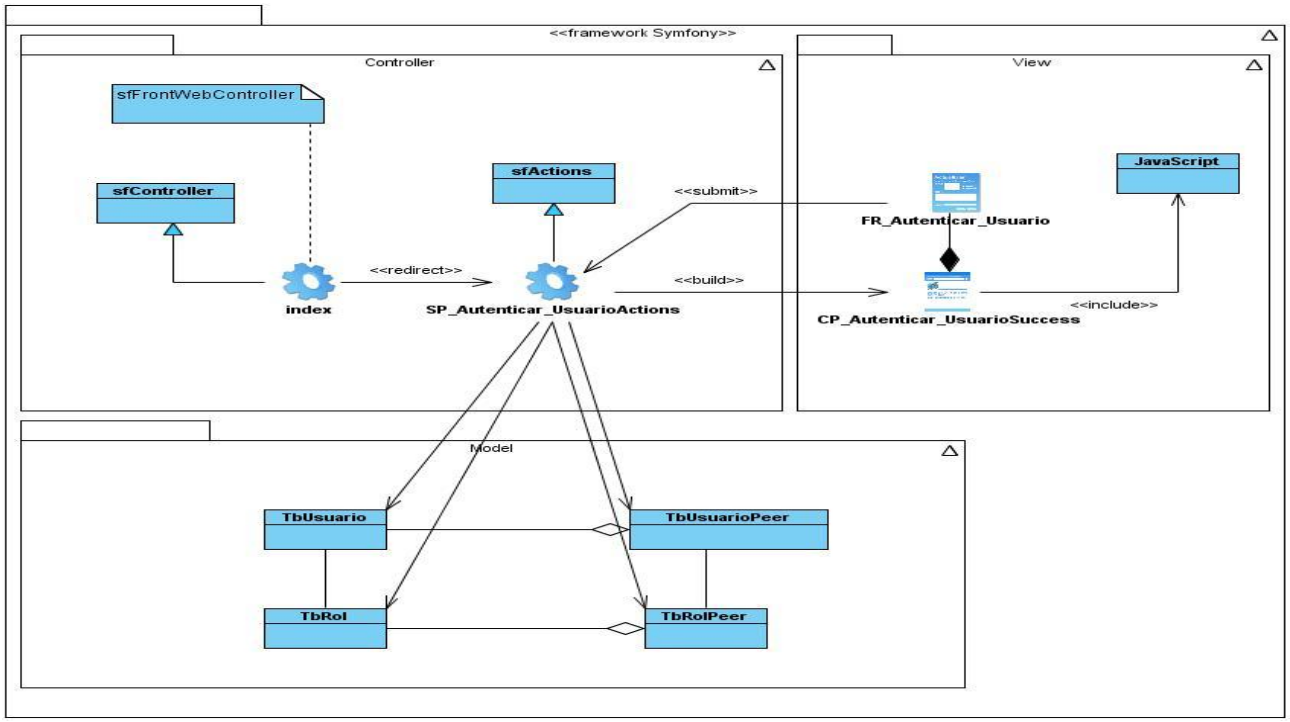


Figura 3.6 CU_Autenticar_Usuario.

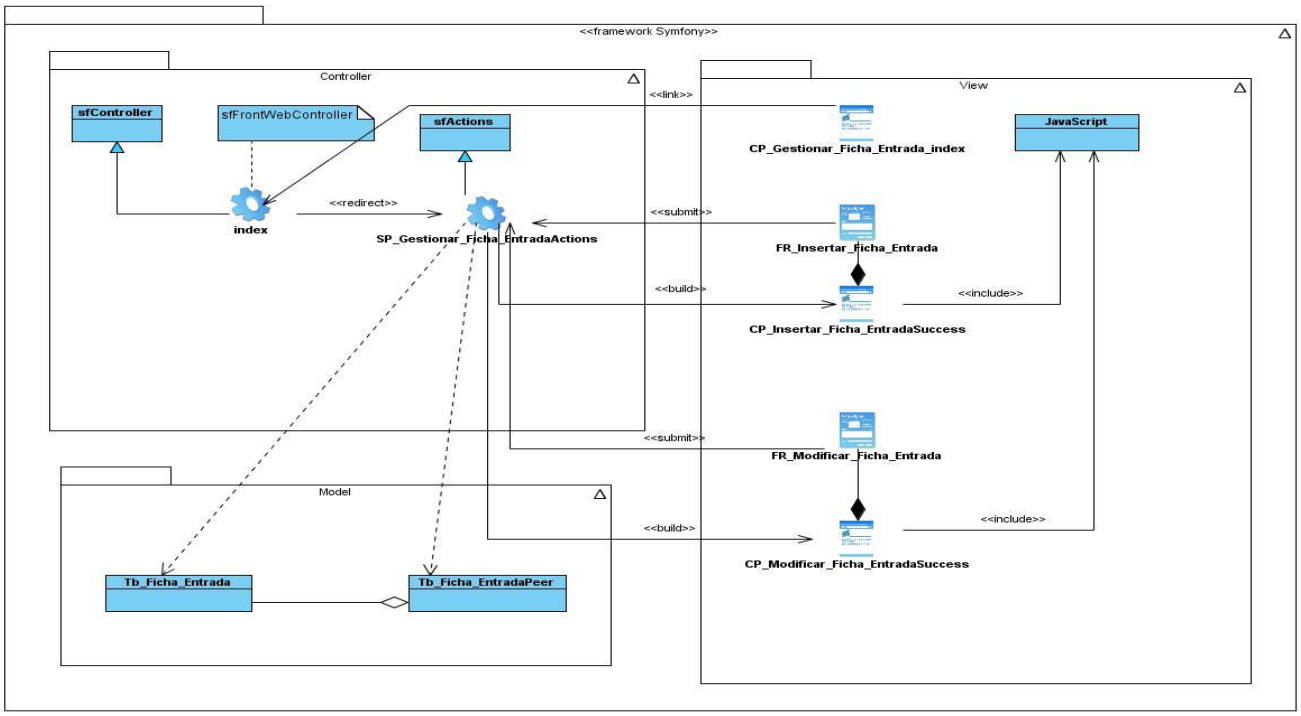


Figura 3.7 CU_Gestionar_Ficha_Entrada.

