# UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

## Facultad 3


**Título:** Diseño e implementación del módulo Ingresos Comerciales del GINA 1.1

**Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas**

**Autor:** Martha Maria González Leyva

**Tutores:** Ing. Iliana de la Rosa Zayas Frías

Ing. Idel Jorge Sánchez González

La Habana, Junio de 2014.

**"Año 56 de la Revolución"**

## DECLARACIÓN DE AUTORÍA

Declaro ser la autora de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los 24 días del mes de Junio del año 2014.

_____

Martha Maria González Leyva

Firma del Autor

_____

Ing. Iliana de la Rosa Zayas Frías

Firma del Tutor

_____

Ing. Idel Jorge Sánchez González

Firma del Tutor

# *Agradecimientos*

*Han sido cinco años de esfuerzos y sacrificios, cerrada esta etapa, me queda agradecer:*

*A Dios por permitirme haber llegado hasta este momento tan importante de mi formación profesional, por todo su amor y su fidelidad, espero nunca soltarme de su mano.*

*A mi mamá, por quererme tanto, por tener el corazón tan grande que tiene, por ser mi ejemplo de fuerza, de trabajo, de sacrificio, de amor, hoy soy una mejor persona gracias a ti. Gracias mami porque siempre he sabido que has luchado para darme lo mejor que has podido, por eso de esta manera hoy te lo agradezco desde el fondo de mi corazón, te amo mucho.*

*A mi papá, que con su amor y enseñanza ha sembrado en mí las virtudes que se necesitan para vivir con anhelo y felicidad, por ayudarme a ser quien soy hoy, por demostrarme siempre su cariño y apoyo incondicional. Por creer en mí y darme la confianza para crecer en la vida.*

*A mi amado novio y futuro esposo, Anel, quien ha sido el motor impulsor de mi vida durante este tiempo juntos y el pilar principal para la culminación de mi carrera, que con su apoyo constante y amor incondicional ha sido amigo y compañero inseparable, fuente de sabiduría, calma y consejo en todo momento. Gracias por amarme como solo tú lo puedes hacer.*

*A mis hermanos bellos que tanto adoro, Ilianita, por ser la mejor hermana, a Alejandro por estar conmigo siempre, a Orlandito, por quererme tanto. A los tres por todos los momentos felices que hemos vivido, son los mejores del mundo.*

*A mi abuela Margarita, a quien quiero por ser la mejor de las abuelas, por cuidarme y por siempre estar dispuesta a escucharme y a ayudarme en cualquier momento.*

*A mis abuelos, gracias por su amor, por estar siempre ahí para mí.*

*A mis tíos Titia y Raciel, gracias por todo lo que han hecho por mí en estos cinco años, gracias porque son la representación de toda mi familia.*

*A mi amiga durante estos últimos años, Lili, gracias por soportarme y por ser tan especial en mi vida. Hoy deseo agradecerte una vez más por estar junto a mí en los momentos que más requerí de tu ayuda, eres una gran amiga y me lo has demostrado con hechos. Te quiero mucho.*

*A mi amigo Osiel, por todos estos años de amistad, tengo que agradecerte cada consejo oportuno y cada regaño, gracias de todo corazón.*

*A los profesores que ayudaron en mi formación profesional.*

*A mi familia en general, porque me han brindado su apoyo incondicional y por compartir conmigo buenos y malos momentos.*

*A mis compañeros de grupo porque de todos aprendí algo en estos cinco años.*

*Quiero agradecer al resto de mis amistades y compañeros, a todos, muchas gracias.*

## *Dedicatoria*

*A Dios por acompañarme todos los días.*

*A mi mamá y a mi papá, porque aun cuando están lejos físicamente, no me han dejado sola ni un solo segundo. Porque en cada tropiezo, ante cada momento difícil de mi vida siempre tuve de ellos un gesto de amor y dedicación. Siempre me apoyaron para que estudiara, me superara y me enseñaron que con esfuerzo todo se puede alcanzar.*

*A mi novio, por lo especial y maravilloso que ha sido conmigo, por brindarme tanto amor y comprensión, por ser paciente y por ayudarme tanto en todo, por alentarme para continuar, cuando parecía que me iba a rendir.*

# Resumen

El objetivo fundamental de la Aduana General de la República de Cuba (AGR) es el control de personas, mercancías y tripulantes en las fronteras cubanas. Para ello se basa en un conjunto de leyes y regulaciones que definen las reglas para la entrada y salida de los mismos. Dentro de los procedimientos que se llevan a cabo en la AGR, se encuentra la gestión de los ingresos comerciales, la cual se realiza con el Sistema Único de Aduana (SUA). Dicho sistema no usa las potencialidades brindadas por los marcos de trabajo de desarrollo de aplicaciones web y las consultas a la base de datos se realizan mediante procedimientos almacenados. Por tal motivo el objetivo del presente trabajo es desarrollar un módulo que gestione el cobro de los ingresos comerciales de derechos y servicios de aduana dentro del sistema Gestión Integral de Aduanas, sustituyendo de esta forma al SUA.

Para su desarrollo se utilizó la metodología de desarrollo del Centro de Gestión de Entidades, como sistema gestor de base de datos Oracle 11g, como marcos de trabajo Symfony 1.2.8 y ExtJS 3.0, y como entorno integrado de desarrollo NetBeans 7.2. Con las pruebas y validaciones realizadas se muestra que el módulo desarrollado cumple con las especificaciones y requisitos definidos en la etapa de concepción del sistema.



**Palabras claves:** Aduana, Gestión Integral de Aduanas, Ingresos Comerciales.

# Índice general

# Índice de figuras

# Índice de tablas

# Introducción

La industria del software en Cuba ha estado evolucionando en los últimos tiempos. Muchas empresas del país están solicitando productos informáticos para agilizar su trabajo, ganar en eficacia y ofrecer servicios de mejor calidad. Un ejemplo de ello se tiene en la Aduana General de la República de Cuba (AGR), la cual es una organización creada con el objetivo fundamental de garantizar la seguridad nacional.

La AGR está hace más de 10 años informatizando la mayoría de sus sectores, áreas y procesos para brindar a sus empleados una vía más cómoda y segura de trabajo, con el objetivo de ofrecer un servicio de mejor eficiencia y calidad. Actualmente la AGR cuenta con un sistema para la gestión de sus operaciones, el Sistema Único de Aduana (SUA) vigente desde el año 2008, el cual consta de diferentes módulos. Dentro de este sistema de gestión se encuentra implementado un módulo para los ingresos comerciales, que se encarga de gestionar los procesos del departamento de Economía, un área de vital importancia para la AGR.

El principal objetivo para la automatización de los ingresos comerciales dentro del SUA es garantizar el proceso de recaudación en las Aduanas de Despacho, donde se lleva a cabo el cobro de mercancías, aranceles, multas y servicios prestados. Además, permite el cobro de los cheques que han sido devueltos por el banco, las resoluciones de multas aplicadas a una empresa y las facturas por los servicios aduaneros utilizados.

Los especialistas del Centro de Informatización para la Dirección de la Aduana (CADI[1]), en conjunto con la Universidad de las Ciencias Informáticas (UCI), decidieron desarrollar el sistema Gestión Integral de Aduanas (GINA), para lograr la integración de todos los procesos que se realizan en la AGR. Este sistema brinda una serie de beneficios para los clientes y usuarios que interactúen con el mismo, permitiendo el intercambio de información a través de servicios web, lo que trae consigo que se comunique con otros sistemas; facilita la gestión de nomencladores, el control de acceso, la gestión de los usuarios, dominios, roles, permisos y trazas. Además utiliza las ventajas brindadas por el marco de trabajo (en lo adelante framework) de interfaz de usuario ExtJS en su versión 3.0, lo que posibilita una mejor visualización de la información.

[1] CADI: Centro de Automatización para la Dirección y la Información, centro dedicado al desarrollo de soluciones informáticas para la aduana, en él se concentran los especialistas de la rama y que tienen precisamente como propósito la creación de aplicaciones que intervengan en los diferentes procesos aduaneros.

El SUA posee una arquitectura incompatible con el sistema GINA, lo que lo hace incapaz de integrarse con este último. Entre las características que impiden integrar los sistemas antes mencionados se destaca el hecho de que el SUA no está implementado con las ventajas que ofrecen los frameworks de desarrollo, en su lugar está concebido en PHP estructurado y las consultas a la base de datos se realizan a través de procedimientos almacenados. Además, la AGR desea que esté presente en el sistema GINA el módulo Ingresos Comerciales, ya que este es un proceso esencial para las Aduanas de Despacho.

Ante la situación expuesta anteriormente, surge el siguiente **problema a resolver**: ¿Cómo obtener un producto que gestione el cobro de los ingresos comerciales para el sistema Gestión Integral de Aduanas?

Como **objeto de estudio** se define: Los sistemas de gestión de los ingresos comerciales de Aduanas.

Para dar solución al problema planteado se define como **objetivo general** de la presente investigación: Desarrollar un módulo que gestione el cobro de los ingresos comerciales de derechos y servicios de aduana para el sistema GINA 1.1, como **campo de acción**: Los sistemas de gestión de ingresos comerciales en la Aduana General de la República de Cuba.

A partir de un análisis del objetivo general se derivan los siguientes **objetivos específicos**:

1. Realizar el marco teórico de la investigación.
2. Diseñar e implementar la solución modelada de manera que cumpla con las necesidades del cliente.
3. Validar la propuesta de solución.

Se plantea como **idea a defender**: Si se desarrolla el módulo Ingresos Comerciales para el sistema GINA 1.1, se permitirá la gestión de los ingresos comerciales por conceptos de derechos y servicios de Aduana de acuerdo a las necesidades de esta entidad.

Como resultado de la presente investigación se obtendrá un módulo para el sistema GINA 1.1, que permita la gestión de los ingresos comerciales en la Aduana General de la República de Cuba.

Para el desarrollo del presente trabajo se hacen uso de los siguientes **métodos de investigación**:

**Métodos teóricos:** Son aquellos que permiten revelar las relaciones esenciales del objeto de investigación, son fundamentales para la comprensión de los hechos y para la formulación de la hipótesis de investigación [1, 2].

- **Análisis-Síntesis:** Mediante este método se descompone un objeto, fenómeno o proceso en los principales elementos que lo integran para analizar, valorar y conocer sus particularidades [3]. En este caso se utilizó con el objetivo de analizar teorías y documentos, y a partir de ello llegar a conclusiones acerca de las características y las necesidades vigentes para la realización de la solución que se persigue, determinando de esta manera los elementos que mejor definirían una buena solución.

**Métodos empíricos:** Estos métodos posibilitan revelar las relaciones esenciales y las características fundamentales del objeto de estudio, accesibles a la detección de la percepción, a través de procedimientos prácticos con el objeto y diversos medios de estudio [4].

- **Observación:** Consiste en la percepción directa del objeto de investigación. Permite conocer la realidad mediante la percepción directa de los objetos y fenómenos [5]. En este caso se utilizó para entender cómo se realizan los procesos relacionados con los ingresos comerciales actualmente en la AGR.
- **Revisión de documentos:** Se realizó un estudio de la documentación existente sobre el sistema GINA y las principales metodologías a utilizar.

El presente documento posee la siguiente **estructuración del contenido:**

Está compuesto por tres capítulos que dan respuesta a los objetivos y tareas planteados. Cada uno de ellos contiene los elementos necesarios para llegar en conjunto a la solución final, partiendo del estudio del problema planteado.

Durante el transcurso del **Capítulo 1** se establece la Fundamentación Teórica de la investigación. Se realiza además un estudio de las soluciones existentes que se relacionan con el problema planteado, así como de las herramientas, metodologías y lenguajes de programación a usar en el desarrollo de la solución.

En el **Capítulo 2** se describe el Diseño e Implementación de la solución, describiendo a su vez varios artefactos que se generan durante estas disciplinas.

En el **Capítulo 3** se describen los resultados de las pruebas para la validar el correcto funcionamiento de la solución implementada.

# Capítulo 1. Fundamentación teórica

Para establecer el marco teórico de la presente investigación se realizó un estudio de los temas necesarios para dar cumplimiento a los objetivos propuestos, tomando como temas principales los relacionados con el objeto de estudio y el campo de acción.

En este capítulo se presenta un estudio de algunos sistemas informáticos nacionales e internacionales para la gestión de las operaciones aduaneras, similares al que se quiere desarrollar a partir del presente trabajo. De igual forma se realiza una descripción de la metodología de desarrollo y herramientas a utilizar.

## 1.1 Sistemas de gestión aduanera

Los sistemas automatizados en las aduanas constituyen una herramienta muy útil que permite incrementar el nivel de facilitación de los procedimientos que rigen el comercio internacional. La modernización de las aduanas trae aparejado la automatización de sus procesos con el objetivo de lograr sustituir el procesamiento manual de los documentos, por el tratamiento de información transmitida por medios electrónicos [6, 7]. Esto último contribuye a lograr un mejor control en la recaudación fiscal, calidad y rapidez en la elaboración de las estadísticas del comercio exterior, obteniéndose como resultado la generación automática de datos comerciales y la reducción de la corrupción, además de reducir congestionamientos en puertos y aeropuertos [8].

### 1.1.1 Sistema Aduanero Automatizado

El Sistema Aduanero Automatizado (SIDUNEA) es una herramienta informática para el control y administración de la gestión aduanal. Este sistema fue desarrollado por la Conferencia de las Naciones Unidas sobre Comercio y Desarrollo (UNCTAD, por sus siglas en inglés); diseñado para proveer estadísticas del comercio exterior. Esta herramienta ha sido desplegada en varios países del mundo con el objetivo de mejorar los procedimientos aduanales para el control de las operaciones de comercio internacional, a través de medidas que incluyen reformas a la práctica administrativa ya existente.

SIDUNEA permite a una aduana definir la información opcional, condicional y obligatoria que considere necesaria. Emplea códigos internacionales y estándares desarrollados por la Organización Internacional para la Estandarización (ISO, por sus siglas en inglés), por la Organización Mundial de Aduanas (OMA) y por la Organización de Naciones Unidas (ONU) [6].

Características tecnológicas del SIDUNEA:

- Utiliza tecnología Java, permitiendo el uso de tarjetas inteligentes y garantizando el control de accesos al sistema y los pagos electrónicos.

El SIDUNEA cuenta con:

- Modernos conceptos de seguridad (PKI).
- Conceptos DOM (Document Object Model), XML.
- Directorios de mensajes XML, estándar que hace posible la cooperación internacional entre sistemas y la creación de la red Customs Global.
- Protocolo REWI extendido, TCP/IP.
- Interfaces de usuario amigables.
- Extensión e implementación dinámica.
- Adaptabilidad (según el número de operaciones).
- Aspectos de seguridad ya integrados.
- Funciones especiales como múltiples idiomas, gestión, propiedad de documentos y auditoría [11].

Ventajas:

1. Optimiza los tiempos y recursos del proceso aduanero.
2. Posibilita el cobro correcto de los impuestos y tasas.
3. Detecta los errores en los valores de la declaración.
4. Monitorea el pago de los impuestos.
5. Evita la evasión de impuestos.
6. Minimiza el contrabando.
7. Crea incentivos para el declarante en la Aduana.
8. Administra efectivamente el proceso de despacho.

9. Pone en práctica un esquema de garantía con la modalidad de pago anticipado, para facilitar el comercio y asegurar el cobro de los derechos aduaneros.
10. Controla la ruta de comercio por medio de las oficinas de despacho de mercancía de cada aduana.

El Módulo para Aduana se describe como el módulo central del sistema SIDUNEA, es la base de los procedimientos de control del cobro y liquidación de los impuestos, su estudio será una gran base para el desarrollo del sistema.

### 1.1.2 Sistema de Computación para el Flete Internacional de la Aduana

El Sistema de Computación para el Flete Internacional de la Aduana (SOFIA) es un sistema de despacho aduanero informatizado que interactúa en forma directa con varios tipos de usuarios: Despachantes de Aduana, Empresas de Transporte, Depositarios, Funcionarios de Aduana y con los Organismos vinculados al comercio exterior.

SOFIA toma como base y punto de partida el sistema francés SOFI (Sistema de Computación para el Flete Internacional) y luego incorpora nuevas funcionalidades para satisfacer las necesidades de gestión de la Aduana Paraguaya, conforme a la evolución impuesta por el comercio globalizado.

EL SOFIA cuenta con un módulo llamado Recaudaciones, el cual es el encargado de toda la gestión de valores: depósitos en efectivo o bancarios, créditos tributarios o aduaneros. En particular administra la:

- Percepción de valores.
- Afectaciones de valores a una liquidación aduanera.
- Generación de comprobantes de pago.
- Transferencia a las cuentas recaudadoras.
- Control de garantías.

El estudio de dicho sistema y sus características aportó conocimiento para entender mejor el funcionamiento de los sistemas aduanales en general.

### 1.1.3 Sistema de Órganos Aduaneros

En Cuba, la Aduana posee un departamento dedicado a la elaboración de software de manera conjunta con otras instituciones como la UCI, dicho departamento es conocido como CADI, el cual desarrolló y desplegó

el Sistema de Órganos Aduaneros (SOA), dicho sistema estaba compuesto por varios módulos que aunque desarrollados en diferentes plataformas, resolvían y gestionaban procesos o parte de ellos, comenzando a evidenciarse resultados satisfactorios de su empleo. Entre dichos módulos se encuentran: Sistema Automatizado de Despacho Mercantil (SADEM), Sistema Automatizado de Control Mercantil (SACOM), Sistema Automatizado de Personas de Interés Aduanal (SAPIA) y Sistema Automatizado de Despacho y Operaciones No Comerciales (SADONCE), los cuales pueden ser contemplados como módulos independientes del SOA, que resolvían casos específicos en direcciones determinadas de la Aduana. Sin embargo, no existía retroalimentación de datos entre ellos, ni una totalidad de funcionalidades dedicadas a cubrir todas las áreas de proceso de la Aduana, dificultando en ocasiones el flujo ininterrumpido de información sin errores o duplicidad.

