Universidad de las Ciencias Informáticas
Facultad 3

Título

Sistema para la Administración Financiera de la Dirección Nacional de los Registros y del Notariado de la República Bolivariana de Venezuela: Diseño e Implementación del Módulo “Presupuesto”.

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

Autores

Mariam Perdomo Jorge
Handy Hernández Dalmau

Tutores

Ing. Diana Valdés González
Ing. René Queizán Pérez

Consultante

Ing. Alieski Sarmiento Almenares

Ciudad de La Habana, Junio del 2008

DECLARACIÓN DE AUTORÍA

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

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

Mariam Perdomo Jorge

Handy Hernández Dalmau

_____ _____

Firma de los Autores

Ing. Diana Valdés González

Ing. René Queizán Pérez

_____ _____

Firma de los Tutores

DATOS DE CONTACTO

Tutora: Diana Valdés González

Correo Electrónico: dvaldesg@uci.cu

Tutor: René Queizán Pérez

Correo Electrónico: rqueizan@uci.cu

Autora: Mariam Perdomo Jorge

Correo Electrónico: mpjorge@estudiantes.uci.cu

Autor: Handy Hernández Dalmau

Correo Electrónico: hdalmau@estudiantes.uci.cu

"...la sociedad que no se prepara para el uso de la computación está liquidada".

Fidel Castro Ruz

AGRADECIMIENTOS

A mis padres Marilú y Jorge por todo el amor, apoyo y confianza que siempre me han brindado, que me ayudaron a ser fuerte cuando más lo necesitaba desde que comencé en la universidad y han estado conmigo en todo momento, en cada etapa de mi vida… Ojalá ésta siempre nos mantenga juntos. Los quiero mucho…

A toda mi familia (que es bastante grande por cierto), en especial a mis primos, mis abuelos y mis tíos… Sin el apoyo de todos ustedes no hubiera podido llegar a lo que soy hoy… gracias…

A mis amigas Llilian, Olguita, Ginita y Anisbert (chicas, a pesar de la distancia siempre nos vamos a tener en el corazón). Ah! Y no se me puede olvidar aquel guajirito de Las Tunas que entró aquí con tanta mala suerte y que ahora todo se le viró al revés (Raúl)… "Nunca me olvidaré de ustedes"

A mi segunda familia de aquí de la UCI, Olga Lidia y Héctor que siempre me ayudaron en todo lo que necesité durante estos últimos cinco años de mi vida…

A mis amigas del preuniversitario Dayana, Yamila, Susel y Yenis M. que me extrañaron todo el tiempo, al igual que yo a ellas…

Al profesor Alieski Sarmiento Almenares, que siempre estuvo dispuesto cuando lo necesitamos…

A todos mis compañeros del proyecto Registro y Notarías de la Facultad 3 en especial a Yunier, Lourdes, Yaumaris, Alejandro, Maikel, Yosvany, Ismary, Armando, Henrik, Julio, Jonny, María, Kiki, que fueron testigos de mi paciencia y soportaron todos mis altercados con Handy y que nos ayudaron muchísimo.

A mis tutores y compañeros también en el proyecto, Diana y René...

A mi antiguo grupo 3301, que a pesar de habernos separado siempre estamos unidos (sobre todo a la hora de organizar "fiestas")…

A todos mis amigos en esta universidad, tanto profesores como estudiantes… todos.

Mariam Perdomo Jorge

A mis padres a quienes les debo tanto.

A papá, sin tu apoyo nada de esto hubiera sido posible, entiende de una vez que me haces mucha falta.

A mi mamá por su cariño inmenso y comprensión en todo momento, eres mi equilibrio, la conciencia que ya no tengo.

Tata, gracias por estar siempre para mí, si acabaras de venir, creo que ambos nos necesitamos y mucho, me gustaría que estuvieras aquí en este momento. A mi sobrino, Tati, por romper todos los esquemas, por tu locura, alegría y malcriadez, te pones de madre, que ganas de que crezcas jajaja.

Abuela, ojalá y la vida te dote de muchos pero muchos años más, eres para mí pieza fundamental en este rompecabezas de la vida que no logro armar.

Ore, para qué decir cuánto le agradezco a Dios por ponerte en nuestros caminos, por querer tanto a mamá que es sobre todo mi mayor tranquilidad, por tu apoyo y comprensión, eres un padre para mí.

A Alex, no importa lo que pase, no importa donde estemos en el futuro, muchacho, te me has vuelto imprescindible…

A mi familia, en especial a mi tía Mayi, me pregunto por qué el destino se aferra de pronto en unirnos más que nunca, y cuánto me alegro, mi psicóloga, no he conocido a nadie que luche tanto por lo que cree, realmente te admiro. A mi tía Marlen, y mi prima Yuli, gracias por todo lo que han hecho por mí, de algo pueden estar seguras, mi futuro tendrá forma gracias a su apoyo.

Lianyi, Yamir, tengo tanto que decir de ustedes, los quiero mucho, no creo haber pasado por tantas y tantas cosas sin ustedes a mi lado, que suerte tenerlos! A Alieski, gracias por todo, por tus consejos, y tu apoyo profesional. A Yadier, ay! a Yadier, trataré de aprender lo poco bueno que quede de ti jajaja. Lázaro mi gran amigo, sabes que puedes contar conmigo para lo que sea. Y gracias a todo el personal de Capital Humano, que son personas maravillosas.

A Mariam mi compañera de tesis por su eterna paciencia, creo que si fuera al revés la hubiera estrangulado, mejor si ella no tuviera que leer esto, jeje. Osmar, Iosmel, Maylen, Alexander, Maurys, Dole son tantas las personas que quisiera mencionar, pero señores sólo me permiten una hoja, cuánto les agradezco su amistad sincera, aunque chama no creas que se me van olvidar los panes que me comiste, con el hambre que yo tenía ese día, y el otro que me ha vuelto un adicto al café, diantres que cosas !!!!, ellos saben quiénes son, y que no me falte Alián, el erudito del apartamento.

Handy Hernández Dalmau

DEDICATORIA

A nuestros padres que siempre han estado aquí para nosotros…

A nuestras familias que nos han apoyado en todo de una forma u otra…

A todos nuestros amigos que nos acompañaron estos últimos cinco años…

A nuestros compañeros del Proyecto Registro y Notarías que nos ayudaron tanto…

A la Universidad de las Ciencias Informáticas por ser siempre “Nuestra Escuela”…

A la vida por habernos permitido disfrutar de este momento…

RESUMEN

Como parte de la necesidad que tiene la República Bolivariana de Venezuela de automatizar sus procesos registrales y notariales, lo que quedó estipulado en el nuevo Decreto Ley de Registro Público y del Notariado con fecha del 13 de noviembre del año 2001, se plantea el Proyecto de Modernización de los Registros y Notarías. Perteneciente a este proyecto se encuentra el módulo para la automatización de la Gestión del Presupuesto en cada Registro o Notaría existente en el país.

El trabajo contiene una interesante investigación sobre las técnicas y herramientas de diseño y programación, así como de la Metodología de Desarrollo de Software utilizada y la aplicación de los patrones de diseño más adecuados. Además, se da una pequeña explicación del Presupuesto en la cual se plantean los objetivos, los conceptos y las etapas de éste. También se dan a conocer los principales problemas existentes en Venezuela por lo que es necesaria esta propuesta de solución y cómo se lleva a cabo el proceso presupuestario en esa nación.

El documento incluye las etapas de Diseño e Implementación del sistema, sobre la base del Análisis que se hizo por parte de los analistas correspondientes. A su vez, el sistema que se presentará, en un futuro permitirá a la Dirección Nacional de Registros y del Notariado ejercer un control total del presupuesto que se maneja en cada Registro o Notaría y ejercer su autonomía en el país.