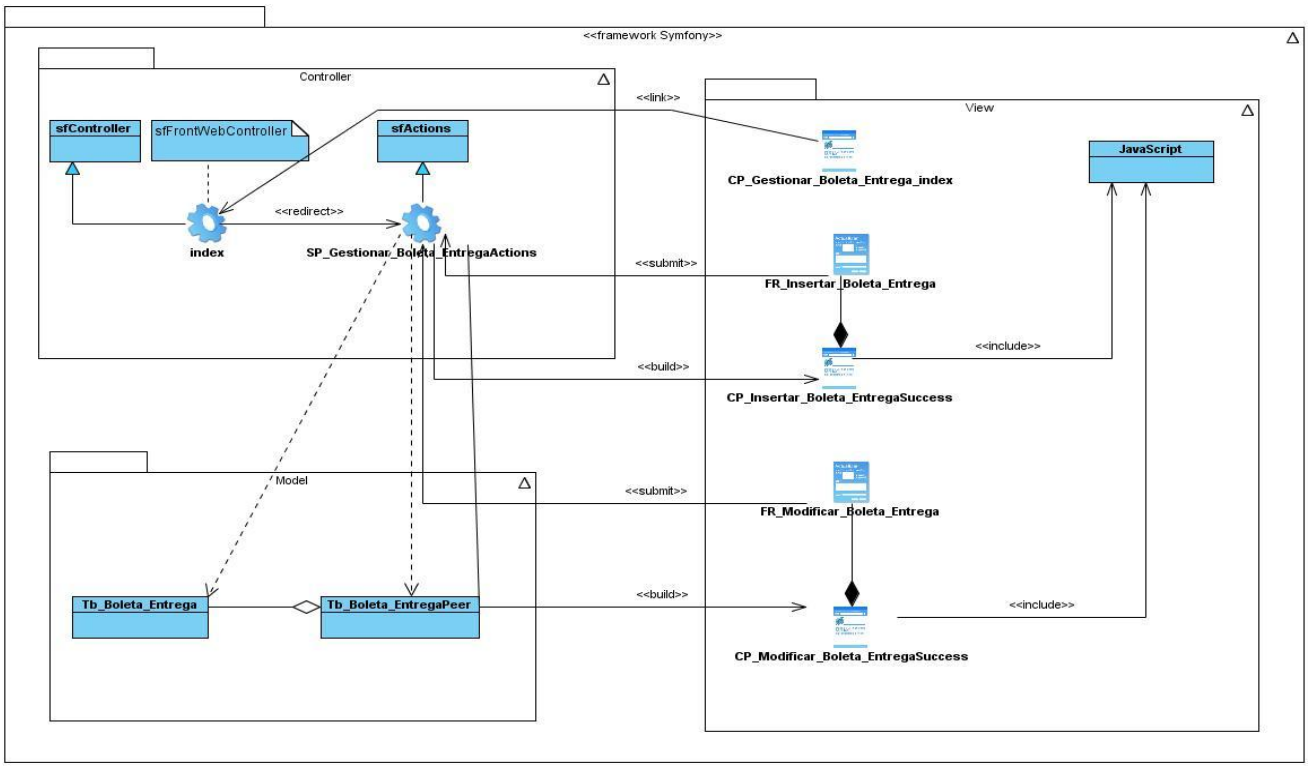


Figura 3.8 CU_Gestionar_Boleta_Entrega.

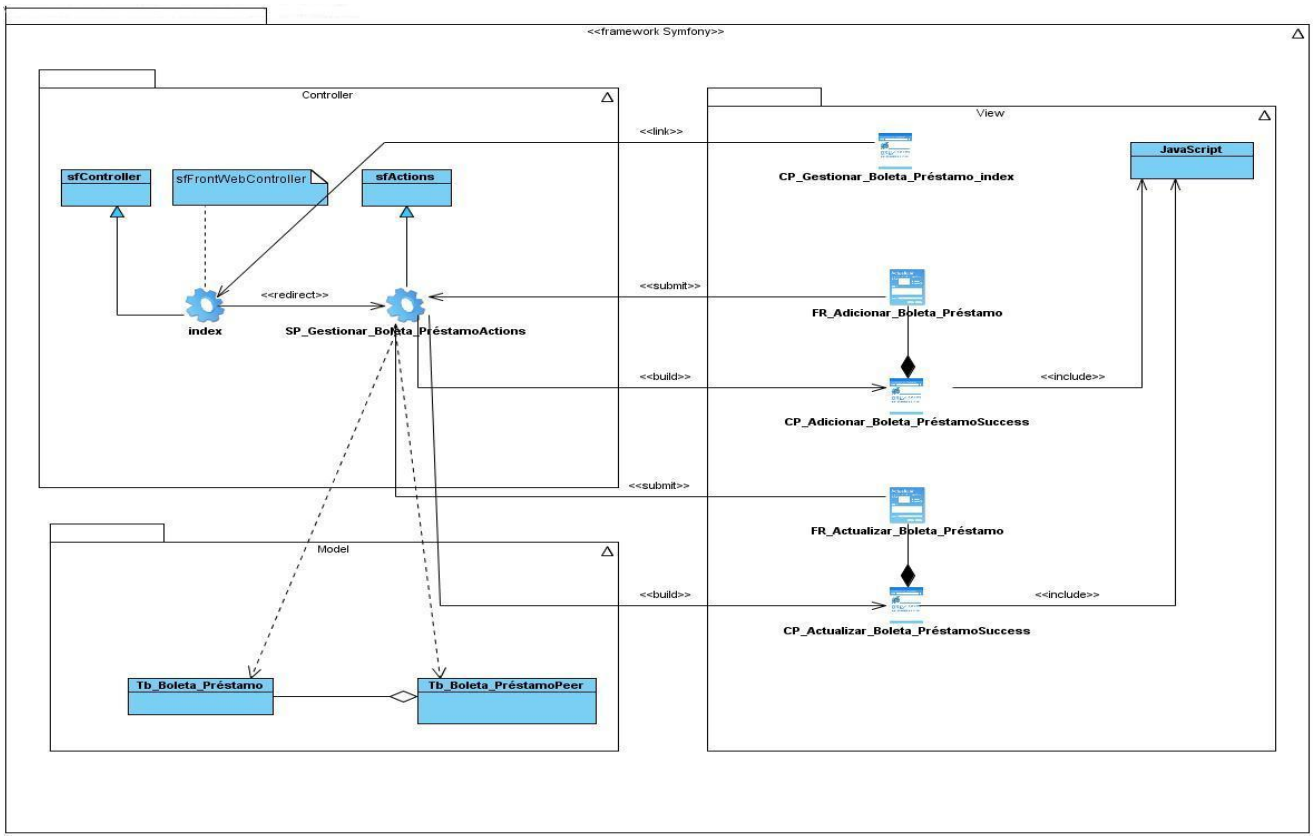


Figura 3.9 CU_Gestionar_Boleta_Préstamo.

3.8 Diagramas de interacción.

Modelan los aspectos dinámicos de un sistema, muestran gráficamente cómo los objetos se comunican entre sí a fin de cumplir con los requerimientos, o sea, describen la interacción entre los objetos que interactúan a través de mensajes para cumplir ciertas tareas. Dentro de los diagramas de interacción se definen los diagramas de secuencia.

3.8.1 Diagramas de secuencia.

Son diagramas de interacción que destacan la ordenación temporal de los mensajes. Muestran los objetos que participan en la interacción mediante sus líneas de vida y los mensajes que intercambian. Este tipo de diagrama se muestra la línea de vida y el foco de control que representa el período de tiempo durante el cual un objeto ejecuta una acción.

A continuación se muestran los diagramas de secuencias de algunas de las funcionalidades significativas del sistema, los restantes pueden encontrarse en los anexos.

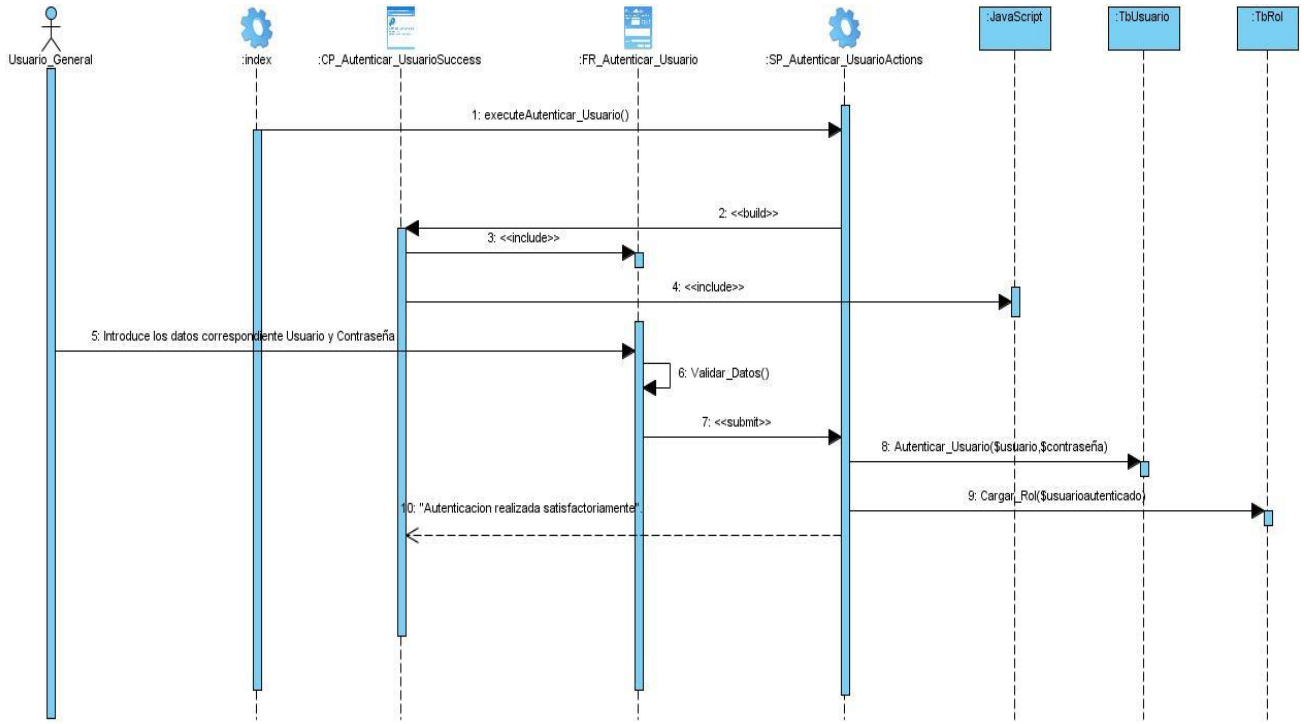


Figura 3.10 CU_Autenticar_Usuario.