La fragmentación de datos existente trajo como consecuencia que esta característica se convirtiera en la principal limitante del sistema, concluyéndose por tanto que era necesario agrupar en un sistema integrado todos los procesos que se llevan a cabo en la Aduana, surgiendo así el Sistema Único de Aduanas (SUA), el cual está actualmente en uso.

**Sistema Automatizado de Despacho Mercantil**

Uno de los principales módulos del SOA es SADEM, que se desplegó en la Aduana cubana en el año 2001. Este sistema es el resultado de un estudio realizado al proceso de importación y exportación. El proceso anteriormente mencionado se encontraba parcialmente informatizado por el sistema SIDUNEA. El SADEM tenía como objetivo informatizar todas las operaciones que conforman el proceso de despacho comercial, abarcando la gestión de las fases de presentación, registro, liquidación e ingresos comerciales [9], previendo su funcionamiento tanto en el momento en que se producen las operaciones como posterior al proceso y de esta manera registrar la información necesaria para las estadísticas del comercio exterior.

Para cumplir los objetivos del SADEM, la Aduana cubana realizó un conjunto de adecuaciones, como por ejemplo:

- Perfeccionar el proceso de despacho, logrando procesos más sencillos, eficientes y ágiles.
- Controlar de manera automatizada los plazos de vencimiento de las facilidades y regímenes suspensivos.

- Controlar las exenciones y bonificaciones del pago, así como las cuotas, contingentes arancelarios, eventualidades de nomenclatura y acuerdos, teniendo de esta forma muy bien delimitado el sacrificio fiscal.
- Realizar los cálculos de los ingresos en frontera por concepto de derechos de aduana y tasas de servicio.
- Resolver el problema de la doble moneda, teniendo bien delimitados y validados los ingresos por cada tipo de moneda en que se debe pagar en dependencia del impuesto, la partida arancelaria y el tipo de entidad, ya sea importadora o exportadora; permitiendo enfrentar el fraude de la política de comercio exterior.

El despliegue del sistema produjo cambios radicales pues fue necesario elaborar un nuevo modelo de la Declaración de Mercancía (DM), documento principal del proceso y reelaborar la metodología interna de trabajo para el despacho de mercancías. Para realizar los cambios a la DM se tuvo en cuenta los requisitos establecidos en el convenio de Kyoto en cuanto a determinados escaques[2].

Entre los cambios que introdujo el sistema se pueden citar [8, 10]:

- Posibilidad de realizar DM que pague impuestos con monedas diferentes.
- Cambio en la codificación de los regímenes.
- Informatización de todos los regímenes y tipos de declaración.
- Intercambio electrónico de datos entre aduanas y con usuarios externos para la obtención de información, como por ejemplo: manifiesto, proveedores, exenciones, entre otros.
- Ingresos por todos los conceptos y formas de pago.

El SADEM cuenta con 7 módulos los cuales son: Tablas de Control, Selectividad, Manifiesto, Ingresos, Almacenes, Administración y Declaración de Mercancías.

### 1.1.4 Sistema Único de Aduana

El Sistema Único de Aduana (SUA) es una herramienta desarrollada por un grupo de especialistas del CADI en conjunto con la UCI. Fue desplegado en la Aduana cubana en el año 2008 con el propósito de resolver

[2] Escaque: campo que conforma el formulario de la declaración de mercancías en el cual se registra información de interés para la aduana.

todas aquellas limitantes que presentaba el SOA. Para el desarrollo de este sistema fue preciso tener en cuenta lo referido en el capítulo 7 del convenio de Kyoto Revisado, el cual aborda el uso y aplicación de las tecnologías de la información a fin de respaldar las operaciones aduaneras [9].

El SUA plantea como objetivo principal informatizar el procesamiento informativo referente a todas las operaciones que conforman los diferentes procesos aduanales, con una base de datos única y centralizada, que permite la interacción con todos los procesos dentro de la Aduana.

Este sistema está actualmente desplegado en todas las aduanas de Cuba, teniendo bien delimitadas todas las necesidades que para cada caso se especifican; ya sean los números de régimen aduaneros existentes, las medidas de facilitación que se aplican, así como las operaciones de comercio exterior que se realizan. El SUA cuenta con 23 módulos de los que se pueden citar como los principales: Despacho de Medios de Transporte Internacional (Despacho MTI), Despacho No Comercial, Solicitudes y Despacho Comercial, siendo este último el más importante dentro de la Aduana. El Despacho Comercial se encarga de todo el procesamiento de la DM, ya sea de importación o de exportación; así como de todos los tipos de facilidades y regímenes aduaneros. Todas aquellas operaciones que son tramitadas ante la Aduana se realizan utilizando el mismo modelo.

Este sistema cuenta con un módulo para la gestión del cobro de servicios y aranceles aduaneros, el cual ha funcionado correctamente hasta el momento. Actualmente la AGR está migrando sus servicios para el sistema GINA, debido a las mejoras que este brinda en cuanto a configuración y administración, además de interfaces de usuarios más amigables. En la nueva versión de sistema GINA, 1.1, se desarrollará el subsistema Despacho Comercial, para el cual se necesita, para su completo funcionamiento, de un módulo que gestione los ingresos comerciales, debido a las grandes dependencias que tiene despacho con el área de ingresos.

## 1.2 Modelo de desarrollo de software

Para llevar a cabo el desarrollo de un proyecto de software con la calidad requerida, se hace necesario el uso de una metodología que se encargue de definir quién debe hacer qué, cuándo y cómo debe hacerlo. Una metodología debe definir con precisión los artefactos, roles y actividades involucradas, junto con prácticas y técnicas recomendadas [11].

La presente investigación se desarrolla dentro del Departamento de Soluciones para la Aduana, por lo cual se ajusta a su modelo de desarrollo de software. A continuación se exponen los elementos fundamentales de dicho modelo.

El modelo de desarrollo de software que se utiliza en el Departamento de Soluciones para la Aduana es el definido para el Centro de Gestión de Entidades (CEIGE).

**Metodología de desarrollo del CEIGE v1.2**

Para guiar el proceso de desarrollo de software, el CEIGE definió su propio Modelo de Desarrollo de software en su versión 1.2, en la cual se detalla el ciclo de vida de los proyectos, con la incorporación de los distintos subprocesos dictados por el Nivel II de CMMI[3].

Para el ciclo de vida de los proyectos el centro tiene en cuenta las fases y actividades por área de procesos que plantea el Nivel II de CMMI establecido en la UCI. Abarca el total de acciones que se realizan en las distintas líneas de desarrollo para la elaboración del servicio o producto final, sin embargo, se debe adaptar a las características particulares del proyecto que puede no ejecute determinada fase, así como la elaboración de determinados artefactos del total definido [12]. En la Figura 1 se pueden apreciar las disciplinas por las que pueden transitar los proyectos del centro.

El modelo seleccionado establece las diferentes fases por las que se debe transitar durante el desarrollo de un producto de software y los distintos artefactos que se generan en cada una de ellas. Las fases definidas son: Inicio o Estudio preliminar y Desarrollo. La solución que se desea implementar se enmarca en la fase de Desarrollo, que tiene como objetivo obtener un sistema que satisfaga las necesidades de los clientes y usuarios finales, y en dicha fase se parte de la disciplina Análisis y diseño, ya que anteriormente se realizaron las de Modelado del Negocio y Requisitos.

Durante la disciplina **Análisis y diseño** se modela el sistema, de forma tal que satisfaga todos los requisitos definidos. Se generan artefactos más orientados a la implementación, estos son: el modelo de diseño, artefacto conformado por los diagramas de clases del diseño con estereotipos web, los diagramas de secuencia, el modelo de datos y los diseños de casos de prueba. En la disciplina **Implementación**, a partir de los resultados de la disciplina Análisis y diseño se implementa el sistema en términos de componentes,

[3]CMMI: por sus siglas en inglés Capability Maturity Model Integration.

es decir, ficheros de código fuente, scripts y ejecutables. Es donde se definen los estándares de codificación a utilizar y se implementan los requisitos funcionales identificados con sus interfaces correspondientes.

En la disciplina **Pruebas internas** se identifican posibles errores en la documentación y el software, es decir requisitos que el producto debería cumplir y que aún no los cumple. En esta etapa se realizan las pruebas de caja blanca y pruebas de caja negra. Además, se resuelven las no conformidades detectadas durante la ejecución de estas pruebas.


**Figura 1: Ciclo de vida de los proyectos del CEIGE. Tomada de [12].**

## 1.3 Herramientas y tecnologías para el desarrollo

La elección correcta de las herramientas y tecnologías a utilizar en el desarrollo de soluciones informáticas influye en la calidad del producto. El departamento de Soluciones para la Aduana ha definido cuáles son las tecnologías y herramientas a emplear para el desarrollo de la solución. El módulo Ingresos Comerciales será implementado haciendo uso de las tecnologías y herramientas definidas para el sistema GINA.

### 1.3.1 Lenguaje para el modelado

El lenguaje de modelado que se utiliza es UML[4], ya que permite especificar, visualizar y construir los artefactos de los sistemas de software [13]. Este lenguaje es un estándar elemental para construir modelos orientados a objetos. Posibilita la corrección de errores en todas las etapas del desarrollo del software. UML permite visualizar de forma gráfica los detalles técnicos de un sistema de forma tal que otro desarrollador lo pueda entender fácilmente, además de especificar cuáles son las características de un sistema antes de su construcción [14].

### 1.3.2 Lenguaje de programación del lado del servidor

Se utiliza PHP[5] 5.3.8 por ser un lenguaje de código abierto, interpretado, de alto nivel, embebido en páginas HTML (del inglés Hyper Text Markup Language) y ejecutado en el servidor [15]. Mediante este lenguaje el cliente solamente recibe el resultado de su ejecución en el servidor, sin ninguna posibilidad de determinar qué código ha producido el resultado recibido [16].

Características de PHP:

- Gratuito. Al tratarse de software libre, puede descargarse y utilizarse en cualquier aplicación, personal o profesional, de manera completamente libre.
- Versatilidad. PHP puede usarse con la mayoría de sistemas operativos, ya sea basados en UNIX (Linux, Solares, FreeBSD), como con Windows.
- Eficiencia. Con escaso mantenimiento y un servidor gratuito, puede soportar sin problema un gran número de visitas diarias.
- Sencilla integración con múltiples bases de datos.

### 1.3.3 Herramienta de modelado

Se utiliza como herramienta de modelado el Visual Paradigm 8.0 por soportar múltiples plataformas, presentar una interfaz de uso intuitivo y con muchas facilidades para modelar los diagramas; además de su robustez, usabilidad y portabilidad [17]. Brinda la posibilidad de realizar ingeniería inversa y flexibilidad para integrarse con los principales entornos de desarrollo como el Eclipse y el NetBeans [18].

[4] UML: Unified Modeling Languaje, por sus siglas en inglés.
[5] Acrónimo de PHP: Hypertext Preprocessor, por sus siglas en inglés,

Algunas características de Visual Paradigm:

- Es profesional.
- Es amigable.
- Sincronización entre diagramas de entidad-relación y diagramas de clases.
- Generación de código / Ingeniería inversa.
- Soporte de UML versión 2.1.
- Interoperabilidad con otras aplicaciones.

### 1.3.4 Framework Symfony

Symfony 1.2.8 es un framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Se caracteriza por separar la lógica de negocio, la lógica de servidor y la presentación de la aplicación web [19]. Además proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. Es compatible con la mayoría de gestores de bases de datos como Oracle, MySQL, PostgreSQL y SQL Server de Microsoft. Se puede ejecutar en varias plataformas (Unix, Linux, Windows, etc.) [20].

### 1.3.5 Framework ExtJS

ExtJS 3.0 es un framework de interfaz de usuario ligero y de alto rendimiento, compatible con la mayoría de navegadores que permite crear páginas e interfaces web dinámicas [21]. Permite realizar aplicaciones web enriquecidas basándose en tecnología AJAX, JSON, DHTML y DOM. Con ExtJS, se pueden desarrollar aplicaciones web multiplataforma con facilidad.

A continuación se mencionan algunas ventajas de ExtJS [22]:

- El modelo de componente de ExtJS mantiene su código bien estructurado posibilitando que aplicaciones web complejas pueden ser de fácil mantenimiento.
- Ofrece la posibilidad de utilizar un gran número de componentes visuales que mejoran considerablemente la calidad de las aplicaciones.
- Brinda la posibilidad de validar de formularios, basándose en expresiones regulares y tipos de datos.

- Trae implícitos componentes como vista en árboles, arrastrado y soltado, cambio de tamaño de imágenes, rejillas, paginado, agrupado de objetos, asistentes, entre otros.
- Su utilización facilita la separación de las capas de la vista con la del controlador desde el punto de vista productivo ya que el código utilizado en la primera es solamente JavaScript y no es necesario utilizar ningún tipo de código PHP, así los desarrolladores pueden centrarse más en el aprendizaje de un solo lenguaje.

### 1.3.6 Entorno de Desarrollo Integrado

El Entorno de Desarrollo Integrado (IDE[6]) NetBeans 7.2 es un entorno desarrollo integrado disponible para Windows, Mac, Linux y Solaris. Es de código abierto y a su vez una plataforma de aplicaciones que permiten a los desarrolladores crear aplicaciones web, empresariales, de escritorio y aplicaciones móviles utilizando la plataforma Java, así como PHP, JavaScript, Ajax, Groovy y Grails, y C/C ++ [23, 24].

### 1.3.7 Sistema Gestor de Base de Datos Oracle Database 11g

El Sistema Gestor de Base de Datos (SGBD) Oracle Database 11g proporciona nuevas e innovadoras funcionalidades que garantizan alto rendimiento, alta escalabilidad, fiabilidad y seguridad mediante el uso de plataformas grid, asegurando altos niveles de calidad de servicio e incrementos de la flexibilidad de negocio reduciendo además los costes de explotación [25, 26].

Incluye funcionalidades que permiten hacer pruebas de cambios en aplicaciones simulando las cargas reales generadas por los usuarios en los entornos de producción. Permite reducir los tiempos, riesgos y costos derivados de la implantación de cambios. Responde de manera más efectiva a los requerimientos variables del negocio y hace una gestión de cambio más segura [27].

A continuación se mencionan algunas ventajas y desventajas del SGBD Oracle Database 11g [25]:

- Las entidades complejas del mundo real y la lógica se pueden modelar fácilmente, lo que permite reutilizar objetos para el desarrollo de base de datos de una forma más rápida y con mayor eficiencia.
- Los programadores de aplicaciones pueden acceder directamente a tipos de objetos Oracle, sin necesidad de ninguna capa adicional entre la base de datos y la capa cliente.

[6]IDE: Integrated Development Environment, por sus siglas en inglés.

- Las aplicaciones que utilizan objetos de Oracle son fáciles de entender y mantener, ya que soportan las características del paradigma orientado a objetos.
- Posee un rico diccionario de datos.
- Brinda soporte a la mayoría de los lenguajes de programación.
- Es un sistema multiplataforma, disponible en Windows, Linux y Unix.
- Permite tener copias de la base de datos en lugares lejanos a la ubicación principal. Las copias de la base de datos pueden estar en modo de lectura solamente.

Desventajas:

- Es un producto de elevado precio por lo que por lo general se utiliza en empresas muy grandes y multinacionales.
- Los costos de soporte técnico y mantenimiento son elevados.
- Vulnerabilidades en la seguridad de la plataforma, se hace necesario aplicar parches de seguridad.

Es la única herramienta privativa que se ha ganado un lugar en la AGR debido a sus insustituibles características. Ha estado presente desde 1997 en la Aduana siempre brindando buenos resultados.

### 1.3.8 Petición AJAX

Ajax es una técnica de desarrollo web para crear aplicaciones interactivas, las cuales se ejecutan en el cliente, mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas.

Ajax permite que las páginas web respondan de forma más rápida mediante el intercambio en segundo plano de pequeñas cantidades de datos con el servidor, por lo que no es necesario recargar la página entera cada vez que el usuario realiza un cambio. El objetivo es aumentar la interactividad y la rapidez de la página [28].

Ajax depende de XML Http Request, un objeto JavaScript cuyo comportamiento es similar a un marco oculto, cuyo contenido se puede actualizar realizando una petición al servidor y se puede utilizar para manipular el resto de la página web. Se trata de un objeto a muy bajo nivel, por lo que los navegadores lo tratan de forma diferente y el resultado es que se necesitan muchas líneas de código para realizar peticiones Ajax.

Afortunadamente, Prototype encapsula todo el código necesario para trabajar con Ajax y proporciona un objeto Ajax mucho más simple y que también utiliza Symfony [20].

Las interacciones de Ajax están formadas por tres partes: el elemento que la invoca (un enlace, un formulario, un botón, un contador de tiempo o cualquier otro elemento que el usuario manipula e invoca la acción), la acción del servidor y una zona de la página en la que mostrar la respuesta de la acción [29].

La Figura 2 muestra un diagrama de ejemplo de peticiones AJAX donde el cliente envía la petición al servidor, este recibe los datos y envía la respuesta al cliente ya sea satisfactoria o no, todo este proceso lo realiza mediante el JSon que es el encargado del intercambio de datos entre la página cliente y la servidora.


**Figura 2: Diagrama de Peticiones AJAX.**

## 1.4 Patrones empleados por Symfony

### 1.4.1 Patrón arquitectónico

Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa la lógica de negocio de la interfaz de usuario, facilita la evolución por separado de ambos aspectos e incrementa la reutilización y la flexibilidad del sistema [30].

- El modelo son las operaciones que se efectúan sobre los datos que se reciben, o las consultas a BD que se hacen, su objetivo es preparar los datos para que la vista solo se tenga que preocupar de mostrarlo, así si hay que realizar cambios en la estructura de la BD o cambiar las operaciones que se hacen con los datos se tendría que hacer sólo en el modelo, evitando que un cambio se tenga que realizar en todas las vistas que utilicen unos determinados datos.
- Las vistas son el código HTML, lo que se muestra al usuario. Estas muestran los datos que vienen por el servidor sin saber realmente lo que se está mostrando, puesto que el modelo es el que ha introducido los datos en ella y pueden variar dependiendo de las circunstancias.

- El controlador es el que escucha los cambios en la vista y se los envía al modelo, el cual le regresa los datos a la vista, es el que decide qué vista se debe de imprimir y qué información es la que se envía.