PALABRAS CLAVES

Presupuesto, Venezuela, Diseño, Implementación, Registros, Notarías

TABLA DE CONTENIDOS

INTRODUCCIÓN

La República Bolivariana de Venezuela como parte del proceso de cambio que está llevando en los últimos años, ha presentado una serie de reformas que comprenden a la Dirección Nacional de Registros y del Notariado perteneciente al Ministerio del Poder Popular para Relaciones Interiores y Justicia. El 13 de noviembre del año 2001, entró en vigor, en la República Bolivariana de Venezuela, el nuevo Decreto Ley de Registro Público y del Notariado con el objetivo de garantizar la seguridad jurídica, la libertad contractual y el principio de legalidad de los actos o negocios jurídicos, bienes y derechos reales, mediante la automatización progresiva de sus procesos registrales y notariales.

Anteriormente la Dirección Nacional de Registros y del Notariado no tenía el control de cómo se utilizaban los recursos recaudados por concepto de servicios registrales y notariales, pues los Registros y Notarías existentes funcionaban con total autonomía en los servicios, lo que significa que actuaban independientes de la Dirección Nacional, tanto administrativa como financieramente. La función de éstos es prestar servicios legales a la población venezolana garantizando la seguridad jurídica de los actos y de los derechos inscritos, con respecto a terceros, mediante la publicidad registral. En la actualidad, con la aprobación del Servicio Autónomo de Registros y Notarías (SAREN), se aplican cambios en la forma de trabajo que se manejaba hasta el momento, por ejemplo, toda la recaudación sería centralizada y por tanto los recursos que necesitan los registros serán destinados a partir de las necesidades establecidas por el Presupuesto. Dicho Presupuesto es ahora formulado desde la Dirección Nacional y consultado con los diferentes Registros y Notarías, logrando el control de toda la actividad financiera tanto de la recaudación como la del gasto.

Los problemas más frecuentes que se presentan en la Dirección Nacional de Registros y del Notariado actualmente y que afectan de una forma u otra la planificación del Presupuesto son:

1. No existe un sistema automatizado para la gestión del Presupuesto donde se puedan reflejar realmente los gastos e ingresos de cada uno de los Registros y Notarías.
2. A pesar de que existe un cierto grado de automatización en algunos Registros y Notarías, muchas de estas soluciones o herramientas no se integran entre sí, y por lo general están enfocadas a atender áreas específicas de la Administración Financiera, lo cual no abarca de forma integral todo el sistema Administrativo-Financiero.
3. Existen deficiencias en la gestión de las modificaciones presupuestarias, al efectuarse algún cambio, ya sea por sobreestimación o subestimación de los créditos anuales otorgados.

4. Se alteran los aranceles e impuestos fijados oficialmente por la máxima dirección para la prestación de los servicios que se brindan en cada Registro o Notaría. Estos son calculados en las oficinas, y aunque existe una ley que los respalde, cada registrador puede interpretarlos de forma diferente, como resultado, los montos para las mismas operaciones pueden ser diferentes en cada territorio.

Tomando en cuenta la situación actual en cada Registro y Notaría, se plantea la creación de una nueva Dirección Nacional de Registros y del Notariado, que supervise y controle las operaciones realizadas en ellos a través de un sistema automatizado que consolide las necesidades reales de cada uno de los Registros y Notarías a nivel de Unidades Desconcentradas y a nivel central, para realizar la planificación del Presupuesto.

Con el objetivo de solucionar las dificultades planteadas anteriormente, así como la necesidad de modernización de los Registros y Notarías; y en aras de los Convenios de Cooperación entre Cuba y Venezuela, se le asigna a la Universidad de las Ciencias Informáticas la construcción de una solución informática denominada SAREN[1] dentro de la cual se encuentra en desarrollo el sistema Administración Financiera.

La Administración Financiera está integrada por un conjunto de módulos que deben operar de forma interrelacionada, atendiendo el mandato que en ese sentido establece la Ley Orgánica de la Administración Financiera del Sector Público. El funcionamiento integrado de la Administración Financiera, permitirá a la gerencia de los entes públicos contar con la información oportuna y confiable requerida para la toma de decisiones, así como para satisfacer las demandas de información que presentan los órganos rectores de los diferentes subsistemas que lo conforman. Entre los subsistemas que contemplará el sistema integrado de gestión y control a implantar se encuentra Presupuesto, el cual está integrado por el conjunto de principios, órganos, normas y procedimientos que rige el Proceso Presupuestario Público.

El Presupuesto es un instrumento básico para la ejecución del Plan de Desarrollo Económico y Social de la nación y de Programación del Sector Público, ya que posibilita concretar en corto plazo planes y políticas de mediano y largo plazo en metas de producción de bienes y prestación de servicios, para lo cual se asignan formalmente recursos. La técnica que se aplica al Proceso Presupuestario es por disposición reglamentaria la del "Presupuesto por Proyecto", mediante la cual se organizan las funciones básicas de los organismos y entes públicos en proyectos y acciones centralizadas, cuyas

[1] Servicio Autónomo de Registros y Notarías.

acciones específicas y productos-bienes y servicios contribuyen al cumplimiento de objetivos y políticas, asignándoseles para ello los insumos reales y financieros requeridos.

El subsistema presupuestario está interrelacionado con los subsistemas de:

- **Recursos humanos**, en lo que respecta a la transmisión de la información anual necesaria para la Formulación y Programación del Presupuesto de Gastos de personal.
- **Compras y Contrataciones**, en relación con la Programación de Bienes y Servicios, y del registro presupuestario de los momentos del gasto (compromiso-causado y pago).
- **Contabilidad**, en el momento del registro del Presupuesto Anual aprobado, lo que da por iniciada la ejecución del Presupuesto. El otro punto de integración con el Subsistema Contable se produce con el registro de las Modificaciones Presupuestarias y la actualización del crédito vigente. También hay punto de vinculación cuando se registran las Transacciones Presupuestarias y cuando se efectúa el cierre.
- **Tesorería,** en el momento de realizar la operación de egreso bancario donde se registra para las partidas involucradas el momento del pagado, como parte de la ejecución del presupuesto ya que en este módulo se realizan operaciones bancarias que son parte o continuidad de un proceso de compra o contratación de servicio, por lo cual es necesario declarar el momento que definen estas operaciones, como es el pago de las obligaciones adquiridas con los proveedores o contratistas.

Hasta el momento, en Presupuesto, se han desarrollado las disciplinas de Modelación del Negocio, Levantamiento de Requisitos y las actividades correspondientes al Análisis de la disciplina Análisis y Diseño, por lo que es necesario continuar con el Proceso de Desarrollo de Software[2] para contribuir a la solución del sistema a crear.

El presente Trabajo de Diploma pretende dar solución a la **situación problémica** anteriormente expuesta, para lo cual se plantea la siguiente interrogante:

- **¿Cómo obtener un producto funcional a partir de los requerimientos identificados para el módulo "Presupuesto" del Sistema de Administración Financiera perteneciente al proyecto Registros y Notarías?**

En consecuencia, el **Objeto de Estudio** abarca el Proceso de Desarrollo de Software. El **Campo de Acción** se centra en el Diseño e Implementación del módulo "Presupuesto" del Sistema de Administración Financiera del proyecto Registros y Notarías.

[2] Conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software.

Para esto se plantea el siguiente **Objetivo General**:

- **Desarrollar el Diseño y la Implementación del módulo "Presupuesto" del Sistema de Administración Financiera perteneciente al proyecto Registros y Notarías.**

Para alcanzar la meta propuesta y orientada fundamentalmente a proveer las especificaciones del Diseño e Implementación se han derivado un conjunto de **Objetivos Específicos**, siendo éstos los siguientes:

- Obtener el Modelo de Diseño.
- Obtener el Modelo de Implementación.