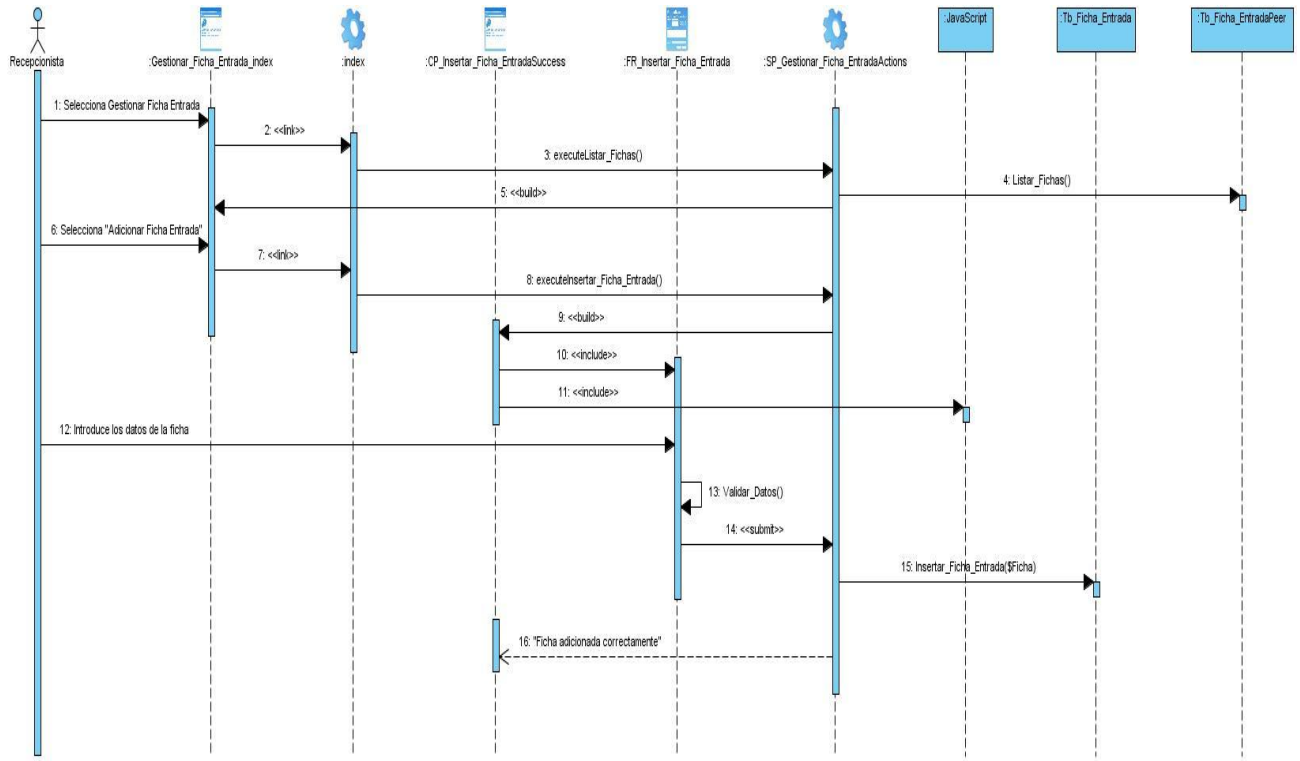


Figura 3.11 CU_Gestionar_Ficha_Entrada Escenario: Insertar.

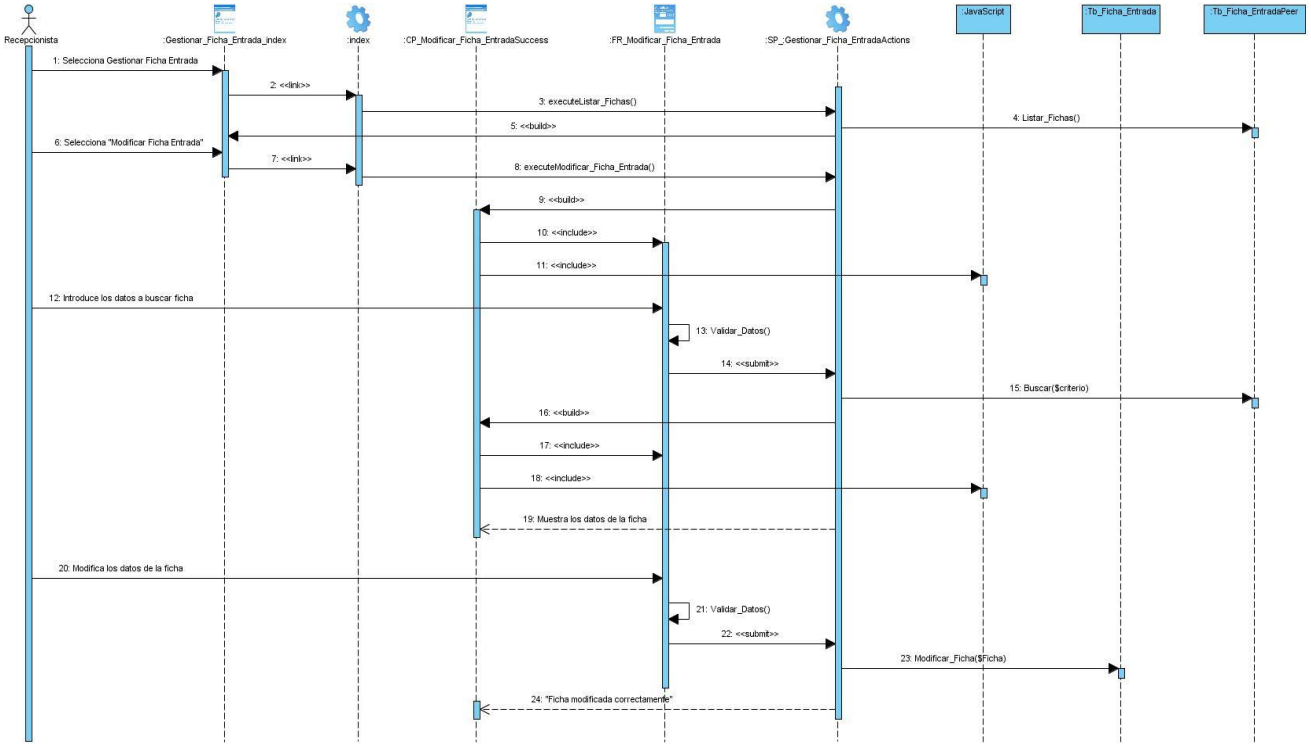


Figura 3.12 CU_Gestionar_Ficha_Entrada Escenario: Modificar.