### 1.4.2 Patrones de diseño

Los patrones son soluciones simples y elegantes a problemas específicos y comunes del diseño Orientado a Objetos basados en la experiencia, estos permiten reutilizar la experiencia de los desarrolladores, clasificar y describir formas de solucionar problemas que ocurren de forma frecuente en el desarrollo y está basado en la recopilación del conocimiento de los expertos en desarrollo de software.

**Patrones GoF**

Patrones GoF[7]: describen las clases y objetos que se comunican entre sí y se adaptan para resolver un problema general de diseño en un contexto particular. Estos vienen agrupados por los patrones creacionales para abstraer el proceso de realizar instancias y crear objetos, los patrones estructurales, que describen como pueden ser combinados las clases y los objetos para formar grandes estructuras y por último los patrones de comportamiento para definir la comunicación entre los objetos del sistema [31, 32].

- Singlenton (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia, Symfony lo utiliza para crear un contexto único, donde guarda las configuraciones del proyecto (sfcontext\frontend-dev.php) y el controlador frontal (sfWebFrontController) se encarga de enrutar todas las peticiones que se hagan a la aplicación.
- Command (Comando): Encapsula una cierta cantidad de funcionalidades en una sola estructura haciendo este funcionamiento oculto para el usuario. Este patrón se observa en la clase sfWebFrontController, en el método dispatch. Esta clase está por defecto y es la encargada de establecer el módulo y la acción que se va a usar según la petición del usuario.
- Decorator (Decorador): Este método pertenece a la clase abstracta sfView, padre de todas las vistas, que contienen un decorador para permitir agregar funcionalidades dinámicamente, el archivo nombrado layout.php es el que contiene el layout de la página, este conocido también como plantilla

[7]GoF: (Gang of Four) llamados así por los cuatro autores del libro: “Patrones de Diseño”.

global, guarda el código HTML, que es usual en todas las páginas del sistema, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el layout.

**Patrones GRASP**

Los patrones GRASP[8] (del inglés General Responsibility Asignment Software Patterns) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Constituyen el fundamento de cómo se diseñará el sistema [32].

- Creador: Ayuda a identificar quién debe ser el responsable de la creación de nuevos objetos o clases, donde la nueva clase deberá ser creada por la clase que tiene toda la información necesaria para realizar la acción, usando directamente las instancias creadas del objeto, almacena o maneja varias instancias de clase y contiene o agrega la clase. En la clase Actions se encuentran las acciones definidas para el sistema. En dichas acciones se crean los objetos de las clases que representan las entidades, lo que evidencia que la clase Actions es "creador" de dichas entidades.
- Bajo acoplamiento: La clase Actions hereda únicamente de sfActions para alcanzar un bajo acoplamiento de clases. Las clases que implementan la lógica del negocio y de acceso a datos se encuentran en el modelo, las cuales no tienen asociaciones con las de la vista, lo que proporciona que la dependencia en este caso sea baja.
- Alta cohesión: Symfony permite la organización del trabajo en cuanto a la estructura del proyecto y la asignación de responsabilidades con una alta cohesión. Un ejemplo de ello es la clase Actions, la cual está formada por varias funcionalidades que están estrechamente relacionadas, siendo la misma la responsable de definir las acciones para las plantillas y colaborar con otras para realizar diferentes operaciones, instanciar objetos y acceder a las propiedades, además se usa ya que cada elemento del diseño debe realizar una labor única dentro del sistema, con el objetivo de que las clases con responsabilidades estrechamente relacionadas no se complejicen.
- Experto: Es uno de los patrones que más se utiliza cuando se trabaja con Symfony, con la inclusión de la biblioteca de clases Propel para mapear la BD. Symfony utiliza esta biblioteca para realizar su capa de abstracción en el modelo, encapsular toda la lógica de los datos y generar las clases con todas las funcionalidades comunes de las entidades, las clases de abstracción de datos (Peer del

[8] GRASP: General Responsibility Asignment Software Patterns, por sus siglas en inglés.

Modelo) poseen un grupo de funcionalidades que están relacionadas directamente con la entidad que representan y contienen la información necesaria de la tabla correspondiente.

- Controlador: Todas las peticiones Web son manipuladas por un solo controlador frontal (sfActions), que es el punto de entrada único de toda la aplicación en un entorno determinado. Este patrón se evidencia en las clases sfFrontController, sfWebFrontController, sfContex y en los “actions”.

El uso de estos patrones garantizó asignar a cada clase la responsabilidad que le corresponde, obtener el menor número de relaciones y dependencias entre clases, aumentando las posibilidades de reutilización de las mismas. Todos estos elementos ayudaron a la confección de un diseño de la solución con claridad y rendimiento.

## 1.5 Conclusiones del capítulo

A partir del estudio realizado de los sistemas de automatización para las aduanas nacionales e internacionales relacionados con la gestión de los despachos aduaneros, se concluye que:

- Existe una necesidad de crear un nuevo sistema para gestionar los ingresos comerciales que se integre al sistema GINA.
- Para el desarrollo del módulo se empleará el Modelo de Desarrollo del Centro CEIGE y la herramienta CASE Visual Paradigm para la descripción del mismo.
- Como sistema gestor de base de datos se utilizará Oracle 11g.
- La implementación del sistema se hará empleando el IDE NetBeans.
- Como servidor de aplicaciones Apache.
- Para el modelado de las clases se utilizará como lenguaje servidor PHP, haciendo uso del framework Symfony en su versión 1.2.8 y como lenguaje cliente JavaScrip, ExtJS 3.0.

# Capítulo 2: Diseño e implementación

Para desarrollar un sistema informático es importante la utilización de métodos y técnicas que permitan la solución de problemas existentes a lo largo del ciclo de vida de un software. La correcta captura de requisitos y el modelado del sistema permiten que se mitiguen las fallas que puedan aparecer durante su desarrollo, además de establecer un entendimiento común entre lo que se desea y lo que se elabora.

En el presente capítulo se parte de los resultados obtenidos de las disciplinas Modelado del Negocio y Requisito. Se listan los requisitos funcionales con que contará el módulo. Se muestran, además, los diferentes artefactos generados durante la disciplina Análisis y Diseño, definiendo el estilo y patrón arquitectónico. Se describe el modelo de base de datos utilizado en el desarrollo del módulo que se propone. Luego se valida el diseño propuesto. Finalmente, se muestra el modelo de implementación, el cual define la organización del código y cómo implementar los elementos de diseño.

## 2.1 Modelo del Negocio

En la Figura 3 se muestra el diagrama del proceso general del módulo Ingresos Comerciales, artefacto resultante de la disciplina Modelado de Negocio, elaborado por analistas del proyecto GINA, utilizado como base para entender los conceptos y procesos para luego realizar la disciplina Requisito.

En la AGR, los procesos de ingresos comerciales juegan un papel importante, son los encargados de efectuar el cobro por los distintos servicios prestados a sus clientes.

Para efectuar el cobro, primero se verifica si los documentos implicados están o no sujetos a acuerdos de aplazamiento, seguidamente se procede a efectuarse un proceso de verificaciones a estos documentos, de acuerdo a su estado.

Aunque el principal proceso de ingresos comerciales sea efectuar el cobro de estos documentos, también brinda la posibilidad de realizar a acuerdos de aplazamiento, dando posibilidades a clientes que no pueden efectuar el pago por estos servicios de una sola vez, que lo hagan en varios plazos, pagando montos fijos, de acuerdo a las posibilidades que estos tengan en sus ingresos, brindando a la AGR la seguridad del cobro de estos documentos. Permite además confirmar los acuerdos de aplazamiento, modificar los ya existentes o cancelar estos acuerdos, en caso de que el cliente falle con la fecha establecida para efectuar el pago del

plazo del acuerdo. Permite también realizar al cobro de un servicio mediante la facturación y cancelar alguna de estas facturadas efectuada erróneamente. Además poner en vía de apremio aquellos documentos que el cliente este negado a pagar, para que sean atendidos por una vía legal y a su vez cobrar estos documentos después de ser entregado por los especialistas que llevan estos casos específicos.

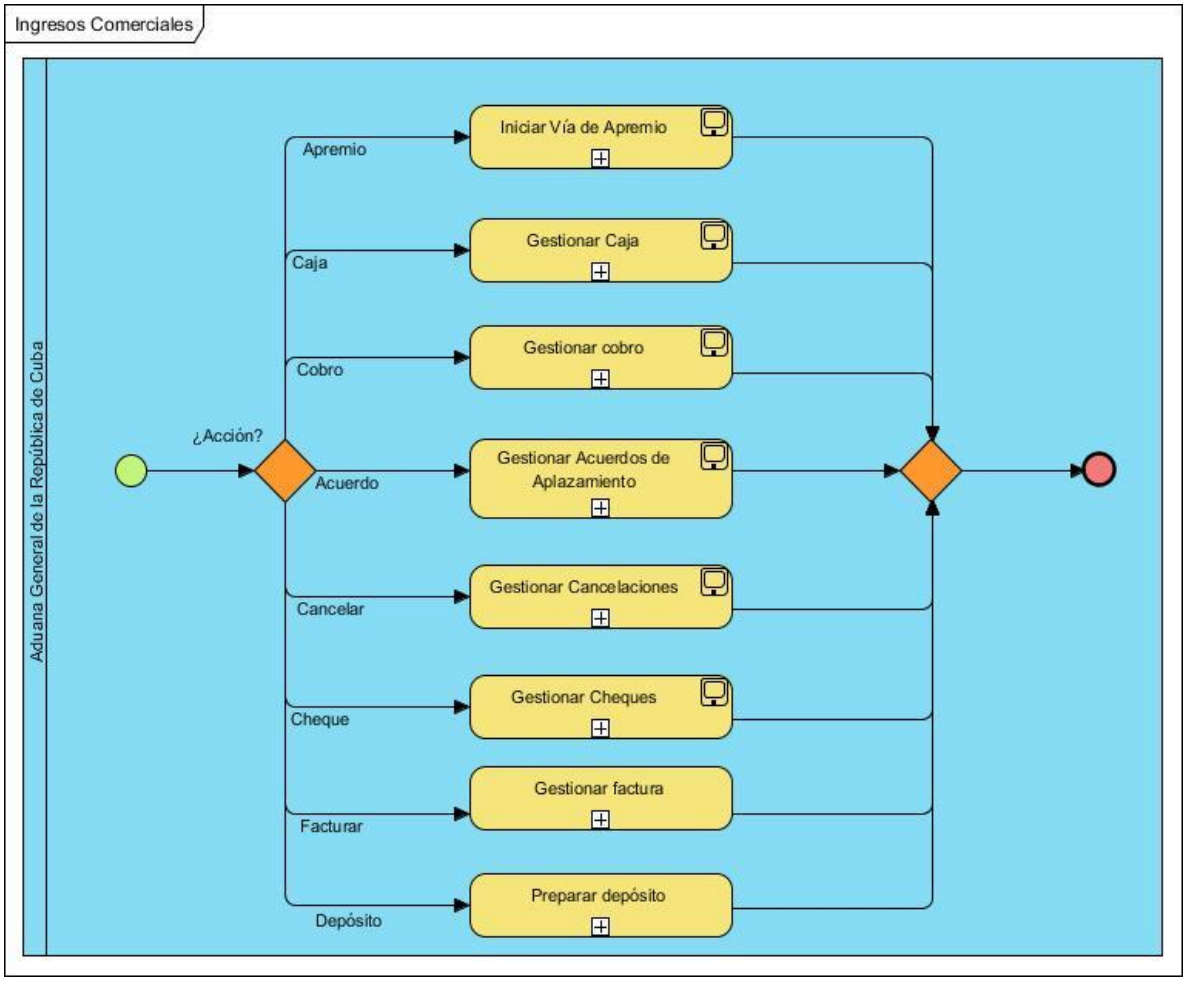


**Figura 3: Diagrama del proceso general de Ingresos Comerciales.**

## 2.2 Requisitos

La obtención de requisitos es el proceso mediante el cual los interesados en un sistema de software descubren, revelan, articulan y entienden sus requisitos. En muchos casos, se requiere tiempo para llegar a especificar claramente lo que el interesado espera de la aplicación de software, por lo que se hace

necesario por parte de los analistas el empleo de técnicas que permitan establecer una buena comunicación con los interesados del producto y así lograr la satisfacción del cliente.

A continuación se enuncian las principales técnicas utilizadas durante el proceso de desarrollo para recopilar los requisitos de software.

- **Entrevista:** Se consultaron a personas que trabajaron en la implementación del módulo Ingresos Comerciales en el SUA y especialistas de la AGR que trabajan en esta área.
- **Observación:** Se llevó a cabo para percibir cómo se desarrolla el control y el cobro de los servicios y aranceles en la Aduana, pudiendo captar directamente las particularidades de estos procesos en las visitas realizadas a la Aduana.

### 2.2.1 Requisitos funcionales

La mayoría de los sistemas de software tienen un número elevado de requisitos, por lo que el desglose del mismo lo convierte en un poco más sencillo en el momento de su desarrollo.

Durante la disciplina de captura de requisitos los analistas del proyecto identificaron 19 requisitos funcionales con los que debe contar la aplicación para satisfacer las necesidades requeridas por el cliente. A continuación se muestra el listado de los mismos.

- RF 1: Gestionar Acuerdo de Aplazamiento
    - RF 1.1: Insertar Acuerdo de Aplazamiento: Permitirle al cliente llegar a un acuerdo con la Aduana al no poder pagar la deuda en el tiempo establecido.
    - RF 1.2: Modificar Acuerdo de Aplazamiento: Permitirle al cliente cambiar los plazos establecidos en el Acuerdo de Aplazamiento.
    - RF 1.3: Confirmar Acuerdo de Aplazamiento: Permitir confirmar un Acuerdo de Aplazamiento, lo que pondría al cobro el acuerdo.
    - RF 1.4: Cancelar Acuerdo de Aplazamiento: Permitir cancelar un Acuerdo de Aplazamiento ya que el cliente no cumplió con los términos bajo los cuales se emitió el acuerdo o petición del mismo.
- RF 2: Gestionar vía de apremio

  - RF 2.1: Iniciar vía de apremio: Permitir poner en vía de apremio los documentos que el cliente no quiera pagar.
  - RF 2.2: Cobrar Vía de Apremio: Permitirle a la Aduana cobrar los documentos que se encuentren en Vía de Apremio.
- RF 3: Gestionar caja
  - RF 3.1: Arquear caja: Permitir mantener un control de los instrumentos de pago y efectivo existente en caja.
  - RF 3.2: Cerrar caja: Permitir marcar la caja como cerrada.
  - RF 3.3: Abrir caja: Permite marcar la caja como abierta, es llamado por los procesos Cobrar Documento, Cobrar Cheque Devuelto, Iniciar Vía de Apremio y Acuerdo de Aplazamiento.
- RF 4: Cobrar documento: Permitir el pago de los servicios prestados, las multas, aranceles y recargos por mora aplicados.
- RF 5: Cancelar nota de crédito: Cuando se entrega el expediente necesitado, el especialista busca la nota de crédito y procede a cancelarla.
- RF 6: Gestionar mora
  - RF 6.1: Aplicar mora: Permitir aplicarle una mora a los documentos que se encuentren fuera de fecha al efectuarse el pago de los mismos o incorporarlos a un acuerdo de aplazamiento, así como al iniciársele un apremio, es llamado por los procesos Cobrar Documento, Cobrar Cheque Devuelto, Iniciar Vía de Apremio y Acuerdo de Aplazamiento.
  - RF 6.2: Cancelar mora: Permitir al cliente cancelar una mora que se le haya aplicado a algún documento por error.
- RF 7: Cobrar Acuerdo de Aplazamiento: Permitir que el cliente pague los documentos que tiene en Acuerdo de Aplazamiento.
- RF 8: Cobrar cheque devuelto: Permitir que el cliente pague un cheque utilizado en un cobro anterior que le fue devuelto por tener algún error.
- RF 9: Cancelar servicios: Permitir al cliente cancelar los servicios que les fueron prestados por la Aduana.
- RF 10: Preparar déposito: Permitir al cajero visualizar los recibos de pago pendientes por depositar, y prepararlos para ser depositados en el banco.
- RF 11: Ajustar factura: Permitir que el cliente obtenga una factura con los servicios prestados y derecho de tonelajes.

- RF 12: Cancelar factura: Permitir cancelar una factura errónea.

Con el propósito de lograr un mayor entendimiento de los requisitos, se realizó una descripción de cada uno. Para consultar estas descripciones, buscar en el Expediente de Proyecto GINA 1.1, Ingresos Comerciales.

### 2.2.2 Requisitos no funcionales

Los requisitos no funcionales para el módulo Ingresos Comerciales son los definidos en el Departamento de Soluciones para la Aduana para el sistema GINA, ya que este módulo va a formar parte de este. Estos se enuncian a continuación.

**Usabilidad**

Los sistemas desarrollados deben prestar facilidades de usabilidad que satisfagan las necesidades de todos los usuarios.

- Estos deben contar con un menú que les permita a los usuarios acceder a las principales funciones que son de su interés.
- Brindar a los usuarios la posibilidad de interactuar con los diferentes productos sin tener previa preparación, solo con los conocimientos necesarios del negocio.
- La resolución de la página se adaptará la resolución de pantalla en cada cliente.

**Fiabilidad**

Disponibilidad: Las aplicaciones deben ser capaces de estar operativas durante el mayor tiempo posible para brindar sus servicios a los usuarios ininterrumpidamente durante el tiempo que estos lo necesiten.

**Eficiencia**

- Tiempo de respuesta por transacción: 1.09 segundos.
- Rendimiento: 1000 transacciones por segundos, cantidad de datos que pueden ser transferidos.
- Capacidad de 80 clientes que pueden estar conectados.
- Utilización de recursos: 18 % memoria y 20 % en disco duro.

**Soporte**

- Las aplicaciones clientes deben ser capaces de correr sobre cualquier plataforma, para el caso de Windows se recomienda XP por la experiencia acumulada por los usuarios. Para la parte servidora se recomienda que corra sobre plataforma Linux.
- Ser programado en PHP 5.x o superior y con un gestor de base de datos Oracle 8 o superior o PostgreSQL 8.x.
- El sistema debe poseer una alta seguridad.