En aras de lograr un trabajo con mejor calidad, se trazaron varias **Tareas**, las cuales se muestran a continuación:

- Estudio del proceso de gestión del Presupuesto en la Administración Financiera de la Dirección Nacional de Registros y del Notariado de la República Bolivariana de Venezuela.
- Estudio de las diferentes leyes que regulan la Administración Financiera de la República Bolivariana de Venezuela, como por ejemplo: La Ley Orgánica de la Administración Financiera del Sector Público, la Ley de Registro Público y del Notariado; y la Ley sobre Simplificación de Trámites Administrativos.
- Estudio de la plataforma de desarrollo y las herramientas asociadas a ésta.
- Estudio de la Metodología de Desarrollo de Software.
- Estudio de los Paradigmas de Programación.
- Estudio y selección de los Patrones de Diseño más factibles para esta propuesta de solución.
- Realización de los Diagramas de Clases del Diseño.
- Realización de los Diagramas de Secuencia.
- Realización del Diagrama de Componentes.
- Implementación de los componentes.

Según lo antes expuesto, se formula la siguiente **Hipótesis**:

- Si se desarrolla el Diseño y la Implementación del módulo Presupuesto del sistema de Administración Financiera del proyecto Registros y Notarías entonces se obtendrá un producto funcional que cumpla con los requerimientos identificados en éste.

El siguiente trabajo consta de tres capítulos:

El Capítulo 1 expone diferentes conceptos de Presupuesto, los objetivos de éste y las etapas por las que pasa una empresa u organización anualmente para la elaboración del plan de presupuesto. Más

enfocado en Venezuela, se cuenta con una explicación del Proceso Presupuestario en ese país y la significación de poseer un Sistema de Administración Financiera Integrada. Se presentan también las principales tendencias y tecnologías actuales en el desarrollo de aplicaciones informáticas.

El Capítulo 2 aborda la arquitectura del sistema y las capas por las que está compuesta. También se explican las diferentes clases que existen en el diseño de la solución, los Patrones de Diseño y se muestran los Diagramas de Clases y de Secuencia de los casos de uso fundamentales dentro del sistema. Se muestran los componentes desarrollados y sus ventajas así como reglas a seguir para la codificación. Además, se incluye el Modelo de Datos, el Diagrama de Despliegue y el de Componentes.

En el Capítulo 3 se hace referencia a la validación de la solución presentándose una serie de casos de prueba como parte de las Pruebas de Caja Negra y se analizan algunos fragmentos de código donde se les aplica la Métrica de la Complejidad Ciclomática definiendo el grado de complejidad del sistema como parte de las Pruebas de Caja Blanca.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se exponen diferentes conceptos de Presupuesto, así como sus objetivos y las etapas por las que pasa éste para la elaboración del plan anual en una empresa u organización. Además, se muestra cómo tiene lugar el Proceso Presupuestario en la República Bolivariana de Venezuela. Se mencionan también las diferentes tendencias y tecnologías actuales en el desarrollo de aplicaciones informáticas.

1.1. Presupuesto

1.1.1. Diferentes conceptos de Presupuesto

- Se puede definir el Presupuesto como un plan de acción dirigido a cumplir una meta pronosticada, expresada en valores y términos financieros que, debe cumplirse en determinado tiempo y bajo ciertas condiciones previstas. (Córdova, 2007)
- Un Presupuesto puede definirse como la presentación ordenada de los resultados previstos de un plan, un proyecto o una estrategia. (Campeche, 2007)
- El Presupuesto es la expresión cuantitativa de un plan de acción y una ayuda a la coordinación y la ejecución. Los presupuestos se pueden crear para la organización en general o para cualquier subunidad. El presupuesto maestro resume los objetivos de todas las subunidades de una organización. (Desoft.SA, 2007)

1.1.2. Objetivos del Presupuesto

Los objetivos del Presupuesto se resumen en la Previsión, Planeación, Organización, Coordinación o Integración, Dirección y Control de fondos. (Fonteboa Vizcaíno, 2006)

- Objetivo de Previsión

Está dirigido a prever lo necesario, tener anticipadamente todo lo conveniente para la elaboración y la ejecución del presupuesto, para atender a tiempo las necesidades posibles. Es necesario realizar un análisis completo de las problemáticas que rodean a la entidad en cuanto a su entorno de funcionalidad y desempeño para que este mecanismo de previsión tenga buenas bases y así no se frustre el cumplimiento de los objetivos de la entidad y el control de sus recursos debido a una mala ejecución del presupuesto

- Objetivo de Planeación

El mismo estará encaminado a qué y cómo se va a hacer. Se establecen los objetivos de la empresa y la organización necesaria para alcanzarlos y se prevé la planificación unificada y sistematizada de las posibles acciones, en concordancia con los objetivos. Cada proceso de planificación tiene que ir acompañado de una adecuada fundamentación que demuestre que lo solicitado es la base esencial de una actividad determinada, y que es imprescindible para el logro de los objetivos de la entidad.

- Objetivo de Organización

Responde a quién lo hará, comprende la estructuración técnica de las relaciones que deben existir entre las funciones, los niveles, y las actividades de los elementos materiales y humanos de una entidad, con el fin de lograr su máxima eficiencia dentro de los planes y los objetivos señalados. Todo ello permite que exista una adecuada, precisa y funcional estructura y desarrollo de la entidad; lo que conlleva a que la misma logre de una manera real la interrelación funcional de cada una de las áreas y desempeño objetivo en las mismas, evitando dualidad de funciones o que algunas de las actividades caigan en terreno de nadie.

- Objetivo de Coordinación o Integración

Incluye lo que se haga y se forme en orden de lo particular y lo general respecto al desarrollo, mantenimiento, y consecución armoniosa de las actividades de la entidad, con el fin de evitar situaciones de desequilibrio, entre las diferentes secciones que integran la organización. Esto también demuestra la interrelación con algunas de las normas del control interno tales como: integridad, confianza mutua, asignación de autoridad y responsabilidad, coordinación entre áreas, etc.

- Objetivo de Dirección

Este objetivo trata de guiar el camino o ratio de éste para que se haga, es de gran ayuda en el cumplimiento de las políticas a seguir, tomas de decisiones y visión de conjunto, también constituye bases para la conducción y guía de los subordinados. Si comenzamos a dirigir el presupuesto en interés no planificado anteriormente, todo ello conllevaría no sólo a violar fundamentos anteriores que sustentaron el monto aprobado, sino que perdemos la línea de dirección por objetivos y valores.

- Objetivo de Control

En este se revisa que se realice la acción por medio de la cual se aprecia si los planes y los objetivos se están cumpliendo, establecen la comparación a tiempo entre lo presupuestado y los resultados habidos, dando lugar a diferencias analizables, para realizar superaciones y correcciones de haber consecuencias. De igual forma logra la fundamentación de los presupuestos posteriores, ya que busca

la funcionalidad en que se han llevado a cabo los anteriores, para prevenir de antemano los nuevos retos y alternativas de los presupuestos subsiguientes.

1.1.3. Etapas del Proceso de Planificación Presupuestaria

De manera general, se puede afirmar que el proceso de Planificación Presupuestaria de cualquier empresa u organización es un proceso secuencial y que está integrado por las siguientes etapas: (Campeche, 2007)

La primera es la ***definición y transmisión de las directrices generales a los responsables de la preparación de los presupuestos***. En esta etapa, la dirección general, o la dirección estratégica, es la responsable de transmitir a cada área de actividad las instrucciones generales, para que éstas puedan diseñar sus planes, programas, y presupuestos; ello es debido a que las directrices fijadas a cada área de responsabilidad, o área de actividad, dependen de la planificación estratégica y de las políticas generales de la empresa u organización fijadas a largo plazo.