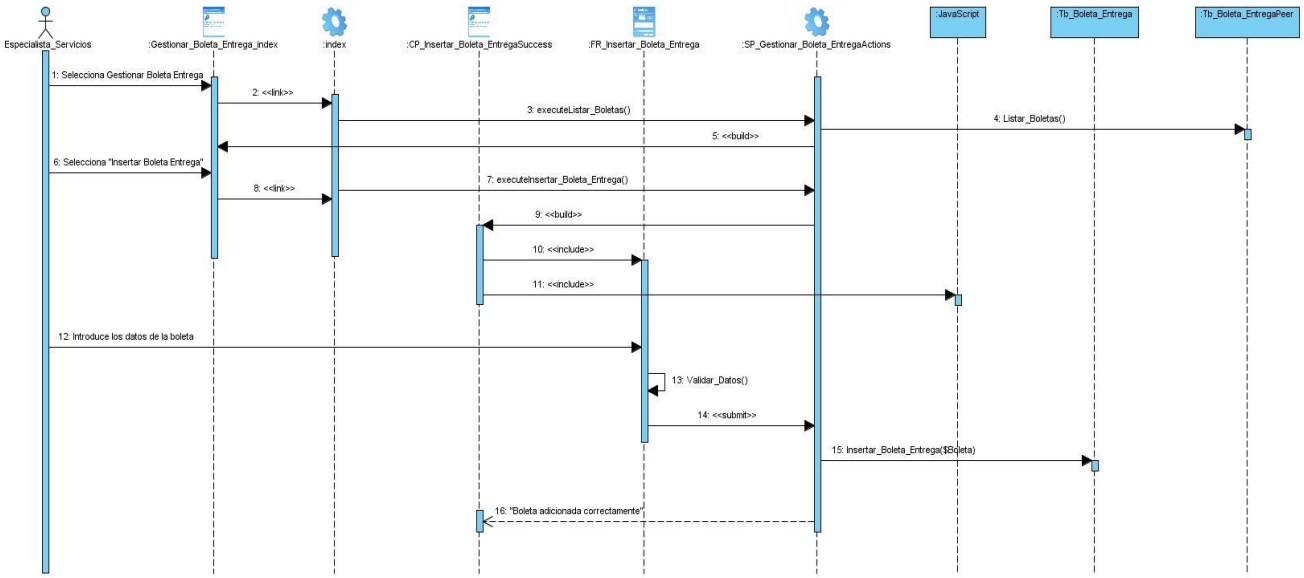


Figura 3.13 CU_Gestionar_Boleta_Entrega Escenario: Insertar.

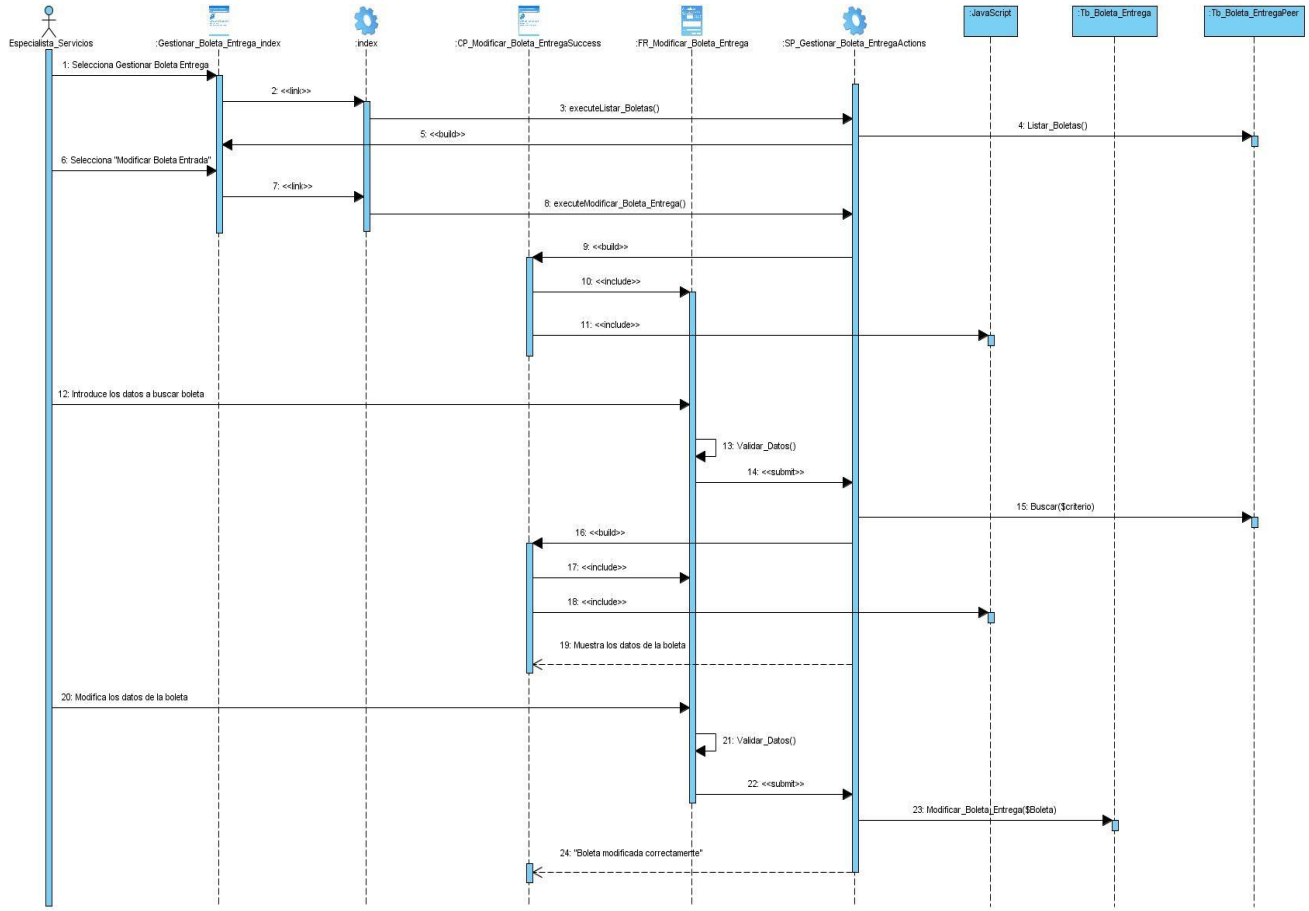


Figura 3.14 CU_Gestionar_Boleta_Entrega Escenario: Modificar.

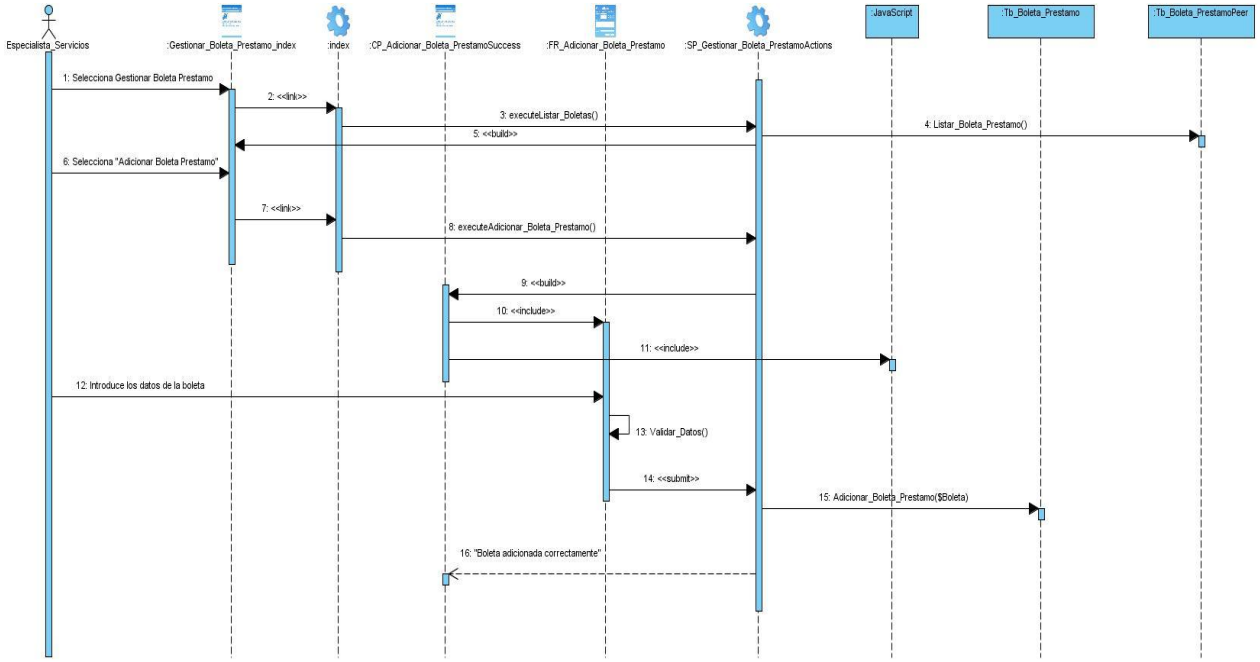


Figura 3.15 CU_Gestionar_Boleta_Préstamo Escenario: Adicionar.