**Restricciones de diseño**

Los sistemas realizados para los procesos aduanales incluyen el manejo de varios asuntos que se enlazan entre sí, por lo que constantemente se necesita acceder a información común para varios de estos procesos, es aquí donde entra la reutilización de dicha información, de forma tal que los datos no aparezcan duplicados.

### 2.2.3 Validación de requisitos

La validación de requisitos tiene como objetivo demostrar que con la definición y especificación obtenida de los mismos, se puede obtener realmente el sistema que el usuario necesita y que el cliente desea. El proceso de validación siempre debe realizarse, pues ayuda a mitigar el riesgo de implementar una mala especificación [33].

Esta actividad fue realizada por los analistas del proyecto GINA. Para la validación de los requisitos fueron aplicadas las siguientes técnicas:

**Construcción de prototipos:** Con el desarrollo de los prototipos se definen los objetivos globales del software, se identifican todos los requisitos y se señalan áreas en las que será necesaria la profundización en las definiciones de requisitos. Todo esto conlleva a obtener un diseño rápido del sistema, centrándose en los aspectos del software que serán visibles para el usuario. En este caso los prototipos a evaluar serán maquetas no operativas o especificaciones formales que un grupo de expertos deberán evaluar [11].

**Revisión del documento de descripción de RF:** El equipo de desarrollo debe explicar a los clientes las implicaciones de cada requisito. Verificar el cumplimiento de los estándares del cliente y la facilidad de lectura de la documentación. Proponer mejoras, agregados y/o estándares nuevos. Como resultado de las mismas se obtiene un documento que contiene la lista de defectos localizados y una lista de acciones recomendadas [34].

**Casos de prueba:** Se diseñaron casos de pruebas para cada uno de los requisitos especificados, los cuales permitirán verificar el cumplimiento de los mismos. Se describieron tanto los datos de entrada como las tareas a realizar y los resultados esperados.

### 2.2.4 Priorización de requisitos

Luego de haber identificado los requisitos funcionales del módulo, se realizó la priorización de los requisitos para determinar cuáles por su complejidad y dependencia con otros, debían implementarse primero.

La clasificación de la complejidad en Alta, Media o Baja, permite estimar el esfuerzo de implementación del requisito y contribuye a la decisión sobre la inclusión en las etapas de desarrollo del software. La priorización de los requisitos identificados para el módulo Ingresos Comerciales se realizó mediante la plantilla Evaluación de requisitos (Consultar el Expediente de Proyecto GINA 1.1, Ingresos Comerciales), teniendo en cuenta los criterios definidos para determinar la complejidad de los requisitos:

- Diferentes comportamientos.
- Formas de inicialización.
- Consultas a fuentes de almacenamientos.
- Restricciones de validación.
- Grado de reutilización.
- Lógica de negocio.

La Tabla 1 muestra los resultados arrojados de la evaluación de los requisitos de acuerdo a los criterios de complejidad a los que se hacían referencia anteriormente.

**Tabla 1: Resultados de la complejidad**

| **Resumen** | |
|---|---|
| Cantidad de requisitos con complejidad Alta | 1 |
| Cantidad de requisitos con complejidad Media | 14 |
| Cantidad de requisitos con complejidad Baja | 4 |

Luego de haber realizado la evaluación de los requisitos de acuerdo a los criterios de complejidad antes expuestos, se muestra que existe un requisito de complejidad alta, el RF 4 Cobrar documento, ya que este es un requisito que presenta un número elevado de restricciones de validaciones, una alta complejidad de interacción con la base de datos y de implementación para la lógica del negocio, 14 con complejidad media y 4 con complejidad baja.

## 2.3 Modelo del diseño

La disciplina de diseño expande y detalla los modelos de análisis tomando en cuenta todas las implicaciones y restricciones técnicas. El propósito del diseño es especificar una solución que trabaje y pueda ser fácilmente convertida en código fuente y construir una arquitectura simple y fácilmente extensible.

Durante esta disciplina se modela el sistema de forma tal que soporte todos los requisitos. Esto contribuye a una arquitectura sólida y estable que se convierte en una guía para la disciplina Implementación. Los artefactos que se generan en esta disciplina son más formales y específicos de una implementación.

El diseño permite materializar con precisión los requisitos del cliente. El proceso de diseño es un conjunto de pasos repetitivos que permiten al diseñador describir todos los aspectos del sistema a construir [35].

### 2.3.1 Diagrama de Paquetes

Un diagrama de paquetes muestra cómo un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema [36, 37].

Los paquetes están organizados para maximizar la coherencia interna dentro de cada uno y minimizar su acoplamiento externo. Cada paquete puede asignarse a un individuo o a un equipo y las dependencias entre ellos pueden indicar el orden de desarrollo requerido [38].

En la Figura 4 se muestra el diagrama de paquetes donde se expone cómo interactúa el módulo Ingresos Comerciales con las Tablas de Control (TC), las bibliotecas de Symfony y paquetes del mismo.

Dentro de los subsistemas que se relacionan con el módulo Ingresos Comerciales se encuentra el subsistema TC que es el encargado de administrar las clases que son nomencladoras y comunes para todos los subsistemas del GINA. El subsistema Persona brinda toda la información referente al personal de

la AGR. El subsistema Registro Central (RC) es el que gestiona la información de los agentes externos de la Aduana. El subsistema Administración es encargado de la seguridad, los subsistemas Despacho MTI y Despacho Comercial utilizan el módulo Ingresos Comerciales para realizar sus cobros.

Dentro del mismo módulo se materializa el vínculo con diferentes paquetes dentro de los que sobresale el paquete WEB encargado de la capa de presentación, el CONTROLADOR que presenta las clases y objetos para el control de las peticiones y el flujo de acciones a realizar, el LIB donde se encuentran las clases del negocio, los formularios y algunas de las encargadas del mapeo de la base de datos y el paquete CONFIG que es donde se encuentra el fichero view.yml, en el cual se establece la estructura de las vistas por defecto: el nombre del Layout, las hojas de estilo CCS y los archivos JavaScript.

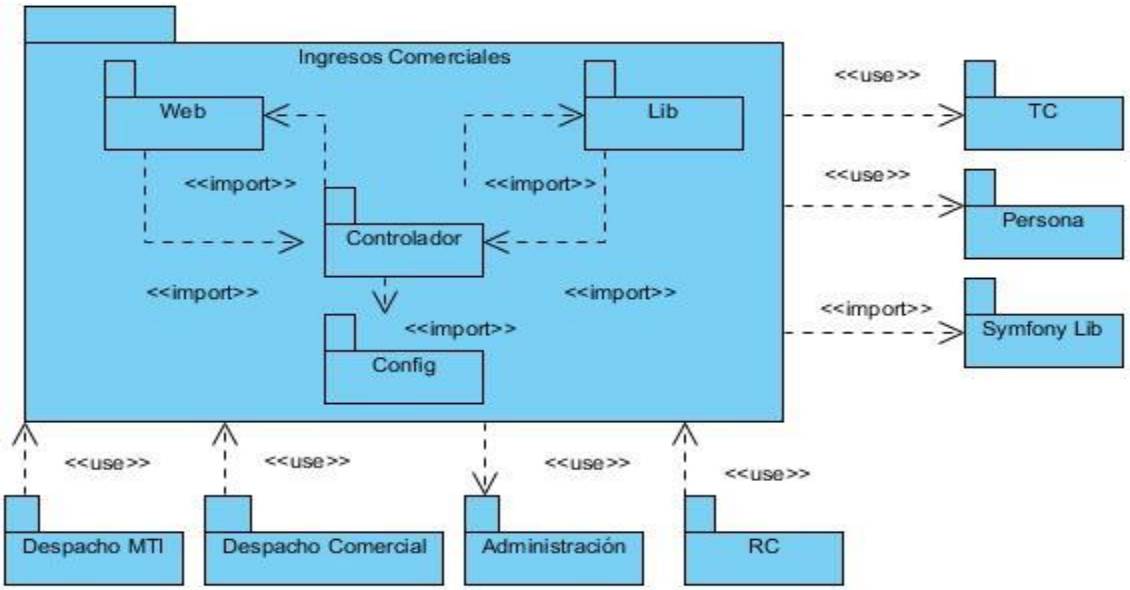


**Figura 4: Diagrama de paquetes del módulo Ingresos Comerciales.**

### 2.3.2 Diagrama de clases del diseño con estereotipos web

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema y los componentes que se encargarán del funcionamiento y la relación entre uno y otro [39].

Un **diagrama de clases** posibilita visualizar las relaciones entre las clases que involucra el sistema, las cuales pueden ser asociativas, de herencia o de uso.

Un diagrama de clases está compuesto por los siguientes elementos:

- Clase: atributos, métodos y visibilidad.
- Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

En la Figura 5 se muestra el diagrama de clases con estereotipos web del requisito Insertar Acuerdo de Aplazamiento del módulo Ingresos Comerciales donde se presenta la interfaz principal del módulo, la cual puede contener uno o varios formularios, que son los encargados de enviar la información a la clase servidora, esta contiene los atributos comunes y es la responsable de darle solución a las peticiones a través de las funcionalidades, así como enviar una respuesta a la página cliente. La página servidora interactúa con el modelo para la obtención de información.

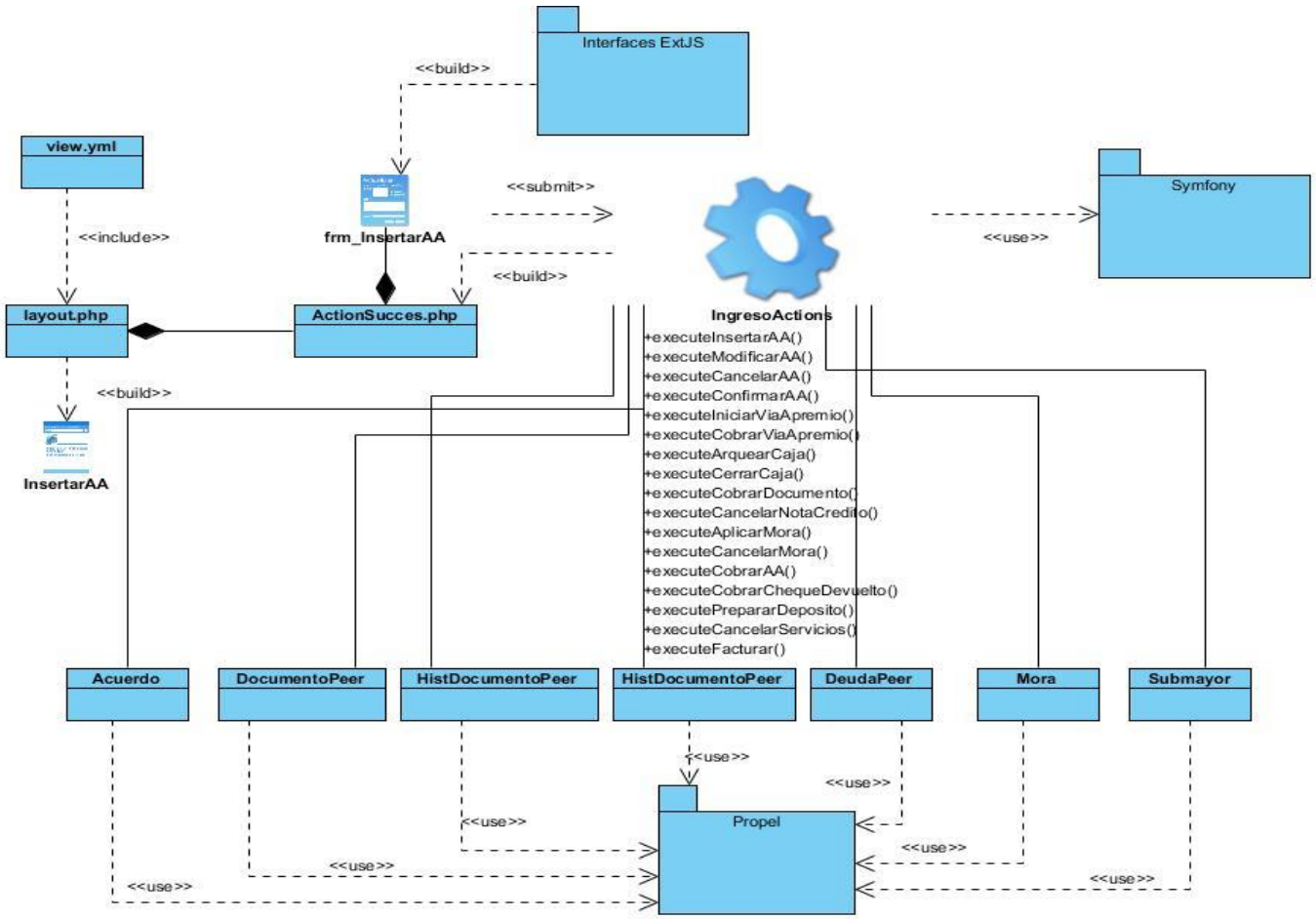


**Figura 5: Diagrama de clases con estereotipos web: Insertar Acuerdo de Aplazamiento.**

Para consultar el resto de los diagramas de clases del diseño con estereotipos web, consultar el Expediente de Proyecto GINA 1.1, Ingresos Comerciales.

### 2.3.3 Diagrama de secuencia

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. Mientras que el diagrama de casos de uso permite el modelado de un escenario de una vista del negocio, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos [39].

Usualmente se examina la descripción de los requisitos para determinar qué objetos son necesarios para la implementación del escenario. Si se dispone de la descripción de cada requisito como una secuencia de varios pasos, entonces se puede seguirlos mismos para descubrir qué objetos son necesarios para que se puedan seguir los pasos. Un diagrama de secuencia muestra los objetos que intervienen en el escenario con líneas discontinuas verticales, y los mensajes pasados entre los objetos como flechas horizontales. El objetivo del diagrama de secuencia es lograr la expresión escrita de lo que se pretende con la realización del programa o proyecto que se planifica [39].

En el diagrama de secuencia de la Figura 6 se representa el requisito Insertar Acuerdo de Aplazamiento, donde se muestra una serie de pasos lógicos a seguir para implementar dicho requisito. El objetivo de este diagrama es realizar todas las validaciones del mismo, además de mantener una secuencia lógica en el momento de realizar un nuevo acuerdo de aplazamiento. Además, incluye también actividades que identifican llamadas a funcionalidades.

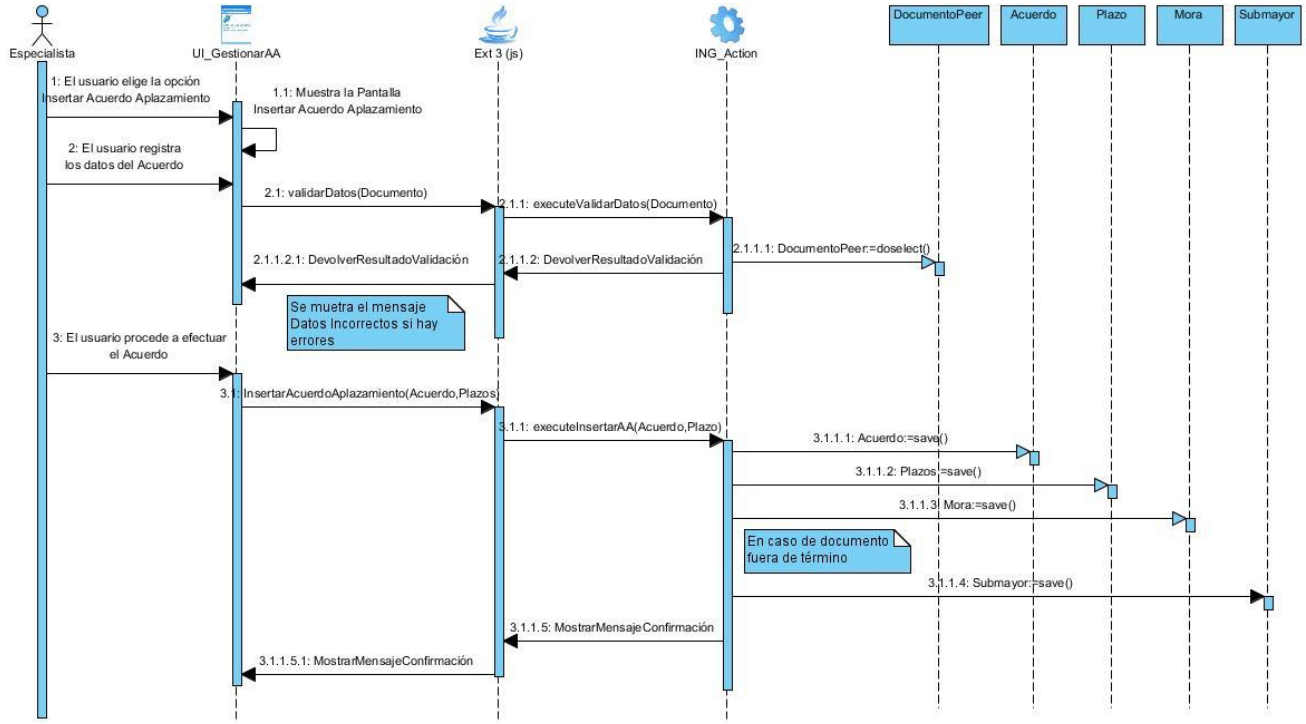


**Figura 6: Diagrama de secuencia: Insertar Acuerdo Aplazamiento.**

Para consultar el resto de los diagramas de secuencias, buscar el Expediente de Proyecto GINA 1.1, Ingresos Comerciales.

### 2.3.4 Diagrama de clases persistentes

La Figura 7 muestra el diagrama de diseño de las clases persistentes del sistema, este sirve como una primera aproximación al diseño definitivo del modelo de datos.