La segunda es la ***elaboración de planes, programas y presupuestos***. En esta etapa, a partir de las directrices recibidas, y aceptadas, cada responsable elaborará el presupuesto considerando las distintas acciones que deben emprender para poder cumplir los objetivos marcados. Sin embargo, conviene que al preparar los planes correspondientes a cada área de actividad, se planteen distintas alternativas que contemplen las posibles variaciones que puedan producirse en dichos planes.

La tercera es la ***negociación de los presupuestos***. La negociación es un proceso que va de abajo hacia arriba, en donde, a través de fases iterativas sucesivas, cada uno de los niveles jerárquicos consolida los distintos planes, programas y presupuestos aceptados en los niveles anteriores.

La cuarta es la ***coordinación de los presupuestos***. A través de este proceso se comprueba la coherencia de cada uno de los planes y programas, con el fin de introducir, si fuera necesario, las modificaciones necesarias y así alcanzar el adecuado equilibrio entre las distintas áreas.

La quinta es la ***aprobación de los presupuestos***. Esta consiste en la aprobación, por parte de la dirección general, de las previsiones que han ido realizando los distintos responsables. Supone evaluar los objetivos que pretende alcanzar la entidad a corto plazo, así como los resultados previstos en base de la actividad que se va a desarrollar.

La sexta y última etapa es la de ***seguimiento y actualización de los presupuestos***. Una vez aprobado el presupuesto es necesario llevar a cabo un seguimiento o un control de la evolución de cada una de las variables que lo han configurado y proceder a compararlo con las previsiones. Este

seguimiento permitirá corregir las situaciones y actuaciones desfavorables, y fijar las nuevas previsiones que pudieran derivarse del nuevo contexto.

1.2. Aspectos generales del Proceso Presupuestario en Venezuela.

La planificación del desarrollo económico y social constituye una de las responsabilidades fundamentales del sector público; el sistema de planificación está constituido por diversos instrumentos, cada uno de los cuales cumple una función específica, complementaria por los demás; y que, dentro de ellos, al Plan Operativo Anual[3] le corresponde la concreción de los planes de largo y mediano plazo.

Uno de los componentes del Plan Operativo Anual es el Presupuesto del Sector Público, a través del cual se procura la definición concreta y la materialización de los objetivos de dicho sector. La concepción moderna del presupuesto está sustentado en el carácter de integridad de la técnica financiera, ya que el presupuesto no sólo es concebido como una mera expresión financiera del plan de gobierno, sino como una expresión más amplia pues constituye un instrumento del sistema de planificación, que refleja una política presupuestaria única.

Bajo este enfoque de la integridad se sustenta la necesidad de que las diversas etapas del proceso presupuestario, sean concebidas como aspectos igualmente importantes del sistema presupuestario y, por lo tanto, estén debidamente coordinados. (Gobierno Bolivariano de Venezuela, 2007)

El alcance del proceso presupuestario público se puede resumir en:

a) Establecer los nexos entre las metas y los objetivos de los planes institucionales, así como medir la contribución de la gestión al logro de las metas de desarrollo económico, social e institucional del país.
b) Permitir la más eficiente asignación, uso y evaluación de los recursos reales y financieros que demanda el cumplimiento de objetivos y metas.
c) Brindar información en cada una de las etapas del proceso presupuestario sobre las variables reales y financieras, a efectos de que la toma de decisiones considere ambas variables.
d) Facilitar la interrelación con los demás subsistemas de Administración Financiera y permitir el ejercicio del control interno y externo.

Esto está a cargo de la Oficina Nacional de Presupuesto (ONAPRE), cuyas funciones principales, en su condición de órgano rector y en el contexto del Sistema Integrado de Administración Financiera son:

[3] Instrumento de Gestión de apoyo a la acción pública, que contiene las directrices a seguir: áreas estratégicas, programas, proyectos, recursos y sus respectivos objetivos y metas, así como la expresión financiera para acometerlas.

- Participar en la preparación del proyecto de ley del marco plurianual del presupuesto del sector público nacional.
- Elaborar el proyecto de Ley de Presupuesto.
- Participar en la elaboración del Plan Operativo Anual y preparar el presupuesto consolidado del sector público.
- Preparar y dictar las normas e instrucciones técnicas relativas al desarrollo de las diferentes etapas del proceso presupuestario.
- Analizar y aprobar, cuando corresponda, las solicitudes de modificaciones presupuestarias.
- Evaluar la ejecución de los presupuestos aplicando las normas y criterios establecidos en las disposiciones legales y las normas técnicas respectivas.

En cada Ente Descentralizado Funcionalmente sin Fines Empresariales[4], la unidad de presupuesto es la responsable de coordinar internamente la aplicación del proceso presupuestario, en acatamiento de las normas e instrucciones que a tales efectos dicte la ONAPRE. (Ministerio de Finanzas, República Bolivariana de Venezuela, 2002)

1.3. Sistema de Administración Financiera Integrada

Se entiende por Sistema de Administración Financiera el conjunto de leyes, normas y procedimientos destinados a la obtención, asignación, uso, registro y evaluación de los recursos financieros del Estado, que tiene como propósito la eficiencia de la gestión de los mismos para la satisfacción de las necesidades colectivas.

Un Sistema de Administración Financiera está integrado cuando las unidades organizativas y entes que lo conforman actúan en forma interrelacionada bajo la dirección de un único órgano coordinador que debe tener la suficiente competencia para reglamentar el funcionamiento de dichas unidades, y cuando el conjunto de principios, normas y procedimientos vigentes en el sistema, son coherentes entre sí y permiten interrelacionar automáticamente sus actividades.

Las unidades organizativas que en el sector público integran la Administración Financiera, son las responsables de programar y evaluar el presupuesto, administrar el sistema de recaudación tributaria y aduanera, gestionar las operaciones de crédito público, programar la ejecución de los presupuestos de gastos, administrar el Tesoro y contabilizar tanto física como financieramente las transacciones relacionadas con la captación y colocación de los fondos públicos. Los recursos humanos, materiales y

[4] Aquellos entes que no realizan actividades de producción de bienes o servicios destinados a la venta, y cuyos ingresos o recursos provengan fundamentalmente del presupuesto de la República tales como los institutos autónomos, las personas jurídicas estatales de derecho público, las fundaciones, asociaciones civiles y demás instituciones constituidas con fondos públicos.

financieros que demanden el funcionamiento de estas unidades, también forman parte de la Administración Financiera. (Ministerio de Finanzas, República Bolivariana de Venezuela, 1999)

Los objetivos generales de un Sistema Integrado de Administración Financiera están referidos a: (Ministerio de Finanzas, República Bolivariana de Venezuela, 2002)

- Mejorar la formulación presupuestaria, conforme a los planes operativos del sector público nacional
- Optimizar los Flujos de Caja de la Tesorería Nacional y de los propios entes, mediante la adecuada programación de la ejecución financiera de ingresos y egresos.
- Jerarquizar e integrar la función contable de la República y sus Entes Descentralizados Funcionalmente, para unificar criterios de administración financiera orientados a garantizar información oportuna de la gestión administrativa, al control presupuestario, al soporte de la gestión, la rendición de cuentas y el control fiscal.
- Garantizar el cumplimiento de normas y procedimientos

1.4. Solución informática existente.

1.4.1. Sistema Integrado de Gestión y Control de las Finanzas Públicas (SIGECOF)

Este sistema ha sido concebido para diseñar, desarrollar e implantar un Sistema de Administración Financiera Integrada que abarque todos los componentes financieros básicos del Gobierno Nacional. En este sentido corresponden:

- Sistema de Presupuesto
- Sistema de Crédito Público
- Sistema de Tesorería
- Sistema de Contabilidad
- Sistema de Gestión Financiera de Recursos Humanos
- Sistema de Gestión de Bienes
- Sistema de Compras

Los procesos administrativos que ejecutan los organismos públicos afectan a uno o varios de estos sistemas simultáneamente. (Ministerio de Finanzas, República Bolivariana de Venezuela, 1999)

El sistema debe garantizar: (Ministerio de Finanzas, República Bolivariana de Venezuela, 2002)

Confiabilidad: Ofrecer certeza de los datos, los hechos y las cifras.