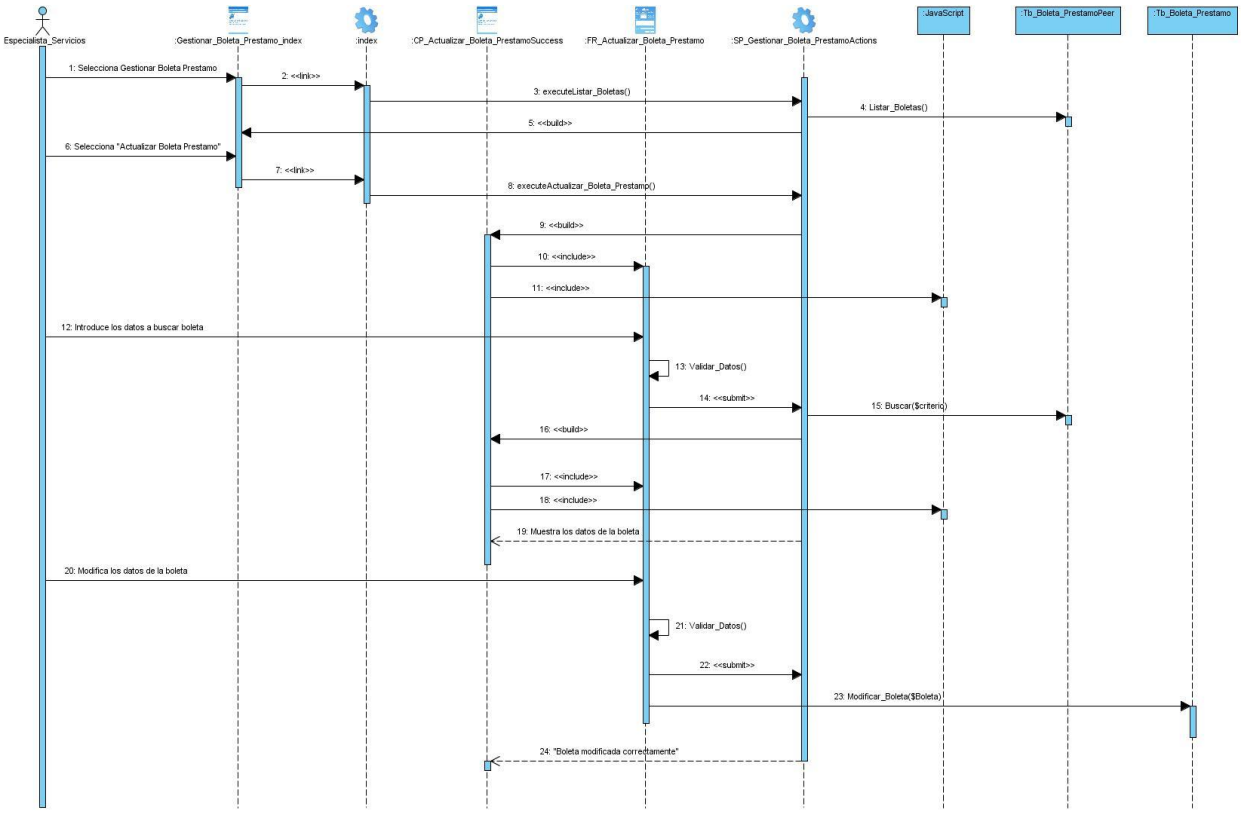


Figura 3.16 CU_Gestionar_Boleta_Préstamo Escenario: Actualizar.

3.9 Descripción de las clases.

Nombre: SP_Gestionar_UsuarioActions.php	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	Listar_Usuario()
Descripción:	Muestra el listado de todos los usuarios.
Nombre:	executeAdicionar_Usuario()
Descripción:	Muestra la vista para adicionar un usuario.
Nombre:	Adicionar_Usuario()
Descripción:	Envía los datos del usuario a adicionar.
Nombre:	executeEliminar_Usuario()
Descripción:	Muestra la vista para eliminar un usuario.
Nombre:	Buscar()
Descripción:	Busca el usuario que cumple con el criterio especificado.
Nombre:	Eliminar_Usuario()
Descripción:	Elimina el usuario seleccionado.
Nombre:	executeModificar_Usuario()
Descripción:	Muestra la vista para modificar un usuario.
Nombre:	Modificar_Usuario()
Descripción:	Modifica los datos del usuario seleccionado.

Figura 3.17 Descripción de la clase SP_Gestionar_UsuarioActions.php.

Nombre: SP_Gestionar_Ficha_Entrada.php	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	Listar_Fichas()
Descripción:	Muestra el listado de todas las fichas.
Nombre:	executeAdicionar_Ficha_Entrada()
Descripción:	Muestra la vista para adicionar una ficha.
Nombre:	Adicionar_Ficha_Entrada()
Descripción:	Envía los datos de la ficha a adicionar.
Nombre:	Buscar()
Descripción:	Busca la ficha que cumple con el criterio especificado.
Nombre:	executeModificar_Ficha_Entrada()
Descripción:	Muestra la vista para modificar una ficha.
Nombre:	Modificar_Ficha_Entrada()
Descripción:	Modifica los datos de la ficha seleccionada.

Figura 3.18 Descripción de la clase SP_Gestionar_Ficha_Entrada.php.

Nombre: SP_Gestionar_Documento.php	
Tipo de clase: Controladora	
Esta es una descripción genérica, solo cambiaría el tipo de documento a gestionar.	
Para cada responsabilidad:	
Nombre:	Listar_Tipos_Documentos()
Descripción:	Muestra la vista con los tipos de documentos.
Nombre:	Buscar_Documentos()
Descripción:	Busca los documentos que cumplen con el criterio especificado.
Nombre:	Listar_Documentos()
Descripción:	Muestra el listado de todos los documentos según el criterio.
Nombre:	executeAdicionar_Documento()
Descripción:	Muestra la vista para adicionar un documento.
Nombre:	Adicionar_Documento()
Descripción:	Envía los datos del documento a adicionar.
Nombre:	executeEliminar_Documento()
Descripción:	Muestra la vista para eliminar un documento.
Nombre:	Buscar()
Descripción:	Busca el documento que cumple con el criterio especificado.
Nombre:	Eliminar_Documento()
Descripción:	Elimina el documento seleccionado.
Nombre:	executeModificar_Documento()
Descripción:	Muestra la vista para modificar un documento.
Nombre:	Modificar_Documento()
Descripción:	Modifica los datos del documento seleccionado.

Figura 3.19 Descripción de la clase SP_Gestionar_Documento.php.

Nombre: SP_Gestionar_Descriptor.php	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	Listar_Descriptor()
Descripción:	Muestra el listado de todos los descriptores.
Nombre:	executeAdicionar_Descriptor()
Descripción:	Muestra la vista para adicionar un descriptor.
Nombre:	Adicionar_Descriptor()
Descripción:	Envía los datos del descriptor a adicionar.
Nombre:	executeEliminar_Descriptor()
Descripción:	Muestra la vista para eliminar un descriptor.
Nombre:	Buscar()
Descripción:	Busca el descriptor que cumple con el criterio especificado.
Nombre:	Eliminar_Descriptor()
Descripción:	Elimina el descriptor seleccionado.
Nombre:	executeModificar_Descriptor()
Descripción:	Muestra la vista para modificar un descriptor.
Nombre:	Modificar_Descriptor()
Descripción:	Modifica los datos del descriptor seleccionado.

Figura 3.20 Descripción de la clase SP_Gestionar_Descriptor.php.

Nombre: SP_Gestionar_Boleta_Uso_Máquina.php	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	Listar_Boletas()
Descripción:	Muestra el listado de todas las boletas.
Nombre:	executeAdicionar_Boleta_Uso_Máquina()
Descripción:	Muestra la vista para adicionar una boleta.
Nombre:	Adicionar_Boleta_Uso_Máquina()
Descripción:	Envía los datos de la boleta a adicionar.
Nombre:	Buscar()
Descripción:	Busca la boleta que cumple con el criterio especificado.
Nombre:	executeModificar_Boleta_Uso_Máquina()
Descripción:	Muestra la vista para modificar una boleta.
Nombre:	Modificar__Boleta_Uso_Máquina()
Descripción:	Modifica los datos de la boleta seleccionada.

Figura 3.21 Descripción de la clase SP_Gestionar_Boleta_Uso_Máquina.php.