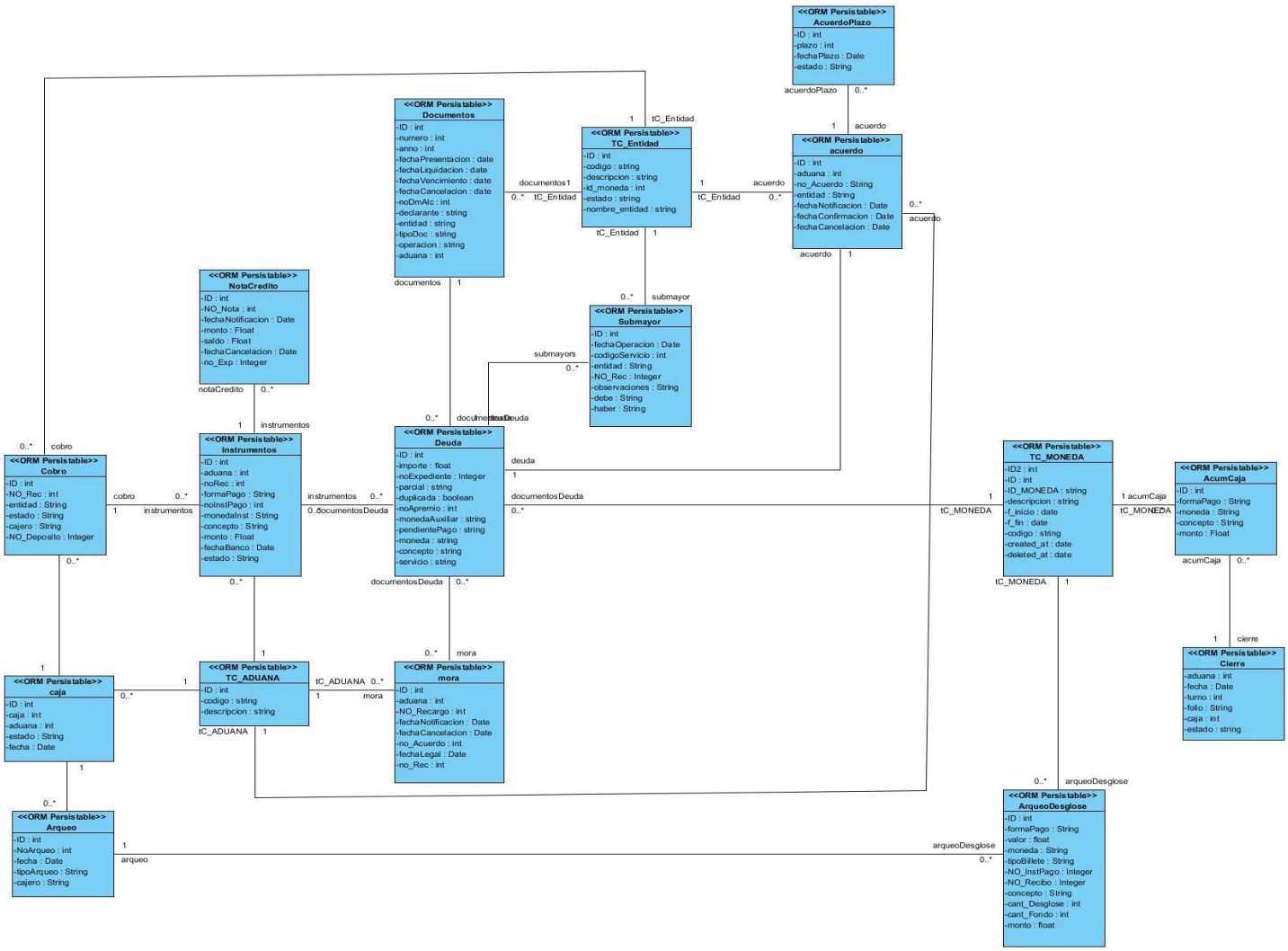


**Figura 7: Diagrama de clases persistentes.**

## 2.4 Modelo de base datos

El diseño de una base de datos es un proceso complejo que abarca decisiones a distintos niveles. La complejidad se controla mejor si se descompone el problema en sub-problemas y se resuelve cada uno de estos independientemente, utilizando técnicas específicas. Así, el diseño de una base de datos se descompone en diseño conceptual, diseño lógico y diseño físico.

El **diseño conceptual:** parte de las especificaciones de requisitos de usuario y su resultado es el esquema conceptual de la base de datos. Un esquema conceptual es una descripción de alto nivel de la estructura de la base de datos, independientemente del SGBD que se vaya a utilizar para manipularla [40].

El **diseño lógico:** parte del esquema conceptual y es una descripción de la estructura de la base de datos en términos de las estructuras de datos que puede procesar un tipo de SGBD [41].

El **diseño físico:** parte del esquema lógico y es una descripción de la implementación de una base de datos en memoria secundaria: las estructuras de almacenamiento y los métodos utilizados para tener un acceso eficiente a los datos [41].

### 2.4.1 Modelo Entidad-Relación

En la Figura 8 se muestra un fragmento del diagrama entidad-relación que fue generado para el sistema, el cual contiene las tablas que guardarán la información de los ingresos comerciales, para el requisito Insertar Acuerdo de Aplazamiento. El diagrama entidad-relación completo está disponible en el Expediente de Proyecto GINA 1.1, Ingresos Comerciales.

Las tablas de color amarillo son las TC las cuales son nomencladoras, estas contribuyen al completamiento de la información, permitiendo el acceso de todos los subsistemas que conforman el sistema GINA.

Las tablas de color azul son las más importantes arquitectónicamente debido a que en ellas se almacenan los datos principales de Ingresos Comerciales.

Todas las tablas conforman un modelo de entidad-relación, normalizado en Tercera Forma Normal (3FN), según la propuesta original de Codd [42], que plantea que una relación está en tercera forma normal si todos los atributos de la relación dependen funcionalmente sólo de la clave y no de ningún otro atributo.

La normalización de los datos puede considerarse como un proceso durante el cual los esquemas de relación que no cumplen las condiciones se descomponen repartiendo sus atributos entre esquemas de relación más pequeños que cumplen las condiciones establecidas. Un objetivo del proceso de normalización realizado para el modelo antes presentado, es garantizar que no ocurran anomalías de actualización.

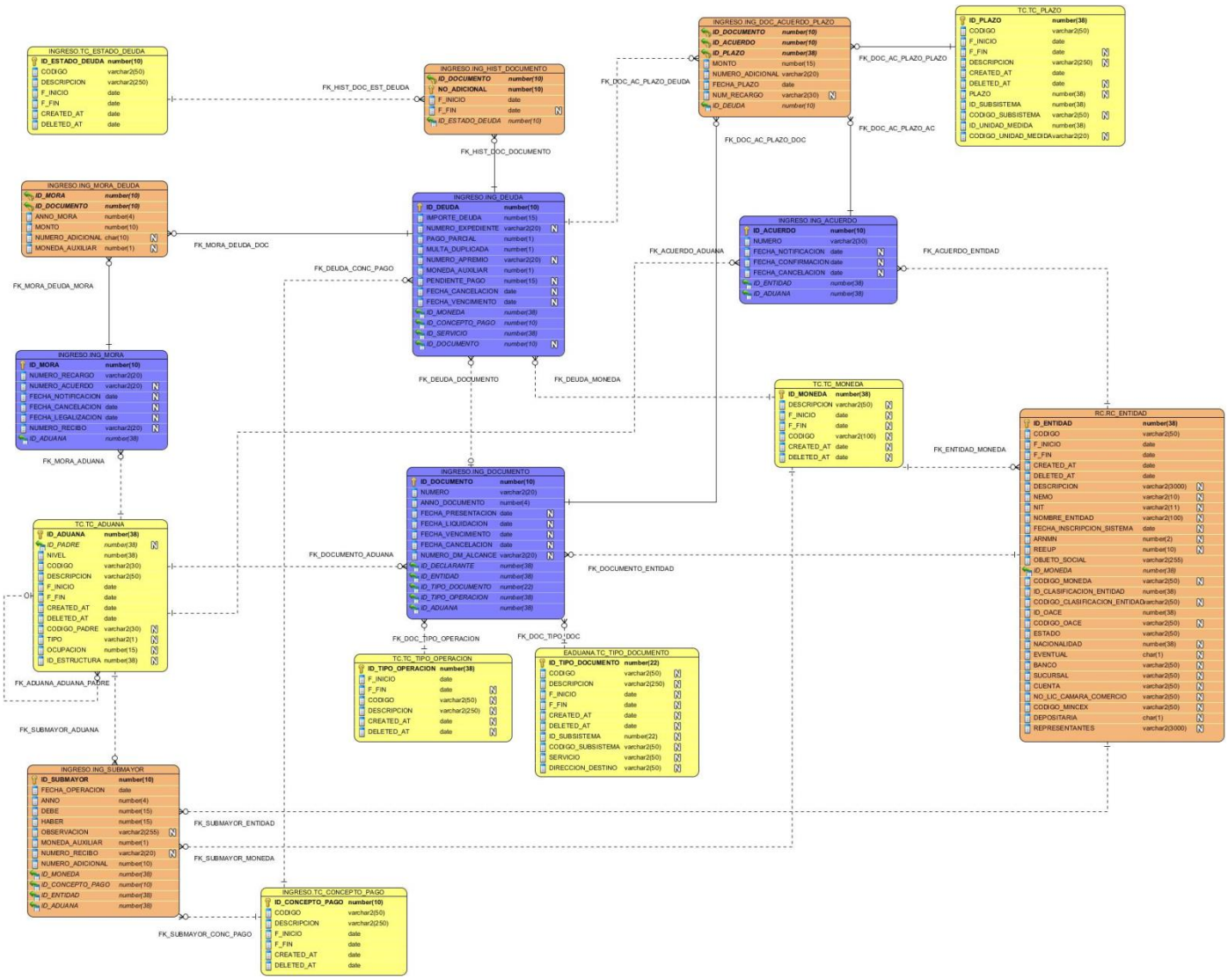


**Figura 8: Fragmento del modelo de datos.**

## 2.5 Validación del diseño

### Métricas orientadas a clases

Se sabe que la clase es la unidad principal de todo sistema Orientado a Objetos. Esto implica que las medidas y métricas para una clase individual, la jerarquía y las colaboraciones sean sumamente valiosas para un ingeniero de software que tenga que estimar la calidad de un diseño [11].

### Métricas propuestas por Lorenz y Kidd

Lorenz y Kidden dividen las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas a tamaños para una clase se centran en cálculos de atributos y de operaciones para una clase individual, y promedian los valores para el sistema en su totalidad. Las métricas basadas en herencia se centran en la forma en que se reutilizan las operaciones en la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión y asuntos relacionados con el código y las métricas orientadas a valores externos examinan el acoplamiento y la reutilización [43].

Del conjunto de métricas planteadas por los autores mencionados anteriormente se aplicó al diseño propuesto:

- Tamaño Operacional de Clase (TOC).
- Relaciones entre clases (RC).

### 2.5.1 Métrica Tamaño Operacional de Clase

Se empleará la métrica TOC la cual está dada por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad [43]:

- Responsabilidad: Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
- Complejidad de implementación: Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
- Reutilización: Un aumento del TOC implica una disminución del grado de reutilización de la clase.

Para la evaluación de dichos atributos de calidad, se definen los siguientes criterios y categorías de evaluación:

**Tabla 2: Criterios de evaluación de la métrica TOC.**

| Atributo | Categoría | Criterio |
|---|---|---|
| Responsabilidad | Baja | <=Promedio de operaciones(PO) |
| | Media | >PO y <=2*PO |
| | Alta | >2*PO |
| Reutilización | Baja | >2*PO |

| | | |
|---|---|---|
| | Media | >PO y <=2*PO |
| | Alta | <=PO |
| Complejidad | Baja | <=PO |
| | Media | >PO y <=2*PO |
| | Alta | >2*PO |

A continuación se muestran las medidas de la evaluación de la métrica TOC para las clases presentes en el módulo Ingresos Comerciales.

**Tabla 3: Clases de la capa de negocio a las que se les aplicó la métrica TOC.**

| No. | Clases | Operaciones | Responsabilidad | Complejidad | Reutilización |
|---|---|---|---|---|---|
| **1** | Acuerdo | 8 | Alta | Alta | Baja |
| **2** | Acumulado | 1 | Baja | Baja | Alta |
| **3** | Arqueo | 1 | Baja | Baja | Alta |
| **4** | ArqueoDesglose | 1 | Baja | Baja | Alta |
| **5** | ChequeTemp | 1 | Baja | Baja | Alta |
| **6** | CierreCaja | 1 | Baja | Baja | Alta |
| **7** | Cobro | 3 | Media | Media | Media |
| **8** | Deuda | 3 | Media | Media | Media |
| **9** | DocAcuerdoPlazo | 2 | Baja | Baja | Alta |
| **10** | Documento | 1 | Baja | Baja | Alta |
| **11** | DocInstrumento | 1 | Baja | Baja | Alta |
| **12** | Efectivo | 1 | Baja | Baja | Alta |
| **13** | HistDocumento | 1 | Baja | Baja | Alta |
| **14** | Instrumento | 1 | Baja | Baja | Alta |
| **15** | Mora | 6 | Alta | Alta | Baja |
| **16** | MoraDeuda | 1 | Baja | Baja | Alta |

| | | | | | |
|---|---|---|---|---|---|
| **17** | NotaCredito | 3 | Media | Media | Media |
| **18** | Submayor | 1 | Baja | Baja | Alta |
| **19** | Caja | 2 | Baja | Baja | Alta |

Se les aplicó la métrica de TOC a un total de 19 clases, con un total de 39 operaciones, para un promedio de procedimientos por clases de 2.05.

En la Figura 9 se observa la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo responsabilidad. Este gráfico muestra un resultado satisfactorio pues el 74% de las clases poseen una baja responsabilidad. Esta característica permite que en caso de fallos, como la responsabilidad está distribuida de forma equilibrada, ninguna clase es demasiado crítica como para dejar al sistema fuera de servicio.


**Figura 9: Resultados de la evaluación de la métrica TOC para el atributo responsabilidad.**

En la Figura 10 se observa la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo complejidad de implementación. Este gráfico muestra un resultado satisfactorio, pues el 74% de las clases poseen una baja complejidad de implementación. Esta característica permite mejorar el mantenimiento y soporte de estas clases.

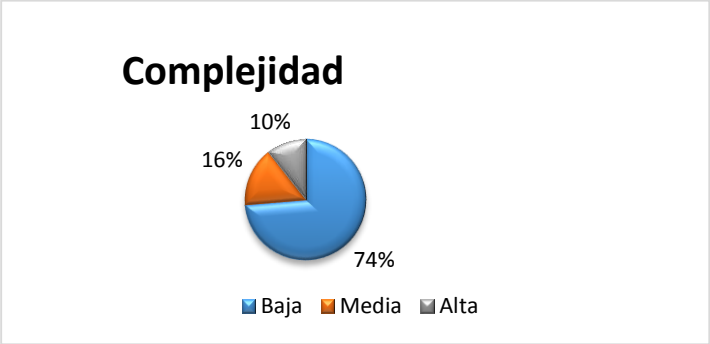


**Figura 10: Resultados de la evaluación de la métrica TOC para el atributo complejidad de implementación.**

La Figura 11 muestra la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo reutilización. Queda demostrado que el diseño de la solución es eficiente ya que solo el 10% de las clases poseen una baja reutilización.

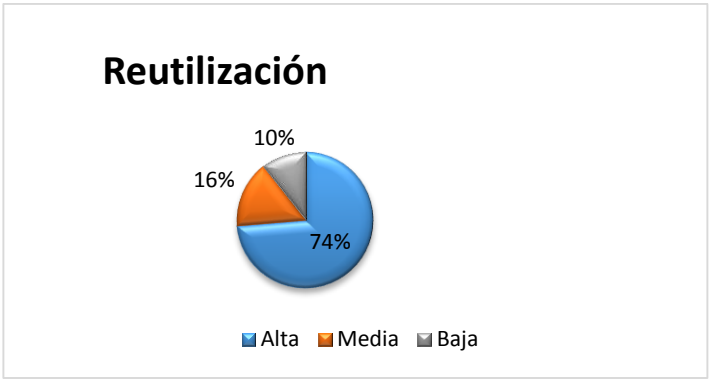


**Figura 11: Resultados de la evaluación de la métrica TOC para el atributo reutilización.**

Luego de haber realizado un análisis de los resultados obtenidos para los atributos de la métrica, se observa que el 90% de las clases que conforman el módulo para los atributos responsabilidad y complejidad están dentro de la categoría Media y Baja, mientras que el atributo reutilización cuenta con igual por ciento en las categorías Alta y Media mostrando así que el módulo cuenta con una elevada reutilización, baja complejidad y responsabilidad en el diseño propuesto. Por lo que se concluye que los resultados obtenidos según esta métrica son positivos.

### 2.5.2 Métrica Relaciones entre Clases

Las relaciones entre las clases, está dado por el número de relaciones de uso de una clase con otras. Se encarga de medir la calidad de acuerdo a los siguientes atributos [43]:

- Acoplamiento: Consiste en las conexiones físicas entre los elementos del diseño orientado a objeto, representan el acoplamiento dentro de un sistema orientado a objeto.
- Complejidad del mantenimiento: Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software.
- Cantidad de pruebas: Consiste en el número o grado de esfuerzo para realizar las pruebas de calidad del producto diseñado.
- Reutilización: Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

Para la evaluación de dichos atributos de calidad, se definen los siguientes criterios y categorías de evaluación:

**Tabla 4: Criterios de evaluación de la métrica RC.**

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

| | | |
|---|---|---|
| | Media | >Promedio y <= 2*Promedio |
| | Alta | >2*Promedio |

A continuación se muestran las medidas de la evaluación de la métrica RC para las clases presentes en el módulo Ingresos Comerciales.