Unicidad: Proporcionar un tratamiento y registro único de los datos en el sitio más cercano donde ocurre cada transacción.
Integridad: Cubrir la totalidad de las operaciones financieras en el ámbito presupuestario y no presupuestario.
Verificabilidad: Favorecer o posibilitar el seguimiento de una pista auditable de todas las transacciones.
Oportunidad: Posibilitar la obtención de la información sobre transacciones realizadas en el momento en que se producen.
Utilidad: Apoyar el proceso de toma de decisiones y la simplificación de procedimientos.
Transparencia: Dar información clara y pública sobre la gestión económica financiera del Gobierno.
Seguridad: Proteger física y lógicamente dicha información contra el acceso no autorizado y el fraude.

1.5. Paradigmas de Programación

1.5.1. Programación Orientada a Objetos

La Programación Orientada a Objetos es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Es una forma especial de programar, más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación.
El elemento básico de este paradigma no es la función (elemento básico de la programación estructurada), sino un ente denominado objeto. Un objeto es la representación de un concepto para un programa, y contiene toda la información necesaria para abstraer dicho concepto: los datos que describen su estado y las operaciones que pueden modificar dicho estado, y determinan las capacidades del objeto.
Existen una serie de principios fundamentales para comprender cómo se modela la realidad al crear un programa bajo el paradigma de la orientación a objetos. Estos principios son: la abstracción, el encapsulamiento, la modularidad, la jerarquía, el paso de mensajes y el polimorfismo.

1. **Principio de Abstracción**

Mediante la abstracción, la mente humana modela la realidad en forma de objetos. Para ello busca parecidos entre la realidad y la posible implementación de objetos del programa que simulen el funcionamiento de los objetos reales. Significa extraer las propiedades esenciales de un objeto que lo distinguen de los demás tipos de objetos y proporciona fronteras conceptuales definidas respecto al punto de vista del observador. Es la capacidad para encapsular y aislar la información de diseño y ejecución.

2. Principio de Encapsulamiento

El encapsulamiento permite a los objetos elegir qué información es publicada y qué información es ocultada al resto de los objetos. Para ello los objetos suelen presentar sus métodos como interfaces públicas y sus atributos como datos privados e inaccesibles desde otros objetos.
Para permitir que otros objetos consulten o modifiquen los atributos de los objetos, las clases suelen presentar métodos de acceso. De esta manera el acceso a los datos de los objetos es controlado por el programador, evitando efectos laterales no deseados.
Con el encapsulado de los datos se consigue que las personas que utilicen un objeto sólo tengan que comprender su interfaz, olvidándose de cómo está implementada, y en definitiva, reduciendo la complejidad de utilización.

3. Principio de Modularidad

Mediante la modularidad, se propone al programador dividir su aplicación en varios módulos diferentes (ya sea en forma de clases, paquetes o bibliotecas), cada uno de ellos con un sentido propio.
Esta fragmentación disminuye el grado de dificultad del problema al que da respuesta el programa, pues se afronta el problema como un conjunto de problemas de menor dificultad, además de facilitar la comprensión del programa.

4. Principio de Jerarquía

La mayoría de nosotros ve de manera natural nuestro mundo como objetos que se relacionan entre sí de una manera jerárquica. Por ejemplo, un perro es un mamífero, y los mamíferos son animales, y los animales seres vivos. Del mismo modo, las distintas clases de un programa se organizan mediante la jerarquía. La representación de dicha organización da lugar a los denominados árboles de herencia.

5. Principio del Paso de Mensajes

Mediante el denominado paso de mensajes, un objeto puede solicitar de otro objeto que realice una acción determinada o que modifique su estado. El paso de mensajes se suele implementar como llamadas a los métodos de otros objetos. Desde el punto de vista de la programación estructurada, esto correspondería con la llamada a funciones.

6. Principio de Polimorfismo

Polimorfismo quiere decir "un objeto y muchas formas". Esta propiedad permite que un objeto presente diferentes comportamientos en función del contexto en que se encuentre. Por ejemplo un método puede presentar diferentes implementaciones en función de los argumentos que recibe, recibir

diferentes números de parámetros para realizar una misma operación, y realizar diferentes acciones dependiendo del nivel de abstracción en que sea llamado.

7. Principio de la Herencia

Es la propiedad que permite a los objetos construirse a partir de otros objetos. La clase base contiene todas las características comunes. Las sub-clases contienen las características de la clase base más las características particulares de la sub-clase. Si la sub-clase hereda características de una clase base, se trata de herencia simple, si hereda de dos o más clases base, herencia múltiple.
(Universidad de Burgos, 1999) (Teso)

1.5.2. Programación Orientada a Eventos

Los lenguajes visuales extienden las posibilidades de los lenguajes convencionales incorporando elementos nuevos, que poseen un comportamiento predefinido, orientados a facilitar el diseño de la interfaz de la aplicación. Estos elementos, también llamados componentes[5], pueden modificarse tanto en la apariencia como en la respuesta esperada al interactuar con ellos.

El uso de herramientas que combinen lo visual con lo algorítmico llevó a cambiar el estilo de Programación Estructurada convencional hacia lo que se conoce como Programación Orientada a Eventos[6], la cual gira en torno al momento en que las señales del mundo real ocurren (Lanzarini). Este tipo de programación es un poco más complicada que la Secuencial pero con los lenguajes visuales de hoy, se hace sencilla y agradable (Orellana, 2006). En la Programación Secuencial es el programador el que define cuál va a ser el flujo del programa mientras que en la Dirigida por Eventos será el propio usuario el que dirija este flujo.

Dentro de este nuevo paradigma, el programador podrá incorporar un botón a su aplicación, darle la apariencia deseada e indicar, por ejemplo, qué hacer al dar clic sobre él. También puede centrar más su atención al análisis y al diseño de la solución, así como a la selección de las estructuras de datos adecuadas al problema. Es importante destacar que el aspecto visual es manejado ahora por el lenguaje de programación. Esto último permite reducir el tiempo de desarrollo, ya que no sólo resuelve el problema de la interfaz con el usuario, sino que facilita el mantenimiento del software y reduce errores (Lanzarini).

Los Programas Orientados a Eventos son los programas típicos de Windows, tales como Word, Excel, PowerPoint y otros. Cuando uno de estos programas ha arrancado, lo único que hace es quedarse a la

[5] Un componente es un objeto particular que puede ser reutilizado en diferentes contextos.
[6] Un evento es un hecho que se produce en un momento dado bajo ciertas condiciones y puede desencadenar reacciones. Constituye las acciones que pueda ejecutar un usuario sobre el programa que está utilizando en ese momento.

espera de las acciones del usuario, que en este caso son llamadas eventos. El usuario dice si quiere abrir y modificar un fichero existente, o bien comenzar a crear un fichero desde el principio. Estos programas pasan la mayor parte de su tiempo esperando las acciones del usuario y respondiendo a ellas. (Orellana, 2006)

Ejemplos de eventos son: el clic hecho sobre un botón, el ingreso de un valor por parte del usuario, la selección de una opción dentro de un menú desplegable, el clic hecho sobre un archivo para abrirlo, arrastrar un ícono, pulsar una tecla o combinación de ellas o simplemente mover el mouse, entre otros.