Nombre: SP_Gestionar_Boleta_Sala_Lectura.php	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	Listar_Boletas()
Descripción:	Muestra el listado de todas las boletas.
Nombre:	executeAdicionar_Boleta_Sala_Lectura()
Descripción:	Muestra la vista para adicionar una boleta.
Nombre:	Adicionar_Boleta_Sala_Lectura()
Descripción:	Envía los datos de la boleta a adicionar.
Nombre:	Buscar()
Descripción:	Busca la boleta que cumple con el criterio especificado.
Nombre:	executeModificar_Boleta_Sala_Lectura()
Descripción:	Muestra la vista para modificar una boleta.
Nombre:	Modificar__Boleta_Sala_Lectura()
Descripción:	Modifica los datos de la boleta seleccionada.

Figura 3.22 Descripción de la clase SP_Gestionar_Boleta_Sala_Lectura.php.

Nombre: SP_Gestionar_Boleta_Préstamo.php	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	Listar_Boletas()
Descripción:	Muestra el listado de todas las boletas.
Nombre:	executeAdicionar_Boleta_Préstamo()
Descripción:	Muestra la vista para adicionar una boleta.
Nombre:	Adicionar_Boleta_Préstamo()
Descripción:	Envía los datos de la boleta a adicionar.
Nombre:	Buscar()
Descripción:	Busca la boleta que cumple con el criterio especificado.
Nombre:	executeModificar_Boleta_Préstamo()
Descripción:	Muestra la vista para modificar una boleta.
Nombre:	Modificar__Boleta_Préstamo()
Descripción:	Modifica los datos de la boleta seleccionada.

Figura 3.23 Descripción de la clase SP_Gestionar_Boleta_Préstamo.php.

Nombre: SP_Gestionar_Boleta_Entrega.php	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	Listar_Boletas()
Descripción:	Muestra el listado de todas las boletas.
Nombre:	executeAdicionar_Boleta_Entrega()
Descripción:	Muestra la vista para adicionar una boleta.
Nombre:	Adicionar_Boleta_Entrega()
Descripción:	Envía los datos de la boleta a adicionar.
Nombre:	Buscar()
Descripción:	Busca la boleta que cumple con el criterio especificado.
Nombre:	executeModificar_Boleta_Entrega()
Descripción:	Muestra la vista para modificar una boleta.
Nombre:	Modificar__Boleta_Entrega()
Descripción:	Modifica los datos de la boleta seleccionada.

Figura 3.24 Descripción de la clase SP_Gestionar_Boleta_Entrega.php.

Nombre: SP_Buscar_Información_Referencial.php	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	executeBuscar_Informacion:_Referencial()
Descripción:	Muestra la vista para seleccionar tipo de documento a buscar.
Nombre:	Buscar()
Descripción:	Busca los documentos que cumple con los criterios especificados.
Nombre:	Listar_Documento()
Descripción:	Muestra los documentos que cumplen con los criterios seleccionados.

Figura 3.25 Descripción de la clase SP_Buscar_Información_Referencial.php.

Nombre: SP_Buscar_Información_Digital.php	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	executeBuscar_Informacion:_Digital()
Descripción:	Muestra la vista para seleccionar tipo de documento a buscar.
Nombre:	Buscar()
Descripción:	Busca los documentos que cumple con los criterios especificados.
Nombre:	Listar_Documento()
Descripción:	Muestra los documentos que cumplen con los criterios seleccionados.

Figura 3.26 Descripción de la clase SP_Buscar_Información_Digital.php.

Nombre: SP_Buscar_Boleta_Recepcionista.php	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	executeBuscar_Boleta_Recepcionista()
Descripción:	Muestra la vista para seleccionar tipo de boleta a buscar.
Nombre:	Buscar()
Descripción:	Busca las boletas que cumple con los criterios especificados.
Nombre:	Listar_Boletas()
Descripción:	Muestra las boletas que cumplen con los criterios seleccionados.

Figura 3.27 Descripción de la clase SP SP_Buscar_Boleta_Recepcionista.php.

Nombre: SP_Buscar_Boleta_Especialista.php	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	executeBuscar_Boleta_Especialista()
Descripción:	Muestra la vista para seleccionar tipo de boleta a buscar.
Nombre:	Buscar()
Descripción:	Busca las boletas que cumple con los criterios especificados.
Nombre:	Listar_Boletas()
Descripción:	Muestra las boletas que cumplen con los criterios seleccionados.

Figura 3.28 Descripción de la clase SP_Buscar_Boleta_Especialista.php.

Nombre: SP_Autenticar_UsuarioActions.php	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	executeAutenticar_Usuario()
Descripción:	Muestra la vista para autenticar el usuario.
Nombre:	Autenticar_Usuario()
Descripción:	Envía el usuario y la contraseña para levantar la sesión del usuario.
Nombre:	Cargar_Rol()
Descripción:	Le asigna al usuario autenticado el rol correspondiente.

Figura 3.29 Descripción de la clase SP_Autenticar_UsuarioActions.php.

3.10 Validación del diseño.

Las métricas tienen por objetivo medir la calidad de los productos intermedios generados en un proyecto de software. La palabra calidad designa el conjunto de atributos o propiedades de un objeto que permiten emitir un juicio de valor acerca de él. Hay cuatro razones para medir los procesos del software, los productos y los recursos: caracterizar, evaluar, predecir y mejorar.

Se caracteriza para comprender mejor los procesos, los productos, los recursos y los entornos, y establecer las líneas base para las comparaciones con evaluaciones futuras. Se evalúa para determinar el estado con respecto al diseño. Se predice para poder planificar. Se hace esto porque se quieren establecer objetivos alcanzables para el costo, planificación, y calidad de manera que se puedan aplicar los recursos apropiados. Se mide para mejorar, la medición permite recoger la información cuantitativa que ayuda a identificar obstáculos, problemas de raíz, ineficiencias y otras oportunidades para mejorar la calidad del producto y el rendimiento del proceso (Pressman, 2005).

Validar el diseño permite entregar artefactos fiables a los desarrolladores encargados de implementar el sistema, para esto se utilizaron las métricas orientadas a clases: Tamaño de clase (TC) y Árbol de Profundidad de Herencia (APH).

Tamaño de clase (TC)

Se aplicó esta métrica para las principales clases definidas en el diseño.

No.	Clases	Cantidad de atributos	Cantidad de operaciones	Tamaño
1	CC_Autenticar_Usuario	1	3	Pequeño
2	Usuario	2	2	Pequeño
3	CC_Gestionar_Usuario	1	8	Pequeño
4	Rol	1	2	Pequeño
5	CC_Gestionar_Ficha_Entrada	1	6	Pequeño
6	Ficha_Entrada	11	2	Pequeño
7	CC_Gestionar_Boleta_Entrega	1	6	Pequeño
8	Boleta_Entrega	11	2	Pequeño
9	CC_Gestionar_Boleta_Prestamo	1	6	Pequeño
10	Boleta_Prestamo	15	2	Pequeño
11	CC_Gestiona_Boleta_Sala_Lectura	1	6	Pequeño
12	Boleta_Sala_Lectura	5	2	Pequeño
13	CC_Gestionar_Boleta_Uso_Maquina	1	6	Pequeño
14	Boleta_Uso_Maquina	8	2	Pequeño
15	CC_Buscar_Informacion_Referencial	5	3	Pequeño
16	DispCon_Gob_Referencial	5	2	Pequeño
17	Referencia	23	2	Medio
18	Publicacion_Seriada	24	2	Medio
19	Flujo_Ascendente	18	2	Pequeño
20	CD	20	2	Medio
21	CC_Buscar_Informacion_Digital	5	3	Pequeño
22	DispCon_Gob_Digital	6	2	Pequeño
23	Legislación	14	2	Pequeño
24	Sentencia	13	2	Pequeño
25	Investigación	13	2	Pequeño
26	Doctrina	17	2	Pequeño
27	CC_Gestionar_Documento	10	10	Pequeño
28	CC_Gestionar_Boleta_Recepcionista	1	3	Pequeño
29	CC_Gestionar_Boleta_Especialista	4	3	Pequeño
30	CC_Gestionar_Descriptor	1	8	Pequeño
31	Descriptor	1	2	Pequeño

Figura 3.30 Representación del tamaño de las clases del sistema SIDIJ.