**Tabla 5: Clases de la capa de negocio a las que se les aplicó la métrica RC.**

| Clases | Cantidad de relaciones de uso | Acoplamiento | Complejidad | Reutilización | Cant. Pruebas |
|---|---|---|---|---|---|
| Acuerdo | 8 | Alto | Alta | Baja | Alta |
| Acumulado | 0 | Ninguno | Baja | Alta | Baja |
| Arqueo | 0 | Ninguno | Baja | Alta | Baja |
| ArqueoDesglose | 0 | Ninguno | Baja | Alta | Baja |
| ChequeTemp | 1 | Bajo | Baja | Alta | Baja |
| CierreCaja | 3 | Alto | Media | Media | Media |
| Cobro | 0 | Ninguno | Baja | Alta | Baja |
| Deuda | 1 | Bajo | Baja | Alta | Baja |
| DocAcuerdoPlazo | 6 | Alto | Alta | Baja | Alta |
| Documento | 1 | Bajo | Baja | Alta | Baja |
| DocInstrumento | 1 | Bajo | Baja | Alta | Baja |
| Efectivo | 0 | Ninguno | Baja | Alta | Baja |
| HistDocumento | 1 | Bajo | Baja | Alta | Baja |
| Instrumento | 0 | Ninguno | Baja | Alta | Baja |
| Mora | 7 | Alto | Alta | Alta | Alta |
| MoraDeuda | 1 | Bajo | Baja | Alta | Baja |
| NotaCredito | 1 | Bajo | Baja | Alta | Baja |
| Submayor | 1 | Bajo | Baja | Alta | Baja |
| Caja | 1 | Bajo | Baja | Alta | Baja |

Se les aplicó la métrica de RC a un total de 19 clases, con un total de 33 relaciones de uso, para un promedio de relaciones de uso por clases de 1.74.

La Figura 12 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo acoplamiento. Se evidencia un bajo acoplamiento entre las clases pues el 68% de las clases presentan una o ninguna dependencia con otra. Este resultado es favorable para el diseño del módulo pues al existir poca dependencia entre las clases aumenta el grado de reutilización del mismo.

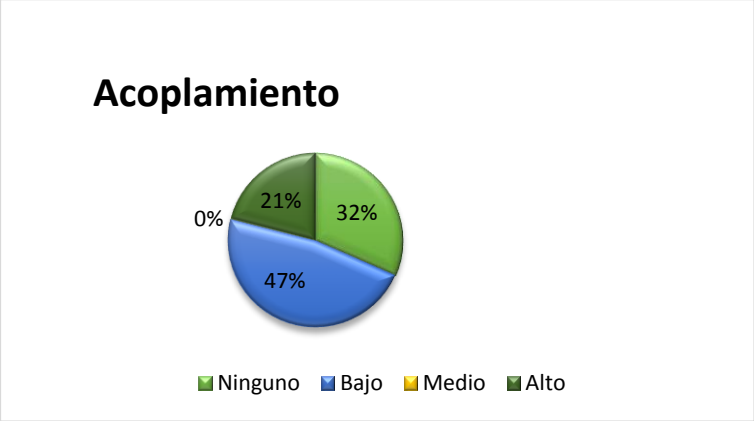


**Figura 12: Resultados de la evaluación de la métrica RC para el atributo acoplamiento.**

La Figura 13 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo complejidad de mantenimiento. El gráfico refleja un resultado aceptable del atributo pues el 79% de las clases presentan una baja complejidad de mantenimiento.

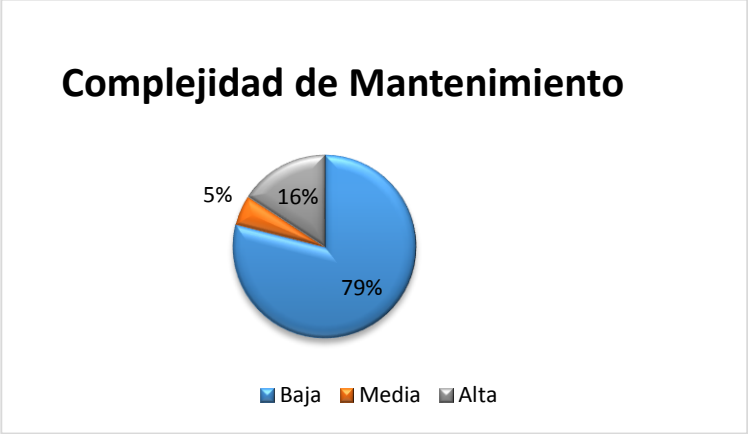


**Figura 13: Resultados de la evaluación de la métrica RC para el atributo complejidad de mantenimiento.**

La Figura 14 muestra la representación de la incidencia de los resultados de la evaluación del atributo reutilización. Esto evidencia que el 79% de las clases poseen una alta reutilización lo que es un factor fundamental que se debe tener en cuenta en el desarrollo de software.

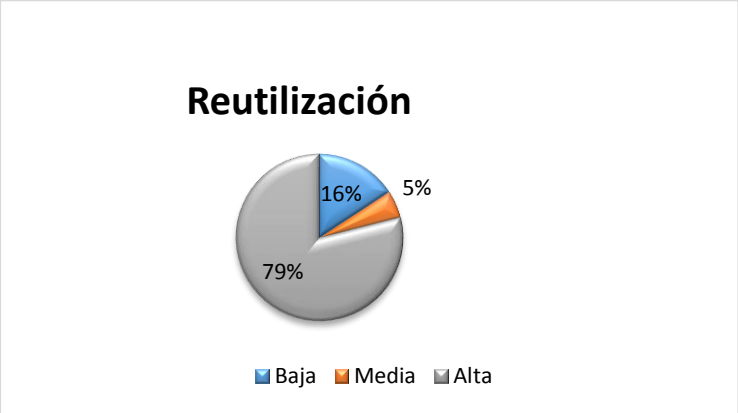


**Figura 14: Resultados de la evaluación de la métrica RC para el atributo reutilización.**

La Figura 15 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo cantidad de pruebas. Esto evidencia un resultado aceptable del atributo ya que el número de pruebas a realizar en el 79% de las clases es bajo.

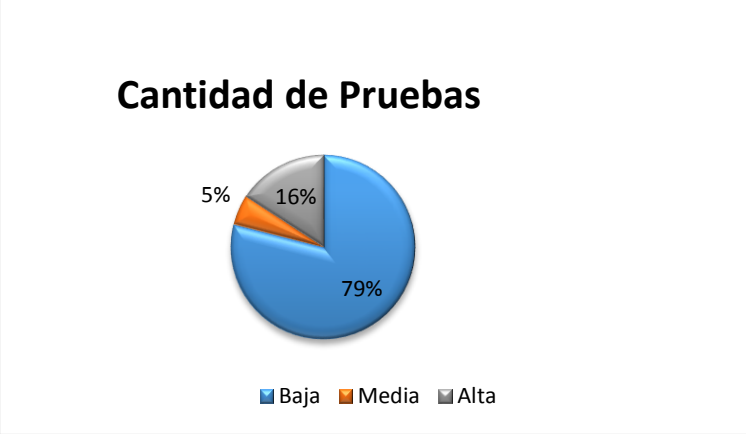


**Figura 15: Resultados de la evaluación de la métrica RC para el atributo cantidad de pruebas.**

Al analizar los resultados obtenidos de la métrica RC, se puede concluir que el diseño asumido tiene una calidad aceptable. Los atributos de calidad se encuentran en un nivel satisfactorio; en el 79% de las clases el nivel de acoplamiento es mínimo. La complejidad de mantenimiento y la cantidad de pruebas son bajas

a un 79%, mientras que el atributo Reutilización cuenta con igual por ciento en la categoría Alta, lo que representa valores favorables para el diseño realizado.

## 2.6 Modelo de implementación

El flujo de trabajo de implementación tiene como objetivo definir la organización del código, la implementación de los elementos de diseño en términos de ficheros fuentes, binarios y ejecutables, para poder integrar los diferentes componentes de desarrolladores o equipos y generar un ejecutable entregable o producto final [11].

### 2.6.1 Estándar de codificación

En la actualidad existen diferentes estándares para cada uno de los lenguajes, su utilización permite una comunicación fluida y directa entre los programadores de manera que se favorece la reutilización y mantenimiento de los sistemas. El Departamento de Soluciones para la Aduana de CEIGE presenta su propia propuesta de estándar de codificación para todas las aplicaciones a desarrollar [44]. A continuación se citan algunos de los aspectos relevantes que presenta dicho documento.

**Acciones (métodos o funcionalidades de la clase nombreDelModuloAction.class.php):**

- Dentro de las especificaciones del Framework utilizado está que cada una de estas acciones debe comenzar con la palabra execute.
- Todos los nombres de acciones deben estar en la nomenclatura “LowerCamelCase” comenzando por la palabra execute.
- En caso de ser acciones referentes a un módulo de CRUD de una tabla deben ser nombres específicos como executeNuevo, executeEditar, entre otros.
- Los nombres de las acciones deben especificar con la menor cantidad de palabras cuál es el objetivo de la acción, de ser posible estar en infinitivo. Debe especificar bien claro cuál es la acción que se pretende ejecutar, pero sin especificar los parámetros que recibe.

**Nombre de las clases:**

- Los nombres de las clases deben estar expresados en notación “UpperCamelCase”.
- No se deben utilizar guiones bajos en su nombre “_”.

- Deben expresar con claridad cuál es el alcance y la responsabilidad de la clase.
- Los nombres de las clases no deben estar atados a las clases de las que se deriva, cada clase debe tener un significado por ella misma, no en dependencia de la clase de la que deriva.
- En los nombres compuestos por más de tres palabras se debe revisar el diseño, no sea que se le estén dando a la clase más responsabilidades de las que realmente tiene.

**Nombres de las funciones:**

- Todas las funciones definidas por los desarrolladores deben seguir la nomenclatura "LowerCamelCase", a no ser que para cierto ámbito se especifiquen características específicas.

Además deben cumplir con las siguientes disposiciones de forma general:

- Los nombres de las funciones deben dejar reflejada claramente cuál es la acción que realiza el mismo.
- Se debe apoyar en la utilización de sufijos que ayuden a identificar el resultado final de la ejecución de un método.
- Se debe apoyar en los prefijos para expresar la acción que realiza sobre un elemento determinado.

**Variables:**

- Los nombres de las variables deben expresar claramente el contenido de la misma.
- Pueden estar referidas en singular o plural.
- Se definen al principio de las estructuras donde son utilizadas.
- En caso de que no se le asigne un valor inicial se deben inicializar con un valor que indique el tipo de dato más general al que debe pertenecer.
- De esta nomenclatura se exceptúan las variables generadas por el Framework.

**A nivel de base de datos** [45]**:**

- Los nombres de las tablas se escribirán siempre en mayúsculas y en singular.
- Las tablas comenzarán con el identificador del esquema al que pertenecen, seguido de un guión bajo y el nombre de la tabla. (Ing_Nombre)

- En caso de que el nombre de una tabla conste de más de una palabra, los espacios en blanco se marcarán con un guión bajo.
- El nombre del campo clave debe estar compuesto por “id” + nombre de la tabla en singular (para claves no foráneas).
- No se pueden poner caracteres extraños en los nombres de los campos como ñ, tilde, estos deben ser sustituidos por símbolos semejantes.

### 2.6.2 Tratamiento de errores

El tratamiento de errores es un aspecto importante para toda aplicación. En el módulo Ingresos Comerciales se controla información importante que influye en la toma de decisiones valiosas para el Despacho Comercial realizado por la AGR, es por eso que se realizan varios tipos de validaciones de errores.

- En Symfony la validación en el servidor es obligatoria para no corromper la base de datos con datos incorrectos. La validación en el lado del cliente es opcional, pero mejora la experiencia de usuario.
- Las validaciones del negocio son las más importantes de todas, estas se realizan por medio de los formularios. Se encargan de controlar el flujo correcto de los datos introducidos, para evitar consecuencias alarmantes. Se lleva a cabo en las diferentes clases por medio del lenguaje PHP. En caso de encontrarse casos de este tipo, se le informa al usuario los datos que están erróneos para que pueda corregirlos.
- Las validaciones realizadas en la vista juegan también un papel importante, estas se realizan por medio de expresiones regulares.

Todas tienen como función principal evitar que se introduzcan tipos de datos incorrectos en los diferentes campos, logrando así evitar ataques contra el sistema por esta vía.

### 2.6.3 Comunicación entre capas

Symfony está basado en un patrón clásico del diseño web conocido como arquitectura MVC, que está formado por tres niveles. A continuación se muestra un ejemplo de cómo se evidencia el patrón arquitectónico MVC en el requisito “Preparar depósito” del módulo.

La vista se encarga de obtener y enviar la información necesaria hacia el controlador a través de peticiones.

```
buttons: [
{
    text:'Depositar',
    iconCls: 'iconOk',
    handler: function(){
        Ext.Ajax.request({
            url: 'ingreso/prepararDeposito',
            method : 'POST',
            params: {
                datos:Ext.encode(json)
            },
```

**Figura 16: Acción desde la interfaz prepararDeposito.js. (Vista)**

Luego el controlador se encarga de obtener los datos enviados desde la vista y según la petición, realiza la acción correspondiente, la cual accediendo al modelo devolverá la respuesta determinada.

```
public function executePrepararDeposito(sfWebRequest $request) {
    $depositos = json_decode($request->getParameter('datos'));
    $noDeposito = str_pad((IngCobroPeer::contarDeposito() + 1) . '/' . date('Y'), 10, "0", STR_PAD_LEFT);

    if (!is_null($depositos)) {
        foreach ($depositos as $numeroRecibo) {
            $resultado = IngCobro::depositar($numeroRecibo, $noDeposito);
            if (is_string($resultado)) {
                return $this->renderText(json_encode(array('success' => 'false', 'msg' => $resultado)));
            }
        }
        return $this->renderText(json_encode(array('success' => 'true', 'msg' => 'Cobros depositados satisfactoriamente.')));
    } else {
        return $this->renderText(json_encode(array('success' => 'false', 'msg' => 'No hay recibos para depositar')));
    }
}
```

**Figura 17: Acción correspondiente a la petición. (Controlador)**

```
public static function depositar($numeroRecibo, $noDeposito) {
    $cobro = IngCobroPeer::devolverCobrosNumero($numeroRecibo);
    $errores = "";

    if (!is_null($cobro)) {
        $cobro->setNumeroDeposito($noDeposito);
        $cobro->save();
        return true;
    } else {
        $errores .= "El cobro con número de recibo " . $numeroRecibo . " no se puede depositar.";
    }

    return $errores;
}
```

**Figura 18: Función "depositar". (Modelo)**

### 2.6.4 Prototipos de clases

Es necesario partir de que Javascript es un lenguaje interpretado por los navegadores en el cliente, y no comprende el paradigma de la programación orientada a objetos [46]. Esto trae consigo que los conceptos de clase, componente y objeto, no son manejados explícitamente por este lenguaje, sino que son simulados a partir de la propiedad prototype. Esta propiedad permite gestionar los atributos y métodos de un objeto, brindando un comportamiento similar a una clase en la programación orienta a objetos.

El framework ExtJS maneja internamente la propiedad prototype, y propone su propia estructura para crear funciones que serán manejadas como clases, y que incluyen simuladamente el concepto de herencia en la implementación de la función Ext.extend(). De esta forma el framework permite crear nuevas funciones que heredan el comportamiento de otras funciones, por lo cual pueden ser denominados indistintamente los conceptos de clase, herencia, polimorfismo, atributo e instancia.

Al igual que la función Ext.extend(), Ext.JS brinda las funciones Ext.ns() y Ext.reg(), para crear paquetes de clases y registrarlas para ser reconocidas por el framework respectivamente. Los componentes visuales a desarrollar se ajustarán a un modelo estructural de clase para su implementación, cumpliendo así el estándar propuesto por ExtJS para la creación de nuevas clases

### 2.6.5 Formato JSon

JSON (JavaScript Object Notation) es un formato sencillo para intercambiar datos. Consiste básicamente en un arreglo asociativo de JavaScript que se utiliza para incluir información del objeto. JSON ofrece 2

grandes ventajas para las interacciones Ajax: es muy fácil de leer en JavaScript y puede reducir el tamaño en bytes de la respuesta del servidor [47].

El formato JSON es el más adecuado para la respuesta del servidor cuando la acción Ajax debe devolver una estructura de datos a la página que realizó la llamada de forma que se pueda procesar con JavaScript [47]. Este mecanismo es útil por ejemplo cuando una sola petición Ajax debe actualizar varios elementos en la página.

JSON se ha convertido en un estándar en el desarrollo de aplicaciones web. Los servicios web proponen la utilización de JSON en vez de XML para permitir la integración de servicios en el navegador del usuario en vez de en el servidor. El formato JSON es seguramente la mejor opción para el intercambio de información entre el servidor y las funciones JavaScript [48].

Desde la versión 5.2 de PHP existen dos funciones, json_encode() y json_decode(), que permiten convertir un arreglo PHP en un arreglo JSON y viceversa. Estas funciones facilitan la integración de los arreglos JSON (y de Ajax en general) y permiten escribir código PHP nativo más fácil de leer.

**json_decode**

Decodifica un string JSON. Retorna el valor codificado de json en un tipo PHP apropiado. Los valores true, false y null (case-insensitive) se retornan como TRUE, FALSE y NULL respectivamente. NULL se retorna si no se puede decodificar json o si los datos superan el límite de profundidad de recursión indicado por depth.

**json_encode**

Retorna la representación JSON del valor dado. Si no hay error, retorna un string codificado en JSON.

### 2.6.6 Acciones y clases del modelo

Las bases de datos son relacionales. PHP 5 y Symfony están orientados a objetos. Para acceder de forma efectiva a la base de datos desde un contexto orientado a objetos, es necesaria una interfaz que traduzca la lógica de los objetos a la lógica relacional, esta interfaz se llama ORM (del inglés Object Relational Mapping).Dicha interfaz está formada por objetos que permiten acceder a los datos y que contiene en sí mismo el código necesario para hacerlo.

La principal ventaja que aporta el ORM es la reutilización, permitiendo llamar a los métodos de un objeto de datos desde varias partes de la aplicación e incluso desde diferentes aplicaciones [49].

La capa ORM también encapsula la lógica de los datos. Para crear el modelo de objetos de datos que utiliza Symfony, se debe traducir el modelo relacional de la base de datos a un modelo de objetos de datos. Para realizar ese mapeo o traducción, el ORM necesita una descripción del modelo relacional, que se llama esquema (schema). En el esquema se definen las tablas, sus relaciones y las características de sus columnas.