1.5.3. Programación Orientada a Componentes

La Programación Orientada a Componentes es una variante natural de la Orientada a Objetos con el objetivo de construir un mercado global de componentes cuyos usuarios son los propios desarrolladores de aplicaciones que necesitan reutilizar componentes ya hechos y probados para construir sus aplicaciones de una forma más rápida. (Navarro, y otros)

Denominamos Componente a la unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio. (Navarro, y otros)

Existen tres tecnologías de componentes predominantes en la actualidad e incompatibles entre sí (Universidad de Jaén, 2008):

1. VBX/OCX/ActiveX de Microsoft: Los VBX surgieron como una forma de poder distribuir controles entre los desarrolladores de Visual Basic. Visual Basic puede ser considerado como el primer Entorno de Desarrollo Orientado a Componentes que tuvo realmente éxito y aceptación. Los OCX y ActiveX son evoluciones posteriores de los VBX.
2. VCL de Borland: Es la tecnología de componentes utilizados por los entornos de desarrollo Delphi y C++ Builder. Curiosamente estos dos entornos también permiten la utilización de componentes OCX y ActiveX.
3. JavaBeans de Sun: Es la tecnología de componentes basada en Java. En principio no permite la utilización de otro lenguaje de programación. Al contrario que las otras tecnologías, JavaBeans funciona en cualquier plataforma.

Características de los componentes (Universidad de Jaén, 2008) (Navarro, y otros)

1. Constituyen una unidad software compilada reutilizable, con una interfaz bien definida.

2. Se distribuyen en un único paquete instalable que contiene en sí todo lo necesario para su funcionamiento, con ninguna o muy pocas dependencias con otros componentes o librerías.
3. Pueden estar implementados en cualquier lenguaje de programación (aunque los Orientados a Objetos son los más adecuados), y ser utilizados también para el desarrollo en cualquier lenguaje de programación. Se ejecutan en diferentes plataformas y sistemas operativos.
4. Pueden constituir un producto comercial de calidad, realizado por un fabricante especializado. Por supuesto pueden existir componentes gratuitos.
5. Pueden ser modificables y sujetos a evolución por ampliación, desaparición, sustitución de componentes o reconfiguración de las relaciones entre ellos.

1.6. Proceso de Desarrollo de Software

Cada año se producen en el mercado computadoras más potentes y eficaces, con grandes capacidades de almacenamiento y velocidad. Debido a esto, la tendencia actual en el desarrollo de software hace que aparezcan sistemas más grandes y más complejos y que por lo tanto, los usuarios esperen más de ellos. Esta tendencia también se ve afectada por el auge que ha tenido Internet en los últimos años en el intercambio de todo tipo de información, ya sean fotos, textos, multimedia, etc. Los usuarios quieren ver productos mejorados, adaptados a sus necesidades y que sean cada vez más rápidos. La comunidad de desarrolladores necesita una forma coordinada de trabajar, una forma de que otros entiendan su trabajo sin haber participado directamente en él, necesita un proceso que integre las múltiples facetas de desarrollo, necesita un método común. (Jacobson, y otros, 2004 pág. 3)

Un Proceso de Desarrollo de Software es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software. (Jacobson, y otros, 2004 pág. 4) (Véase Figura 1.1).Un proceso define *quién* está haciendo *qué, cuándo* y *cómo* alcanzar un determinado objetivo que en ingeniería de software sería construir un producto software o mejorar uno existente. Es necesario que este sirva como guía común para todos los participantes (clientes, usuarios, desarrolladores y directores ejecutivos), que sea nuevo, que sea el mejor.

Un proceso que es efectivo proporciona normas para el desarrollo eficiente de un producto de calidad además de capturar y presentar las mejores prácticas que el estado actual de las tecnologías permite, también reduce el riesgo y hace al proyecto más predecible. Como consecuencia de esto se crea un método común. (Jacobson, y otros, 2004)

Figura 1.1

1.7. Metodología de Desarrollo de Software

1.7.1. Rational Unified Process (RUP)

El Proceso Unificado de Desarrollo de Software (RUP) es más que un simple proceso, es un marco de trabajo genérico que puede aplicarse a una gran variedad de sistemas software ya sean para diferentes áreas de aplicación, niveles de aptitud, tipos de organizaciones como para diferentes tamaños de éstos.
(Jacobson, y otros, 2004)

RUP se sostiene sobre tres ideas básicas:

1. **Dirigido por Casos de Uso:** En RUP, los casos de uso se utilizan para capturar los requisitos funcionales y para definir los contenidos de las iteraciones. La idea es que cada iteración tome un conjunto de casos de usos y desarrolle todo el camino a través de las distintas disciplinas: diseño, implementación, prueba, etc.
2. **Centrado en la Arquitectura:** RUP asume que no existe un modelo único que cubra todos los aspectos del sistema. Por dicho motivo existen múltiples modelos y vistas que definen la arquitectura de software de un sistema. La analogía con la construcción es clara, cuando se construye un edificio existen diversos planos que incluyen los distintos servicios del mismo: electricidad, fontanería, etc. (Molina Díaz, 2007)
3. **Iterativo e Incremental:** RUP se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo concluye con una versión del producto y cada versión es un producto preparado para su entrega. Cada ciclo consta de cuatro fases: Inicio, Elaboración, Construcción y Transición, y estas a su vez se dividen en iteraciones. Cada iteración pasa por los diferentes flujos de trabajo: Modelación del Negocio; Requerimientos; Análisis y Diseño; Implementación; Prueba; Instalación; Configuración y Administración de Cambios; Administración del proyecto y por último, Ambiente (Véase Figura. 1.2). Aunque todas las iteraciones suelen incluir trabajo en casi todas las

disciplinas, el grado de esfuerzo dentro de cada una de ellas varía a lo largo del proyecto. (Jacobson, y otros, 2004)

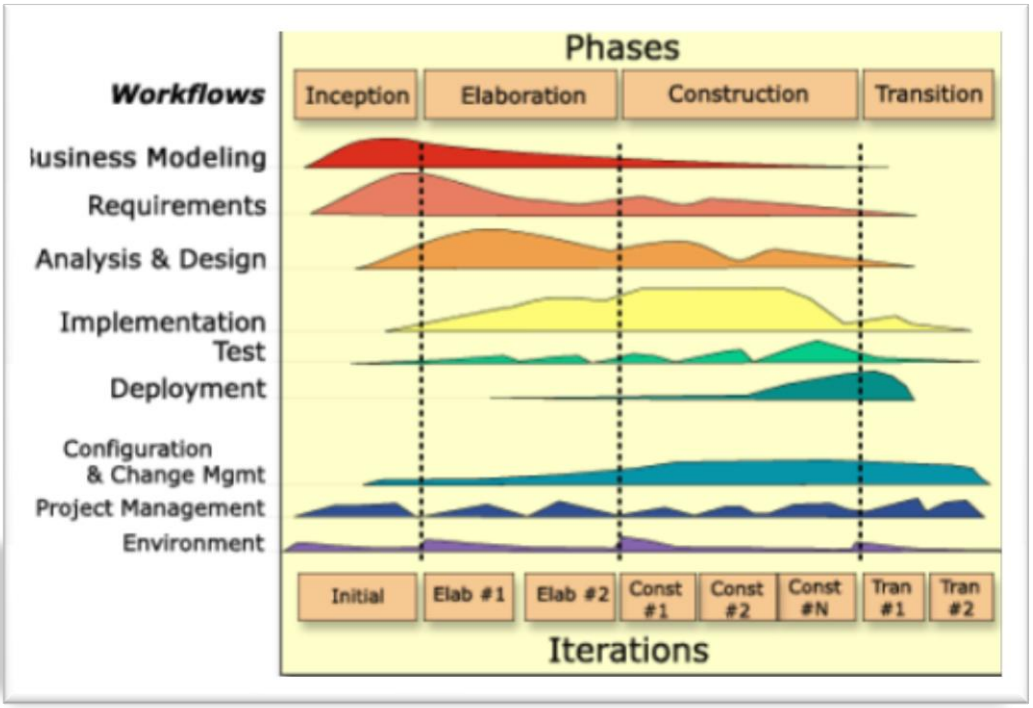


Figura 1.2. Las 4 fases de RUP con los diferentes Flujos de Trabajo e iteraciones

El Proceso Unificado está basado en componentes y utiliza el estándar de modelado visual UML (Lenguaje Unificado de Modelado) para preparar todos los esquemas de un sistema de software, de hecho, UML es una parte esencial del Proceso Unificado (Jacobson, y otros, 2004). Rational Rose es la herramienta CASE[7] que comercializan los desarrolladores de UML y que soporta de forma completa la especificación del UML (Martínez).