Se trabajó con un total de **31 clases** para un **promedio de 7.74 atributos** por clases y **de 3.45 de operaciones**, como se muestra en la figura 3.31.

Cantidad de clases	Promedio de atributos	Promedio de operaciones
31	7.74	3.45

Figura 3.31 Resultados de la Métrica TC.

La mayoría de las clases quedaron en la clasificación de "pequeño" y algunas en "medio", este indicador demuestra que las clases se pueden reutilizar y que el sistema no tendrá una compleja implementación, por lo que los resultados obtenidos son positivos.

En la siguiente tabla se muestran los resultados:

Clasificación	Cantidad de clases
Pequeño	28
Medio	3
Grande	0

Figura 3.32 Cantidad de clases por clasificación.

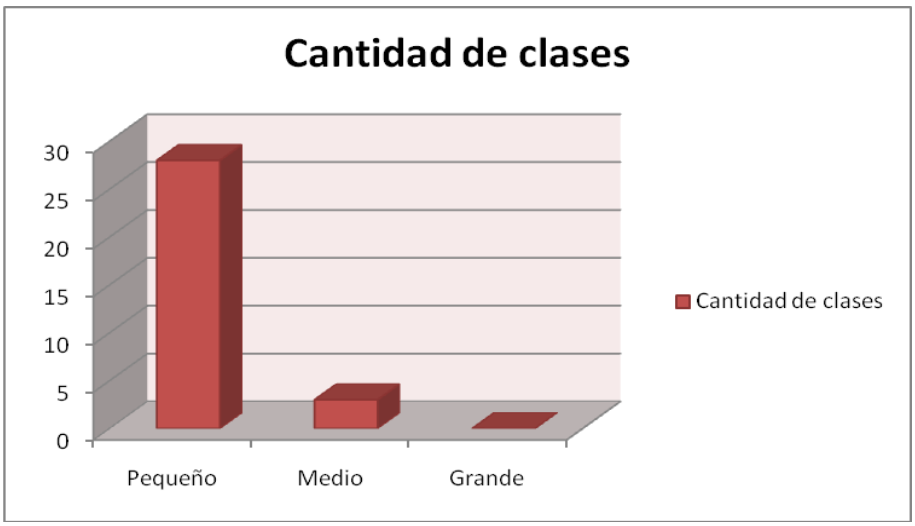


Figura 3.33 Gráfico: Cantidad de clases por categoría.

Árbol de Profundidad de Herencia (APH)

Esta métrica está definida por la máxima longitud que existe entre el nodo y la raíz del árbol. Si los valores de APH son grandes, entonces se garantiza que se reutilice gran cantidad de código, pero al mismo tiempo hace que el diseño sea más complejo. Esto provoca un mayor acoplamiento entre las clases.

En el sistema SIDIJ no fue necesario hacer mucho uso de herencia por lo que no se presenta un alto nivel de jerarquía de la misma.

Aplicando esta métrica al diseño propuesto se obtiene un árbol de herencia que toma valor 1, evidenciando que existe bajo acoplamiento, lo que garantiza la poca complejidad del diseño.

3.11 Conclusiones

Al culminar la modelación del diseño del sistema se refina el diagrama de clases y las relaciones entre estas, transformando los requerimientos en cómo va a ser implementado el sistema. A través de los diagramas de interacción se refleja la colaboración entre las clases del diseño mostrando la ordenación temporal de los mensajes que intercambian los objetos para realizar las tareas necesarias.

- Se definió la estructura de la aplicación dentro del framework Symfony.
- Con el desarrollo de los diferentes artefactos se ha logrado un diseño preparado para la implementación del software.
- La aplicación de los patrones de diseño permitió resolver problemas que contribuyeron al incremento de la calidad.
- El 90 % de las clases diseñadas están consideradas como pequeñas, lo que facilitará el proceso de construcción del sistema.
- Las métricas aplicadas durante la validación permitieron comprobar la fiabilidad y calidad del diseño elaborado.

Conclusiones Generales

Al concluir el presente trabajo, se logró cumplir con el objetivo general, obteniendo el análisis y diseño del Sistema Integral de Documentación e Información Judicial (SIDIJ), que permitirá la implementación del sistema, solucionándose de esta forma, la limitación presentada anteriormente por el Centro Nacional de Documentación e Información Judicial (CENDIJ) .

- Con el estudio de las tendencias y tecnologías actuales en aplicaciones de software, se definieron las herramientas adecuadas, que permitieron obtener de forma eficiente los artefactos en el flujo de trabajo Análisis y Diseño.
- Para modelar el sistema se tuvieron en cuenta patrones de diseño permitiendo obtener los artefactos de análisis y diseño que facilitarán la implementación futura del sistema basándose en un modelo robusto, flexible y poco complejo.
- Entre los artefactos generados se pueden mencionar: diagramas de clases y de colaboración del análisis, y diagramas de clases y secuencia del diseño, imprescindibles para el desarrollo de la solución informática.
- Los resultados fueron validados satisfactoriamente, lo que permite demostrar que el análisis y diseño realizado son fiables y poseen la calidad necesaria para dar paso a la implementación del sistema.

Recomendaciones

El Sistema Integral de Documentación e Información Judicial (SIDIJ) permitirá gestionar adecuadamente los procesos que se realizan en el Centro Nacional de Documentación e Información Judicial (CENDIJ), por lo que se recomienda:

- A partir de los artefactos generados implementar el SIDIJ, que una vez terminado deberá ser sometido a un proceso de pruebas por el Grupo de Calidad de la Facultad.
- Para próximas versiones desarrollar otras funcionalidades que puedan ser sugeridas por los clientes.
- Integrar el SIDIJ al sistema que se desarrolla la Facultad 3 para el Tribunal Supremo Popular.

Referencias bibliográficas.

Albin, Stephen. 2003. The Art of Software Architecture: Design methods and techniques. Nueva York: Wiley, 2003.

Booch. 1996. Anáisis y diseño orientado a objetos con aplicaciones (2da Edición).

Buschmann F., y otros. 1996. Pattern Oriented Software Architecture: A System of Patterns.

Chidamber S. R., Kemerer C. F. 1994. "A metric suite forsz object oriented design", lEEE Transactions on Software Engineering. 1994.

Clements, Paul y Northrop, Linda. 1996. Software architecture: An executive overview. s.l.: Technical Report, 1996.

Coding Standar for PHP. 2005. Coding Standar for PHP. [En línea] 2005. [Citado: diciembre 10, 2010.] http://alltasks.net/code/php_coding_standard.html#important

Fowler, Martin. 2002. Patterns of Enterprise Application Architecture. s.l.: Addison Wesley, 2002.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (2003). Patrones de diseño. Elementos de software orientado a objetos reutilizable. C. Fernández, Trad. Madrid: Pearson Education, S.A.

González Duque, Raúl. 2003. Python para todos. 2003.

Jacobson, G.B. Ivar. 2000. Proceso Unificado de Desarollo de Software. Madrid : Addison Wesley, 2000.

Kroll, Per y Kruchten, Philippe. abril 08, 2003. The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP.s.l.: Addison-Wesley Profesional, abril 08, 2003.

Larman Graig. 1999. UML y Patrones. [citado 25 de febrero, 2011]. http://bibliodoc.uci.cu/pdf/reg00061.pdf.

Navarro, J. 2006. Recuperado en enero de 2011, de http://agu.inter.edu/jnavarro/comp3400Lec08EstructuracionEspecProce.pdf

Ocaña, J. y Sánchez, A. 2003. Diseño orientado a objetos y software estadístico: patrones de diseño, UML y composición vs. Herencia.

Referencias bibliográficas.

Perry, Dewayne. 1997. Software Architecture and its relevance for Software Engineering. s.l. : Coord'97, 1997.

PostgreSQL. 2008. [En línea] 2008. [Citado: febrero 10, 2011.] http://www.postgresql.org/about/

Pressman, Roger S. 2002. Ingeniería de Software. Un enfoque practico. s.l. : McGraw.Hill/Interamericana de España., 2002. 5.

Pressman, Roger. 2005. Ingeniería del Software. Un Enfoque práctico. Sexta edición. MacGraw-Hill, 2005.