El esquema se utiliza para construir las clases del modelo que necesita la capa del ORM. Al analizar el esquema se generan las clases base del modelo:

- BaseAcuerdo.php
- BaseAcuerdoPeer.php
- BaseDocumento.php
- BaseDocumentoPeer.php
- Acuerdo.php
- AcuerdoPeer.php
- Documento.php
- DocumentoPeer.php

Las clases con nombre Base del modelo son las que se generan directamente a partir del esquema. Nunca se deberían modificar esas clases, porque cada vez que se genera el modelo, se borran todas ellas.

Por otra parte, las clases de objetos propias heredan de las clases con nombre Base. Estas clases no se modifican cuando se ejecuta la tarea propel:build-model, por lo que son las clases en las que se añaden los métodos propios. Las clases de tipo *"peer"*, son clases que tienen métodos estáticos para trabajar con las tablas de la base de datos. Proporcionan los medios necesarios para obtener los registros de dichas tablas.

La tarea del action.class.php es crear e iniciar variables, realizar cálculos complejos, comprobaciones de datos que provienen de la base de datos, los cuales se obtienen a través de las clases peer. Esta obtención de datos se realiza a través de la clase Criteria() la cual se utiliza para definir consultas sin utilizar SQL.

## 2.7 Conclusiones del capítulo

Durante el desarrollo del capítulo fueron expuestos los artefactos generados del diseño e implementación del módulo para los Ingresos Comerciales para la Aduana, por lo que se concluye que:

- Fueron generados los artefactos del modelado del diseño de las funcionalidades del módulo para los Ingresos Comerciales.
- Con el diseño del Modelo de Datos se representaron las entidades presentes en el sistema.
- Las métricas que fueron empleadas para realizar la validación del diseño propuesto, ofrecieron como resultado que el diseño estaba realizado de forma simple y cumple satisfactoriamente con los atributos medidos permitiendo con esto, darle paso a las actividades de la disciplina Implementación.
- El uso de estándares de codificación ofrecen una homogeneidad en la nomenclatura de las clases y métodos facilitando su comprensión a otros programadores.

# Capítulo 3: Validación de la solución

El desarrollo de sistemas de software lleva implícito una serie de actividades en las cuales los fallos y errores son frecuentes desde el momento inicial. Es por ello que el desarrollo de software debe ir acompañado de una o varias actividades que garanticen la calidad. Las pruebas de software son un elemento crítico para la garantía de la calidad y representa una revisión final de las especificaciones, del diseño y del código. El éxito de las mismas puede mejorar la percepción de calidad del usuario final y lograr su satisfacción. Tienen como objetivo principal asegurar que el software cumpla con las especificaciones requeridas y eliminar los posibles defectos que pudiera tener [50]. De manera específica tiene como propósito:

- Realizar la validación del software desarrollado, determina si el software satisface los requisitos.
- Realizar la verificación del software desarrollado, determina si los productos de una fase satisfacen las condiciones de la misma.

Es preciso aclarar que las pruebas no aseguran la ausencia de defectos en el software, pero sí detectan la mayor cantidad de defectos posibles para su debida corrección.

En este capítulo se especifica el conjunto de validaciones y pruebas que evalúan la calidad del sistema, así como el grado de cumplimiento a las necesidades del cliente o usuario.

## 3.1 Pruebas

Una vez generado el código fuente, el software debe ser probado para descubrir y corregir el máximo de errores posibles antes de su entrega al cliente. Las pruebas de software son una actividad en la cual el sistema es ejecutado bajo condiciones específicas, para determinar la calidad del producto, detectar todo posible mal funcionamiento y comprobar el cumplimiento de los requisitos del cliente. Estas técnicas facilitan una guía sistemática para diseñar pruebas que comprueben la lógica interna de los componentes software y verifiquen los dominios de entrada y salida del programa para descubrir errores en la funcionalidad, el comportamiento y rendimiento [50].

### 3.1.1 Prueba de caja blanca

La prueba de caja blanca, denominada a veces prueba de caja de cristal, es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba [51].

Mediante los métodos de prueba de caja blanca, el ingeniero del software puede obtener casos de prueba que [51]:

1. Garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo.
2. Ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa.
3. Ejecuten todos los bucles en sus límites y con sus límites operacionales.
4. Ejerciten las estructuras internas de datos para asegurar su validez.

**Prueba de camino básico**

La prueba del camino básico es una técnica de prueba de caja blanca propuesta inicialmente por Tom McCabe [52]. El método del camino básico permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa [51].

Pasos para aplicar esta técnica:

1. Usando el diseño o el código como base, se dibuja el correspondiente grafo de flujo.
2. Determinar la complejidad ciclomática del grafo de flujo resultante.
3. Determinar un conjunto básico de caminos linealmente independientes.
4. Preparar los casos de prueba que forzarán la ejecución de cada camino del conjunto básico.

A continuación se expone cómo se obtuvo el diseño de los casos de prueba de caja blanca aplicando la técnica del camino mínimo.

Se toma como funcionalidad de muestra el requisito “Insertar Acuerdo Aplazamiento”. Este requisito se realiza con el objetivo de registrar los acuerdos de aplazamientos que se le concedan al cliente, cuando este no puede pagar una deuda en tiempo.

Para utilizar la técnica del camino básico, primeramente se hace necesario el cálculo de la complejidad ciclomática del algoritmo que vaya a ser analizado, para lo cual se deben enumerar sus sentencias de código y a partir de ahí elaborar el grafo de flujo de esta funcionalidad.

Las sentencias enumeradas del método executeInsertarAA() se encuentran en el Anexo 1: Método Insertar Acuerdo de Aplazamiento.

En la Figura 19 se muestra el grafo de flujo asociado a la funcionalidad executeInsertarAA()

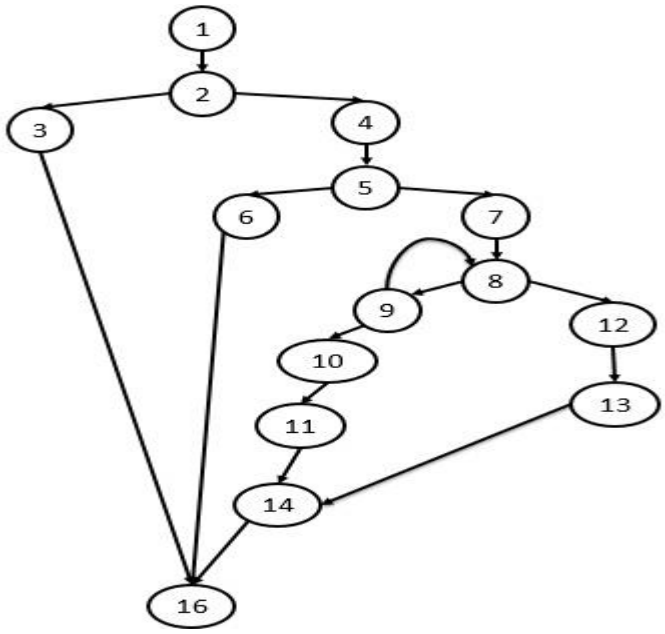


**Figura 19: Grafo de flujo asociado a la funcionalidad executeInsertarAA().**

### Cálculo de la complejidad ciclomática a partir de un segmento de código

La complejidad ciclomática es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y ofrece un límite superior para el número de pruebas que se deben realizar para asegurar que sea ejecutada cada sentencia al menos una vez [53].

Luego de haber construido el grafo se realiza el cálculo de la complejidad ciclomática mediante las tres fórmulas descritas a continuación, las cuales deben arrojar el mismo resultado para asegurar que el cálculo de la complejidad sea el correcto. Las fórmulas para realizar dicho cálculo son [54]:

$$V(G) = (A - N) + 2, V(G) = (18 - 16) + 2, V(G) = 4$$

Siendo A la cantidad total de aristas del grafo y N la cantidad de nodos.

$$V(G) = P + 1, V(G) = 3 + 1, V(G) = 4$$

Siendo P la cantidad de nodos predicado (son aquellos de los cuales parten dos o más aristas)

$$V(G) = R, V(G) = 4$$

Siendo R la cantidad de regiones que posee el grafo.

En cada una de las fórmulas V( G) ha representado el valor del cálculo. A partir de los resultados obtenidos en cada uno, se puede determinar que la complejidad ciclomática del código analizado es 4, que a su vez es el número de caminos posibles a circular el flujo y el límite superior de casos de prueba que se le pueden aplicar a dicho código.

A continuación se muestran los caminos básicos por donde puede circular el flujo para la funcionalidad Insertar Acuerdo de Aplazamiento:

**Camino básico # 1:** 1-2-3-16

**Camino básico # 2:** 1-2-4-5-6-16

**Camino básico # 3:** 1-2-4-5-7-8-9-8-10-11-14-16

**Camino básico # 4:** 1-2-4-5-7-8-12-13-14-16

Después de haber extraído los caminos básicos del flujo, se procede a ejecutar los casos de pruebas para este procedimiento, se debe realizar al menos un caso de prueba por cada camino básico. Para realizarlos es necesario cumplir con las siguientes exigencias:

- Descripción: Se hace la entrada de datos necesaria, validando que ningún parámetro obligatorio pase nulo al procedimiento o no se entre algún dato erróneo.
- Condición de ejecución: Se especifica cada parámetro para que cumpla una condición deseada y ver el funcionamiento del procedimiento.
- Entrada: Se muestran los parámetros que entran al procedimiento.
- Resultados esperados: Se expone el resultado que se espera que devuelva el procedimiento.
- Resultados: Se muestra el resultado obtenido.

**Tabla 6: Caso de prueba para el Camino básico # 1**

| **Camino básico #1: 1-2-3-16** | |
|---|---|
| Descripción | A partir de la entrada de los acuerdos y su respectivo número, se verifica si estos son válidos, para posteriormente insertarlo. |
| Condición de ejecución | Se debe tener el acuerdo con todos sus datos (acuerdo).<br>Se debe tener el número del acuerdo (noAcuerdo).<br>Se debe tener el resultado de la validación del acuerdo (result). |
| Entrada | `$acuerdos=[{"idPlazo":"1","fecha":"2014-05-01","tipoDoc":"1","numero":"00001","anno":"2010",""operacion":"2","moneda":"1","monto":400,"pagar":195}], $noAcuerdo=201445127 y $result =false.` |
| Resultado esperado | Se espera que no se inserte el nuevo acuerdo al sistema y muestre un mensaje de error informando que el acuerdo no cumple con los requisitos, ya que el documento no existe y no se le paga en ese plazo el total de la deuda. |
| Resultados obtenidos | No se inserta el acuerdo. |

**Tabla 7: Caso de prueba para el Camino básico # 2**

| **Camino básico #2: 1-2-4-5-6-16** | |
|---|---|
| Descripción | A partir de la entrada de los acuerdos y su respectivo número, se verifica si estos son válidos, para posteriormente insertarlo. |

| | |
|---|---|
| Condición de ejecución | Se debe tener el acuerdo con todos sus datos (acuerdo).<br>Se debe tener el número del acuerdo (noAcuerdo).<br>Se debe tener el resultado de la validación del acuerdo (result). |
| Entrada | `$acuerdos=[{"idPlazo":"1","fecha":"5","tipoDoc":"1","numero":"00001","anno":"2014",""operacion":"1","moneda":"1","monto":400,"pagar":195}], $noAcuerdo=201445127 y $result =false.` |
| Resultado esperado | Se espera que no se inserte el nuevo acuerdo al sistema y muestre un mensaje de error informando que el acuerdo no cumple con los requisitos, ya que el campo fecha no está correcto, mostrando un mensaje informando que el acuerdo no ha sido insertado, dado que la fecha no es correcta. |
| Resultados obtenidos | No se inserta el acuerdo. |

**Tabla 8: Caso de prueba para el Camino básico # 3**

| **Camino básico #3: 1-2-4-5-7-8-9-8-10-11-14-16** | |
|---|---|
| Descripción | A partir de la entrada de los acuerdos y su respectivo número, se verifica si estos son válidos, para posteriormente insertarlo. |
| Condición de ejecución | Se debe tener el acuerdo con todos sus datos (acuerdo).<br>Se debe tener el número del acuerdo (noAcuerdo).<br>Se debe tener el resultado de la validación del acuerdo (result). |
| Entrada | `$acuerdo= {plazo= 1, fecha= 5, tipodoc= 11, noDoc= 00005, operación= 1, moneda= 1, monto= 400, pagar= 400}, $noAcuerdo=201445127 y $result =true.` |

| | |
|---|---|
| Resultado esperado | Se espera que no se inserte el nuevo acuerdo al sistema y muestre un mensaje de error informando que el acuerdo no cumple con los requisitos, mostrando un mensaje informando que no es posible actualizarle el estado al documento ya que este no existe. |
| Resultados obtenidos | No se inserta el acuerdo. |

**Tabla 9: Caso de prueba para el Camino básico # 4**

| **Camino básico #4: 1-2-4-5-7-8-12-13-14-16** | |
|---|---|
| Descripción | A partir de la entrada de los acuerdos y su respectivo número, se verifica si estos son válidos, para posteriormente insertarlo. |
| Condición de ejecución | Se debe tener el acuerdo con todos sus datos (acuerdo).<br>Se debe tener el número del acuerdo (noAcuerdo).<br>Se debe tener el resultado de la validación del acuerdo (result). |
| Entrada | `$acuerdo= {plazo= 1, fecha= 5, tipodoc= 11, noDoc= 00001, operación= 1, moneda= 1, monto= 400, pagar= 400}, $noAcuerdo=201445127 y $result =true.` |
| Resultado esperado | Se espera que se inserte correctamente el Acuerdo, mostrando un mensaje informando que el acuerdo ha sido insertado satisfactoriamente. |
| Resultados obtenidos | Se inserta el acuerdo. |

Con la aplicación de los casos de prueba expuestos anteriormente se comprobó que el flujo de trabajo de las funcionalidades es correcto, se probó que cada sentencia es ejecutada al menos una vez, cumpliéndose así las condiciones de la prueba.

### 3.1.2 Prueba de caja negra.

Las pruebas de caja negra, también denominada prueba de comportamiento, se centran en los requisitos funcionales del software. Dichas pruebas permiten al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La prueba de caja negra no es una alternativa a las técnicas de prueba de caja blanca, más bien se trata de un enfoque complementario que intenta descubrir diferentes tipos de errores. La prueba de caja negra intenta encontrar errores de las siguientes categorías: funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos o en accesos a BD externas, errores de rendimiento y errores de inicialización y de terminación [55].

Como resultado de las pruebas de caja negra se realizó pruebas a los diseño de casos de prueba por cada requisito, los cuales fueron aplicados en 3 iteraciones de pruebas de revisiones internas realizadas por el equipo de desarrollo del proyecto, en las cuales se detectaron varias no conformidades, 25 durante la primera iteración, 10 en la segunda y 0 en la tercera, las cuales fueron mayormente errores de interfaz de usuario, 5 errores de funcionalidad y 5 de ortografía. Todas las no conformidades fueron resueltas comprobando que la aplicación cumple con lo planteado en la descripción de los requisitos.

El diseño de casos de pruebas basado en requisitos se realizó a partir de una plantilla definida en el expediente 3.3 del proceso de mejora del nivel 2 de CMMI. Esta plantilla tiene 2 páginas, en la primera se describe qué debe suceder cuando se ejecuta la funcionalidad que se está probando. Para esto se especifican diferentes combinaciones de datos válidos y no válidos para verificar cuál es la respuesta del sistema ante estos tipos de entrada. En la Figura 20 se muestra el caso de pruebas del requisito “Insertar Acuerdo de Aplazamiento”. Para consultar el resto de los casos de prueba, buscar en el Expediente de Proyecto GINA 1.1, Ingresos Comerciales.

El usuario debe estar autenticado en el sistema
Deben existir los documentos a los que se le desea otorgarle una acuerdo aplazamiento

**SC Insertar Acuerdo de Aplazamiento**

| Escenario | Descripción | Plazo | Fecha | Tipo Doc | Número Doc | Operación | Año | Concepto | Moneda | Pagar | Respuesta del sistema | Flujo central |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| EC 1.1 Insertar Acuerdo de Aplazamiento | Se inserta un nuevo acuerdo de aplazamiento a un documento | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | El sistema debe mostrar un mensaje indicando que no son válidos los datos. | 1- Seleccionar del menú la opción Gestionar Acuerdo de Aplazamiento.<br>2- Especificar el plazo que se le va a asignar a el documento.<br>2- Introducir la fecha enla que se va a pagar el plazo.<br>3- Especificar el tipo de documento.<br>4- Introducir el número del documento que se va a aplazar.<br>5- Especificar el tipo de operación.<br>6- Introducir el año del documento.<br>7- Especificar el tipo de concepto de deuda que se va a pagar.<br>8- Especificar el tipo de moneda con la que se va a pagar.<br>9-Insertar la cantidad que se va a pagar en el plazo.<br>10- Se presiona el botón Aceptar.<br>11- El sistema valida los datos registrados, si son correctos muestra un mensaje indicando que se ha realizado la operación con éxito o un mensaje de error en caso de ocurrir alguno, en caso contrario muestra un mensaje indicando que los datos no son válidos. |
| | | vacío | vacío | vacío | vacío | vacío | vacío | vacío | vacío | vacío | | |
| | | V | V | V | V | V | V | V | V | V | El sistema debe mostrar un mensaje indicando que se ha realizado la operación con éxito. | |
| | | 1 | 1/5/2014 | DMTransferencia | 1 | Importación | 2014 | Arancel | Peso Cubano | 400 | | |
| | | V | N/A | N/A | I | V | V | N/A | N/A | N/A | El sistema debe mostrar un mensaje indicando que el documento no existe. | |
| | | 2 | | Multa | Doc4 | Importación | 2014 | | | | | |

*[Las celdas de la tabla contienen V, I, o N/A. V indica válido, I indica inválido, y N/A que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante.]*

**Figura 20: Diseño de casos de pruebas del requisito Insertar Acuerdo de Aplazamiento.**

## 3.2 Conclusiones del capítulo

- Se realizaron pruebas funcionales a la aplicación mostrando como resultado un correcto funcionamiento.
- Con las pruebas y validaciones realizadas se puede concluir que la herramienta desarrollada cumple con las especificaciones y requisitos definidos por los clientes.