1.8. Artefactos y Trabajadores de la Disciplina de Diseño

(Rational Software Corporation, 2003)

La disciplina de Diseño, perteneciente al Flujo de Trabajo de Análisis y Diseño, tiene como propósitos transformar los requerimientos en un diseño del sistema a crear; definir y desarrollar una arquitectura robusta para el sistema y crear una entrada apropiada y un punto de partida para actividades de implementación.

La relación que tiene con el Flujo de Trabajo de Requerimientos es que éste constituye su entrada primaria y con el de Implementación es que éste implementa precisamente ese diseño que se conciba.

[7] Computer Aided Software Engineering (Ingeniería de Software asistida por Ordenador): Son aquellas aplicaciones informáticas que tienen como objetivo lograr una mayor productividad en el desarrollo de software y una reducción de costos de desarrollo.

A continuación, en la Figura 1.3 se muestran los Artefactos y Trabajadores del Flujo de Trabajo Análisis y Diseño, siendo los más importantes en el Diseño los que se mencionan en el epígrafe siguiente.

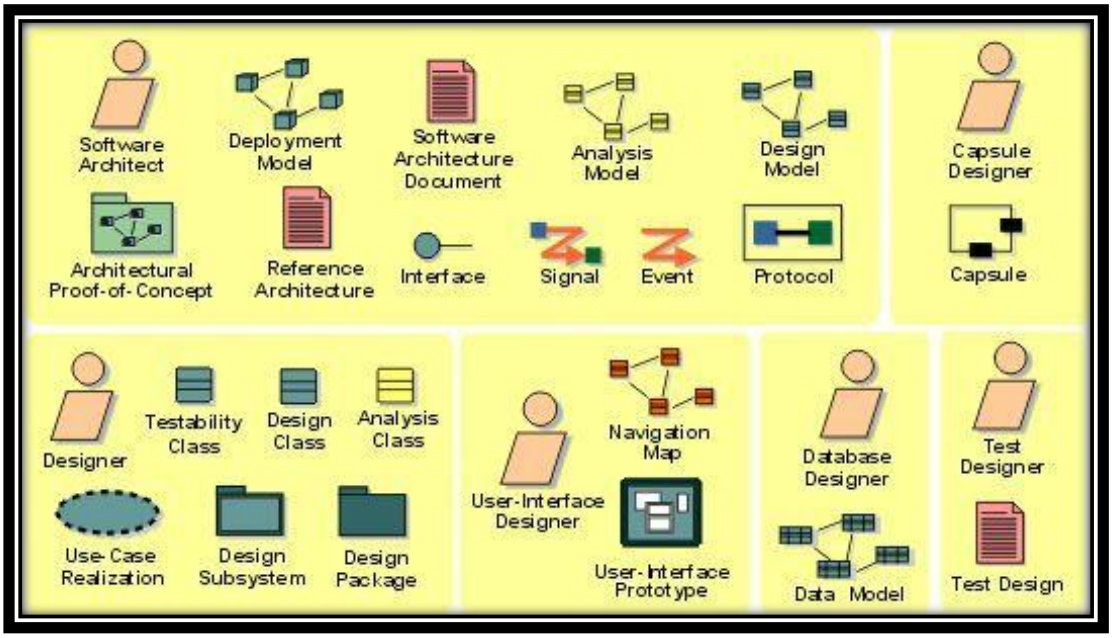


Figura 1.3. Artefactos y Trabajadores del Flujo de Trabajo Análisis y Diseño

1.8.1. Principales Artefactos

1. Modelo de Diseño

El Modelo de Diseño es un modelo de objetos que describe la realización de casos de uso y sirve como abstracción del Modelo de Implementación y del código fuente. Constituye una entrada esencial a las actividades en Implementación y Prueba.

Ante todo, establece la arquitectura, pero también es usado como un vehículo de análisis durante la fase de Elaboración pero se refina mediante un diseño detallado durante la fase de Construcción. El Arquitecto de Software es el responsable de la integridad de este modelo, pero no de los paquetes, clases, relaciones, diagramas y realizaciones de casos de uso.

2. Documento de Arquitectura de Software

El Documento de Arquitectura de Software provee una vista comprensiva de la arquitectura del sistema, utilizando 4+1 vistas arquitectónicas para representar diferentes aspectos de éste. Las vistas son las siguientes*: Vista de Casos de Uso, Lógica, Procesos, Implementación y Despliegue*; pero todas están regidas por la de Casos de Uso.

Es desarrollado fundamentalmente durante la fase de Elaboración ya que uno de los objetivos de ésta es establecer una arquitectura sólida. El Arquitecto de Software es el responsable de este documento, además, de mantenerlo actualizado.

3. Modelo de Despliegue

El Modelo de Despliegue muestra la configuración de los nodos procesadores en tiempo de ejecución, los enlaces de comunicación entre ellos y los componentes y objetos que residen en ellos. Consta de uno o varios *nodos* (elementos de procesamiento con al menos un procesador, memoria y posiblemente otros dispositivos); *dispositivos* (nodos estereotipados sin capacidad de procesamiento) y *conectores* (expresan las conexiones entre los nodos y entre éstos y los dispositivos). El Arquitecto de Software es el responsable de este modelo.

4. Clases del Diseño

Las Clases del Diseño son una descripción de un conjunto de objetos que comparten las mismas responsabilidades, relaciones, operaciones, atributos, y semántica. Forman parte del Modelo de Diseño y fundamentales en un enfoque orientado a objetos. Las Clases del Diseño Arquitectónicamente significativas son identificadas y descritas durante la fase de Elaboración, las restantes, durante la de Construcción. El Diseñador es el responsable de la integridad de la clase.

5. Realizaciones de Casos de Uso

Una Realización de Casos de Uso describe cómo un caso de uso particular es realizado dentro del Modelo de Diseño, en términos de colaboración de objetos. Forman parte del Modelo de Diseño y son creadas en la fase de Elaboración para los casos de uso arquitectónicamente más significativos, los restantes, son realizados en la de Construcción. Presentan una descripción de flujos de eventos textual, diagramas de clases y diagramas de interacción, etc. El Diseñador es el responsable de estas realizaciones.

6. Subsistemas de Diseño

Un Subsistema de Diseño es una parte de un sistema que encapsula comportamiento, expone un conjunto de interfaces, paquetes y otros elementos del modelo. Desde afuera, un subsistema es un elemento sencillo del Modelo de Diseño que colabora con otros elementos del modelo para realizar sus responsabilidades. Las interfaces visibles externamente y su comportamiento se conoce como subsistema de especificación. En el interior, un subsistema es una colección de elementos del modelo (clases del diseño y otros subsistemas) que realizan las interfaces y el comportamiento del subsistema de especificación. Esto se conoce como subsistema de realización. Forma parte del Modelo de Diseño y es precisamente el Diseñador el responsable de su integridad, no el Arquitecto.