Reynoso, Carlos Billy y Kicillof, Nicolás. 2004. MSDN Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. [En línea] 2004. [Citado: febrero 20, 2011.] http://www.microsoft.co.ke/spanish/msdn/arquitectura/roadmap_arq/style.aspx

—. **2005**. MSDN webcast#1. Seminario de Arquitectura de Software. [En línea] 2005. [Citado: febrero 22, 2011.] http://www.microsoft.com/spanish/msdn/latam/mediacenter/webcast/architect.aspx

—. **2005**. MSDN webcast#2. Profundizando en Estilos de Arquitectura de Software. [En línea] 2005. [Citado: febrero 20, 2011.] http://www.microsoft.com/spanish/msdn/latam/mediacenter/webcast/architect.aspx

Reynoso, Carlos Billy. 2006. MSDN Introducción a la Arquitectura de Software. [En línea] 2006. [Citado: marzo 15, 2011.] http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.mspx

Szyperski, Clemens. 2000. Components and Architecture. Software Development. 2000.

Van Schermbeek, Mariangela Pocaterra, 2008. Proyecto final de Máster en Tecnologías de la Información Geográfica, 10ª edición. [En línea] 2008. [Citado: marzo 20, 2011.] http://www.recercat.net/bitstream/2072/41829/1/Treball+de+recerca.pdf

Velasco, Carlos Canal. 2000. Un Lenguaje para la Especificación y Validación de Arquitecturas de Software. Málaga: Universidad de Málaga, 2000.

Vilain, P., Schwabe, D. y Sieckenius, C. 2000. A Diagrammatic Tool for Representing USer Interection in UML. York, England : s.n., 2000.

Wesley, Addison. 2000. Manual de Referencia. 2000.

Glosario de Términos

Términos informáticos:

AOLserver: es un servidor web de código abierto, tiene procesamiento multihilo, y soporte para Tcl, se usa para sitios web dinámicos de gran tamaño.

API: Aplication Programming Interface – Interfaz de Programación de Aplicaciones, es el conjunto de funciones y procedimientos que ofrece cierta librería para ser utilizada por otro software.

Base de datos: Conjunto de datos no redundantes, almacenados en un soporte informático, organizado de forma independiente de su utilización y accesible simultáneamente por distintos usuarios y aplicaciones. La diferencia de una BD respecto a otro sistema de almacenamiento de datos es que estos se almacenan en la BD de forma que cumplen tres requisitos básicos: no redundancia, independencia y concurrencia.

Código abierto: En inglés Open Source, conlleva como idea más importante la posibilidad de acceder al código fuente, modificarlos y distribuirla como establezca la licencia bajo la que está distribuido.

CSS: Cascade Style Sheet. Conjunto de instrucciones HTML que definen la apariencia de uno o más elementos de un conjunto de páginas Web con el objetivo de uniformizar su diseño.

CVS: Aplicación que implementa un sistema de control de versiones.

Framework: Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

GNU: GNU/Linux es uno de los términos empleados para referirse a la combinación del núcleo o kernel libre similar a Unix denominado Linux, que es usado con herramientas de sistema GNU. Su desarrollo es uno de los ejemplos más prominentes de software libre; todo su código fuente puede ser utilizado, modificado y redistribuido libremente por cualquiera bajo los términos de la GPL (Licencia Pública General de GNU, en inglés: General Public License).

Glosario de términos.

GPL: Son las siglas en inglés General Public License que es una licencia creada por la Free Software Foundation en 1989 y está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

Herramienta CASE: Las herramientas CASE (Computer Aided Software Engineering: Ingeniería de Software Asistida por Ordenadores) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo del software reduciendo el coste de las mismas en términos de tiempo y dinero.

IDE: Son las siglas en inglés de Integrated Development Enviromment (Ambiente de Desarrollo Integrado).

IEEE: Son las siglas en inglés de The Institute of Electrical and Electronics Engineers (El Instituto de Ingenieros Eléctricos y Electrónicos), una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas.

Multihebra: estilo de ejecución en el que se conmuta entre distintas partes del código de un mismo programa durante la ejecución.

MVC: "Modelo, Vista, Controlador" es un patrón arquitectónico que permite organizar el código de una aplicación, facilitando modificar una de las capas sin necesidad de modificar las restante, así como la posterior reutilización del código.

Objeto: En el paradigma de la Programación Orientada a Objetos (POO, o OOP en inglés), un objeto es la unidad individual que en tiempo de ejecución realiza las tareas de un programa.

OpenACS: del inglés Open Architecture Community System (Arquitectura Abierta para Sistemas de Comunidades) es un kit de herramientas libre (de código abierto) para el desarrollo rápido de aplicaciones web, con licencia GPL.

Plataforma: En informática, una plataforma es precisamente el principio, ya sea de hardware o software, sobre el cual un programa puede ejecutarse. Ejemplos típicos incluyen: arquitectura de hardware, sistema operativo, lenguajes de programación y sus librerías de tiempo de ejecución.

Plataforma J2EE (Java 2 Platform Enterprise Edition): Consiste en un conjunto de servicios API y protocolos que proporcionan la funcionalidad para el desarrollo, basado en la Web de las aplicaciones de varios niveles.

Glosario de términos.

PHP: Hypertext Preprocessor. Lenguaje de script diseñado para la creación de páginas Web activas, muy popular en Linux, destaca por su capacidad de ser embebido en el código HTML.

RUP: Rational Unified Process es un proceso de desarrollo de software y junto con el Lenguaje de Modelado Unificado (UML), constituye una de las metodologías estándares más utilizadas para el análisis, implementación y documentación de sistemas orientados a objetos.

Scrum: Metodología ágil para el desarrollo de sistemas software.

SIDIJ: Sistema Integral de Documentación e Información Judicial.

SOAP: Son las siglas en inglés de Simple Object Access Protocol (Protocolo Simple de Acceso a Datos). Es un protocolo estándar que permite que dos objetos de diferente arquitectura puedan comunicarse mediante el intercambio de XML.

Software: Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.

SVN: Subversion es un sistema de control de versiones, de software libre bajo una licencia de tipo Apache/BSD. Es utilizado como repositorio donde se tiene un único número de versión que identifica un estado común de todos los archivos que se encuentran en el repositorio.

TCL: "Tool Command Language" o lenguaje de herramientas de comando, es un lenguaje de script creado por John Ousterhout, que ha sido concebido con una sintaxis sencilla para facilitarse su aprendizaje, sin ir en desmedro de la funcionalidad y expresividad. Se utiliza principalmente para el desarrollo rápido de aplicaciones "script" e interfaces gráficas.

Tecnologías de la información y la comunicaciones (TIC): Conjunto de avances tecnológicos que proporcionan la informática, las telecomunicaciones y las tecnologías audiovisuales, que comprenden los desarrollos relacionados con los ordenadores, Internet, la telefonía, las aplicaciones multimedia y la realidad virtual. Estas tecnologías básicamente proporcionan información, herramientas para su proceso y canales de comunicación.

UML: Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

Glosario de términos.

Términos Judiciales:

CENDIJ: Centro Nacional de Documentación e Información Judicial.

Descriptor: Es cada uno de los términos o expresiones escogidos entre un conjunto de sinónimos o cuasi sinónimos para representar generalmente de manera unívoca, un concepto susceptible de aparecer con cierta frecuencia en los documentos indizables, y en las consultas que se realicen.

Disposición del Consejo de Gobierno: Instrucciones, Acuerdos y Dictámenes del Consejo de Gobierno del Tribunal Supremo Popular.

Doctrina: Conjunto de ideas u opiniones sustentadas por una persona o un grupo.

Doctrina Legal: La doctrina establecida por los tribunales al interpretar y aplicar de modo reiterado e idéntico las leyes, y que referida al Tribunal Supremo, configura la jurisprudencia.

Flujo Ascendente: trabajos de investigaciones, tesis, tesinas y otros trabajos científicos.

Fondo general: Es el conjunto de documentos que conforman la colección de la biblioteca del CENDIJ, cuyas temáticas no pertenecen a la especialidad de Derecho y Justicia.

Legislación: Ley, Acuerdo o Decreto-Ley de la Asamblea Nacional del Poder Popular, perteneciente al Centro de Documentación (CENDIJ) y a la Biblioteca de la Asamblea Nacional por colaboración e intercambio.

Sentencia: Acto del Órgano jurisdiccional que pone término al proceso resolviendo todas las cuestiones litigiosas planteadas por las partes.