# Conclusiones

Luego del desarrollo del módulo Ingresos Comerciales para el sistema GINA 1.1, se arriba a las siguientes conclusiones:

- La realización de un estudio del estado del arte de los sistemas para la gestión de los Ingresos Comerciales para las Aduanas, arrojó como resultado que ninguno de los sistemas estudiados podía ser íntegramente parte de la solución, ya que cada país tiene sus propias legislaciones y leyes del comercio exterior, por lo que se estudiaron de ellos características que podían ser ajustadas a las necesidades de la Aduana General de la República de Cuba.
- Como parte de la solución obtenida se lograron generar una serie de artefactos, que fueron la base para la posterior implementación del sistema, destacando el diagrama de secuencia, el diagrama de paquetes y el diagrama de clases.
- Se utilizó, en la creación del sistema, el patrón arquitectónico MVC, trayendo como resultado, que quedara separada la arquitectura en tres niveles distintos, logrando facilitar el mantenimiento en caso que sea necesario.
- Se desarrolló el módulo Ingresos Comerciales obteniendo un producto funcional acorde a los requisitos identificados.
- Se realizaron pruebas para validar el diseño mediante las métricas Tamaño Operacional de Clase y Relaciones entre Clases arrojando resultados satisfactorios.
- El módulo Ingresos Comerciales fue implementado correctamente y validado a través de pruebas de software, dándole solución a las necesidades del cliente.

# Recomendaciones

Se recomienda en el presente trabajo, considerando que se han cumplido los objetivos trazados:

- Continuar realizando pruebas de calidad a los componentes para garantizar su buen funcionamiento y certificarlos.
- Realizar el despliegue del módulo propuesto como parte del sistema GINA.

# Referencias bibliográficas


1. HERNÁNDEZ SAMPIERI, Roberto, FERNÁNDEZ COLLADO, Carlos and BAPTISTA LUCIO, Pilar. Fundamentos de metodología de la investigación. *Madrid [etc.]: McGraw-Hill*. 2007.

2. ASTI VERA, Armando. *Metodología de la investigación*. Caracas, 1968. ISBN 9802850217.

3. LAKATOS, Eva María and DE ANDRADE MARCONI, Marina. *Metodología científica*. Atlas São Paulo, 1991. ISBN 8522406413.

4. SAMPIERI, Roberto Hernández, COLLADO, Carlos Fernández, LUCIO, Pilar Baptista and PÉREZ, María de la Luz Casas. *Metodología de la investigación*. McGraw-Hill México, 1998. ISBN 9684229313.

5. ÁLVAREZ DE ZAYAS, Carlos and SIERRA LOMBARDÍA, Virginia. Metodología de la investigación científica. *La Habana: Editorial Ciencias Sociales*. 1995.

6. PEÑA, B. and SOFÍA, H. PROCEDIMIENTOS DE CONTROL APLICADOS POR EL AGENTE ADUANAL ADUATECA, CA, PARA EL DESADUANAMIENTO DE LA MERCANCÍA IMPORTADA A TRAVÉS DEL SISTEMA ADUANERO AUTOMATIZADO (SIDUNEA). 2009.

7. AMÉRICA DE LA CARIDAD MUÑOZ ALGUEZABAL, MECEDES VASALLO BERMÚDES Y NILDA VINICIA LEÓN CHAVEZ. Evolución de la Aduana Cubana. 2010. Dpto. Recursos Humanos Escuela Nacional de Formación Aduanera

8. DE KYOTO, Convenio. Protocolo sobre Cambio Climático de la ONU. *UNFCCC), puesto en marcha*. 2005.

9. SUAREZ, Mercedes. *Sistema automatizado de despacho mercantil*. La Habana, 2005. s.n.

10. CAMPINS-ERITJA, Mar. La acción internacional para reducir los efectos del cambio climático: El convenio marco y el protocolo de Kyoto. 1999.

11. ROGERS, Presman. Ingeniería de Software un Enfoque Práctico. *Editorial McGraw-Hill, Madrid*. 2005.

12. ING. WILLIAM GONZÁLEZ OBREGÓN. *Modelo de desarrollo de software*. 2013.

13. BOOCH, Grady, JACOBSON, I. and RUMBAUGH, James. The UML specification documents. *Santa Clara, CA.: Rational Software Corp. See documents at www. rational. com*. 1997.

14. LARMAN, Craig. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3/e*. Pearson Education India, 2012. ISBN 8177589792.

15. HOLZNER, Steven and FAGOAGA, Juan Carlos Vega. *PHP: manual de referencia*. McGraw-Hill, 2009. ISBN 9701067576.

16. ACHOUR, Mehdi, BETZ, Friedhelm, DOVGAL, Antony, LOPES, Nuno, MAGNUSSON, Hannes, RICHTER, Georg, SEGUY, Damien and VRANA, Jakub. Manual do PHP. *PHP Documentation Group. Disponível em*. 2007.

17. PARADIGM, Visual. Visual paradigm for UML. *Visual Paradigm for UML-UML tool for software application development*. 2010.

18. FIGUEROA, Roberth G., SOLÍS, Camilo J. and CABRERA, Armando A. Metodologías Tradicionales vs. Metodologías Ágiles. *Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación.(En línea), Disponible en: http://adonisnet.files.wordpress.com/2008/06/articulo-metodologia-de-sw-formato.doc*.

19. POTENCIER, Fabien and ZANINOTTO, François. symfony. *Eyrolles Collection*. 2009.

20. FABIEN POTENCIER, FRANÇOIS ZANINOTTO. Symfony 1.1, la guía definitiva. [En línea]. [Consultado el: 5 de junio de 2014]. Available from: http://librosweb.es/symfony_1_1/

21. ORCHARD, Leslie M., PEHLIVANIAN, Ara, KOON, Scott and JONES, Harley. *Professional JavaScript Frameworks: Prototype, YUI, ExtJS, Dojo and MooTools*. Wrox Press Ltd., 2009. ISBN 047038459X.

22. SHUIPING, Xiong. Information management system design based on ExtJS. *Science*. 2011. Vol. 11.

23. BOUDREAU, Tim, GLICK, Jesse, GREENE, Simeon, SPURLIN, Vaughn and WOEHR, Jack J. *NetBeans: the definitive guide*. O'Reilly Media, Inc., 2002. ISBN 1449332552.

24. BÖCK, Heiko, TULACH, Jaroslav and WIELENGA, Geertjan. *The Definitive Guide to NetBeans Platform*. Springer, 2009. ISBN 1430224177.

25. BRYLA, Bob. *Oracle Database 11g DBA Handbook*. Tata McGraw-Hill Education, 2007. ISBN 0070223645.

26. FREEMAN, Robert. *Oracle Database 11g New Features*. McGraw-Hill, Inc., 2007. ISBN 0071496610.

27. GREENWALD, Rick, STACKOWIAK, Robert, STERN, Jonathan, SAMUELSON, Kjell, PRESSMAN, Roger S., HANNA, M., BOEHM, B., CARRILLO, Antonio Garrido, SCHMULLER, Joseph and WEITZENFELD, Alfredo. Oracle Database 11g. *Fundamentos*. 1988.

28. GARRETT, Jesse James. Ajax: A new approach to web applications. 2005.

29. PAULSON, Linda Dailey. Building rich web applications with Ajax. *Computer*. 2005. Vol. 38, no. 10, p. 14–17.

30. LÓPEZ, Carlos Armando. Cómo mantener el patrón modelo vista, controlador en una aplicación orientada a la WEB. *Revista Inventum*. 2009. No. 7.

31. BUSCHMANN, Frank, HENNEY, Kelvin and SCHIMDT, Douglas. *Pattern-oriented Software Architecture: On Patterns and Pattern Language*. John Wiley & Sons, 2007. ISBN 8126512830.

32. SCHMIDT, Douglas C., STAL, Michael, ROHNERT, Hans and BUSCHMANN, Frank. *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*. John Wiley & Sons, 2013. ISBN 1118725174.

33. ZAPATA, Carlos Mario, GELBUKH, Alexander, ARANGO, Fernando, HERNÁNDEZ, Arturo and ZECHINELLI, José L. UN-Lencep: Obtención automática de diagramas UML a partir de un lenguaje controlado. *Avances en la Ciencia de la Computación*. 2006. P. 254–259.

34. SANCHEZ, D. J. *Validación de requisitos de usuario mediante técnicas de transformación de modelos y prototipación automática de interfaces de usuario*. PhD Thesis, Department of Information Systems and Computation, Valencia University of Technology, 2003.

35. KENDALL, Kenneth E. and KENDALL, Julie E. *Análisis y diseño de sistemas*. Pearson educación, 2005. ISBN 9702605776.

36. SOMMERVILLE, Ian. *Ingeniería del software*. Pearson Educación, 2005. ISBN 8478290745.

37. RODRÍGUEZ, Fco Javier Briones, RODRÍGUEZ, Alejandro Mesa, BUENDÍA, José Peso and BARRILADO, Fco Manuel Abril. Gestión del Proyecto. 2010.

38. STEVENS, Perdita, POOLEY, Rob, ALARCÓN, Marta Fernández, MARTÍNEZ, Óscar Sanjuan and SORROZAL, Francisco Pérez. *Utilización de UML en Ingeniería del Software con Objetos y Componentes*. Addison Wesley, 2007. ISBN 8478290869.

39. BOOCH, Grady, RUMBAUGH, James, JACOBSON, Ivar, MARTÍNEZ, José Sáez and MOLINA, Jesús J. García. *El lenguaje unificado de modelado*. Addison-Wesley, 1999.

40. ELMASRI, Ramez, NAVATHE, Shamkant B., CASTILLO, Verónica Canivell, ESPIGA, Beatriz Galán and PÉREZ, Gloria Zaballa. *Fundamentos de sistemas de bases de datos*. Addison-Wesley, 2002. ISBN 8478290516.

41. ROB, Peter and CORONEL, Carlos. *Sistemas de bases de datos: diseño, implementación y administración*. Cengage Learning Editores, 2004. ISBN 9706862862.

42. CODD, Edgar F. Further normalization of the data base relational model. *Data base systems*. 1972. Vol. 6, p. 33–64.

43. LORENZ, Mark and KIDD, Jeff. *Object-oriented software metrics: a practical guide*. Prentice-Hall, Inc., 1994. ISBN 013179292X.

44. CENTRO DE GESTIÓN DE ENTIDADES (CEIGE). *Estándares de codificación*. 2012. Universidad de las Ciencias Informáticas.

45. GARCÍA, Adrián Naranjo. *Estándar de codificación de Bases de Datos*. 2014.

46. FLANAGAN, David. *JavaScript. La Guía Definitiva*. 2007. ISBN 8441522022.

47. CROCKFORD, Douglas. The application/json media type for javascript object notation (json). 2006.

48. CROCKFORD, Douglas. JSON: The fat-free alternative to XML. En: *Proc. of XML*. 2006.

49. POREBSKI, Bartosz, PRZYSTALSKI, Karol and NOWAK, Leszek. *Building PHP Applications with Symfony, CakePHP, and Zend Framework*. John Wiley and Sons, 2011. ISBN 1118067924.

50. VEREAU ARAGÓN, Angel Enrique, SIFUENTES, SIFUENTES, James Jesús and AUCAHUASI, Barnet Molina. Arquitectura para el software de aseguramiento de calidad de los proyectos de software bajo el marco CMMI. 2013.

51. EZQUERRO CASADO, José Luis. Generador de conjuntos de pruebas paramétricos BPELUnit para composiciones WS-BPEL. 2013.

52. MCCABE, Thomas J. A complexity measure. *Software Engineering, IEEE Transactions on*. 1976. No. 4, p. 308–320.

53. SHEPPERD, Martin. A critique of cyclomatic complexity as a software metric. *Software Engineering Journal*. 1988. Vol. 3, no. 2, p. 30–36.

54. GILL, Geoffrey K. and KEMERER, Chris F. Cyclomatic complexity density and software maintenance productivity. *Software Engineering, IEEE Transactions on*. 1991. Vol. 17, no. 12, p. 1284–1288.

55. SCHROEDER, Patrick J. and KOREL, Bogdan. *Black-box test reduction using input-output analysis*. ACM, 2000. ISBN 1581132662.

# Anexos

## 1. Método Insertar Acuerdo de Aplazamiento

```
public function executeInsertarAcuerdo(sfWebRequest $request) {
        $acuerdo = json_decode($request->getParameter('acuerdo'));                                                                          //1
        $noAcuerdo = $request->getParameter('noAcuerdo');                                                                                   //1
        $result = IngAcuerdo::validarAcuerdo($acuerdo);                                                                                     //1
        if ($result != true) {                                                                                                              //2
            return $this->renderText(json_encode(array('success' => 'false', 'msg' => $result)));                                           //3
        } else {
            $idAduana = IngDocumentoPeer::getDocumentoDadoNumero($acuerdo[0]->numero)->getIdAduana();                                       //4
            $idEntidad = IngDocumentoPeer::getDocumentoDadoNumero($acuerdo[0]->numero)->getIdEntidad();                                     //4
            $result = IngAcuerdo::insertarAcuerdo($noAcuerdo, $idAduana, $idEntidad, $acuerdo);                                             //4
            if (is_string($result)) {                                                                                                       //5
                return $this->renderText(json_encode(array('success' => 'false', 'msg' => $result)));                                       //6
            } else {
                $documentosComprobados = array();                                                                                           //7
                $numDoc = $acuerdo[0]->numero;                                                                                              //7
                $idConceptoPago = $acuerdo[0]->concepto;                                                                                    //7
                $idMoneda = $acuerdo[0]->moneda;                                                                                            //7
                foreach ($acuerdo as $plazo) {                                                                                              //8
                    if (!array_search($numDoc, $documentosComprobados)) {                                                                   //9
                        $idDeuda = sfAuxiliarIngresoTC::idEstadoDeuda("AA")->getIdEstadoDeuda();                                            //10
                        $idDoc = IngDocumentoPeer::retrieveByPK($plazo->numero)->getIdDocumento();                                          //10
                        $noAdicional = IngHistDocumentoPeer::contarHistDoc($idDoc);                                                         //10
                        $resultActualizarEstado = IngHistDocumento::actualizarEstadoDocumento($noAdicional, $idDoc, $idDeuda);              //10
                        if ($resultActualizarEstado != true) {                                                                              //11
                            return $this->renderText(json_encode(array('success' => 'false', 'msg' => $resultActualizarEstado)));           //14
                        }
                        $documentosComprobados[] = $plazo->numero;                                                                          //10
                    }
                }
                $actualizarSubmayor = IngSubmayor::actualizarSubmayor($idAduana, $idEntidad, $result, 0, $idConceptoPago, $idMoneda);       //7
                if ($actualizarSubmayor != true) {                                                                                          //12
                    return $this->renderText(json_encode(array('success' => 'false', 'msg' => $actualizarSubmayor)));                       //13
                }
            }

            $msg = 'El acuerdo ' . $noAcuerdo . ' ha sido añadido satisfactoriamente.';                                                    //14
            return $this->renderText(json_encode(array('success' => 'true', 'msg' => $msg)));                                               //16
        }
    }
```

# Glosario de términos

**Acuerdo de Aplazamiento:** Pago aplazado de la deuda tributaria con independencia de la forma en que haya sido determinada esta y el período voluntario o forzoso en que se encuentre el deudor para efectuar dicho pago. Esta puede ser con o sin fraccionamiento.

**Aduana:** Oficina pública o institución fiscal establecida generalmente en costas y fronteras, con el fin de registrar el tráfico internacional de mercancías que se importan o exportan en y desde un país concreto y cobrar los impuestos que establezcan las aduanas. También se regula el tráfico de personas y control de capitales.

**Apremio:** Cobro forzoso que se le aplica al contribuyente cuando de forma voluntaria no se realiza la contribución del pago de la deuda al presupuesto.

**CASE:** (Computer-Aided Software Engineering) Conjunto de herramientas y métodos asociados que proporcionan asistencia automatizada en el proceso de desarrollo del software a lo largo de su ciclo de vida.

**Declaración de Mercancías:** Manifestación en la forma prescrita por la Aduana, por la que los interesados indican el régimen aduanero que se ha de aplicar a las mercancías y proporcionan los datos que la Aduana exige para la aplicación de este régimen.

**Declarante:** Toda persona natural o jurídica que hace una declaración en aduana o en nombre de la cual esta declaración es hecha.

**Despacho:** Conjunto de actos y formalidades relativos a la entrada de mercancías al territorio nacional y a la salida del mismo, que de acuerdo con los diferentes tráficos y regímenes aduaneros, deben realizar en la Aduana las autoridades aduaneras y los consignatarios, destinatarios, propietarios, poseedores o tenedores en las importaciones y los remitentes en las exportaciones, así como los agentes o apoderados aduanales.

**Despacho Comercial:** Cumplimiento de las formalidades aduaneras necesarias para exportar, importar o para colocar las mercancías bajo otro régimen aduanero.

**Documento:** Formulario de obligatorio cumplimiento con la información o datos necesarios para la formalización o trámite de determinadas operaciones, ventas o servicios, que se prestan en la gestión u objeto social de la entidad.

**Factura:** Documento económico o contable mediante el cual se determina los importes a cobrar por un servicio o venta realizada.

**Instrumento de pago:** Documento, forma o medio de pago que se emplea por las personas naturales o jurídicas, para el pago de obligaciones fiscales o como resultado de los compromisos de pagos contraídos en las relaciones monetarias mercantiles entre personas jurídicas o por la prestación de un servicio o venta.

**Nota de Crédito:** Documento en el cual el comerciante envía a su cliente, con el objeto de comunicar la acreditación en su cuenta una determinada cantidad, por el motivo expresado en la misma. Este tipo de comprobante se emite para modificar las condiciones de venta originalmente pactadas (anular operaciones, devoluciones, descuentos, bonificaciones, subsanar errores). Debe contener los mismos requisitos y característica de los comprobantes de venta que dan derecho a crédito tributario.

**Recargo por mora:** Recargo que se le aplica al contribuyente por realizar el pago de la deuda tributaria fuera de los términos establecidos en la legislación vigente.