7. **Paquetes de Diseño**

Un Paquete de Diseño es una colección de clases, relaciones, realizaciones de casos de usos, diagramas y otros paquetes. Es usado para estructurar el Modelo de Diseño dividiéndolo en partes más pequeñas. Deben usarse fundamentalmente como herramienta organizacional para agrupar elementos relacionados, si se requiere un comportamiento o semántica se deben usar subsistemas. Forma parte del Modelo de Diseño y el Diseñador es el responsable de su integridad, pero dentro de él, sólo de las Clases del Diseño.

8. **Modelo de Datos**

El Modelo de Datos describe las representaciones lógicas y físicas de los datos persistentes usados por la aplicación. En casos donde ésta utilice un Sistema de Gestión de Bases de Datos Relacional (SGBD), el Modelo de Datos puede incluir elementos como procedimientos almacenados, triggers, restricciones (constraints), etc., que definen la interacción de los componentes de la aplicación con el SGBD. El Diseñador de Bases de Datos es el responsable de la integridad del modelo, asegurando que sea completamente correcto, entendible y consistente.

9. **Diagramas de Interacción**

Los Diagramas de Interacción se utilizan para modelar los aspectos dinámicos de un sistema. Muestran una interacción que consiste en un conjunto de objetos y sus relaciones, incluyendo los mensajes que se pueden enviar entre ellos. Existen dos tipos: Los Diagramas de Secuencia y los de Colaboración. El primero destaca la ordenación temporal de los mensajes y el segundo destaca la organización estructural de los objetos que envían y reciben mensajes.

1.8.2. Trabajadores más importantes

1. **Arquitecto de Software**

Es el responsable de la arquitectura del software incluyendo las decisiones técnicas que restringen el diseño y la implementación del proyecto. Debe identificar y documentar los aspectos arquitectónicamente significativos de las vistas de requerimientos, diseño, implementación y despliegue. Debe poseer las habilidades de experiencia en el dominio del problema, liderazgo, buena comunicación y estar bien enfocado en las metas del proyecto.

2. **Diseñador**

Es el responsable de diseñar una parte del sistema cumpliendo con las restricciones de los requerimientos, la arquitectura y el proceso de desarrollo para el proyecto. Identifica y define las responsabilidades, operaciones, atributos y relaciones de los elementos de diseño. Debe asegurarse

de que el diseño sea consistente con la arquitectura y que esté detallado al punto que se pueda proseguir con la implementación.

3. Diseñador de Base de Datos

Es el responsable de diseñar el almacenamiento de los datos persistentes que se usarán en el sistema. Debe definir el diseño detallado de la base de datos incluyendo tablas, índices, vistas, restricciones, triggers, procedimientos almacenados y cualquier otro elemento que se necesite para almacenar, recuperar y eliminar objetos persistentes.

1.9. Artefactos y Trabajadores de la Disciplina de Implementación

(Rational Software Corporation, 2003)
El Flujo de Trabajo de Implementación tiene como propósitos definir la organización del código en términos de subsistemas de implementación organizados en capas; implementar elementos de diseño en términos de elementos de implementación tales como ficheros fuente, binarios, ejecutables, etc.; probar los componentes desarrollados como unidades; integrar los resultados producidos por los implementadores individuales (o los equipos) en un sistema ejecutable.

La disciplina de Diseño describe cómo desarrollar un Modelo de Diseño el cual representa la intención de la implementación y es la entrada primaria al Flujo de Trabajo de Implementación.

A continuación en la Figura 1.4 se muestran los Artefactos y los Trabajadores del Flujo de Trabajo de Implementación.

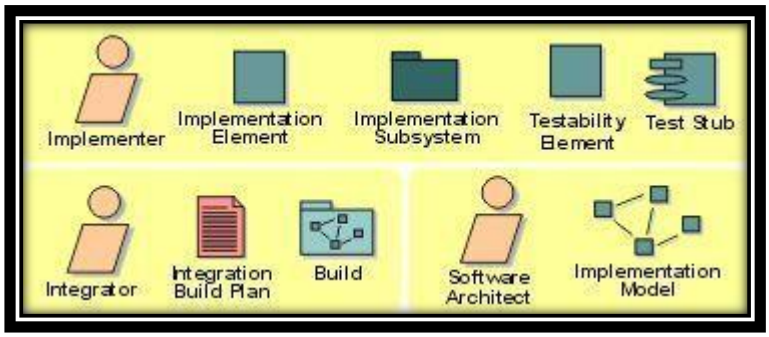


Figura 1.4. Artefactos y Trabajadores del Flujo de Trabajo de implementación

1.9.1. Principales Artefactos

1. Modelo de Implementación

El Modelo de Implementación representa la composición física de la implementación en términos de Subsistemas de Implementación y Elementos de Implementación tales como directorios y ficheros

incluyendo código fuente, datos y ejecutables. El Arquitecto de Software es el responsable de la integridad de este modelo.

2. Subsistema de Implementación

Los Subsistemas de Implementación son una serie de Elementos de Implementación. Estructuran el Modelo de Implementación al dividirlo en pequeñas partes que puedan ser integradas y probadas individualmente. Son análogos a los Paquetes de Diseño y pertenecen a la Vista de Implementación. El Implementador es el responsable del subsistema.

3. Elementos de Implementación

Constituyen la parte física que compone una implementación incluyendo archivos y directorios. Contienen archivos de código (fuente, binarios o ejecutables), archivos de datos y de documentación como por ejemplo, archivos de ayuda en línea. El Implementador es el responsable de los elementos.

4. Build

Un Build es una versión operativa de un sistema o de una parte de un sistema que demuestra un subconjunto de las capacidades que se proveerán en el producto final. Consta de uno o varios Elementos de Implementación (a menudo ejecutables), cada uno construido a partir de otros elementos, por lo general por un proceso de compilación y vinculación de código fuente. El Integrador es el responsable de la producción de los Builds.

5. Plan de Integración del Build

El Plan de Integración del Build provee un plan detallado para la integración dentro de una iteración. Define el orden en el cual los componentes deben ser implementados, cuáles Builds crear cuando se integra el sistema y cómo estos van a ser evaluados. El Integrador es el responsable de este plan y además, debe mantenerlo actualizado.

6. Diagrama de Componentes

Muestran la estructura de los componentes[8], incluyendo clasificadores que los especifican y artefactos que los implementan. También son usados para mostrar la estructura de alto nivel del Modelo de Implementación en términos de Subsistemas de Implementación y las relaciones entre los Elementos de Implementación.

[8] Parte modular de un sistema, desplegable y reemplazable que encapsula implementación y expone un conjunto de interfaces y proporciona la realización de los mismos. Típicamente contiene clases y puede ser implementado por uno o más artefactos.

1.9.2. Trabajadores más importantes

1. Arquitecto de Software

También participa en el Flujo de Trabajo de Análisis y Diseño (ver epígrafe 1.8.2, Arquitecto de Software) Actualiza el Documento de Arquitectura de Software añadiéndole la Vista de Implementación y el Modelo de Implementación también, ya sea porque se crea por primera vez o se añaden nuevos Elementos de Implementación a los ya existentes.

2. Implementador

Es responsable de desarrollar y probar componentes, en acuerdo con los estándares adoptados en el proyecto, para una posterior integración en subsistemas. Se le puede asignar el trabajo de implementar una parte estructural del sistema (puede ser una clase o un subsistema) o una parte funcional, como por ejemplo, una realización de un caso de uso.

3. Integrador

El Integrador es el responsable de planear la integración y llevarla a cabo para los Elementos de Implementación y así, producir diferentes Builds.

1.10. Plataforma de Desarrollo Microsoft.Net

.NET es un proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma y que permita un rápido desarrollo de aplicaciones. Basado en esta plataforma, Microsoft intenta desarrollar una estrategia horizontal que integre todos sus productos, desde el Sistema Operativo hasta las herramientas de mercado. (Cuevas Guerrero)

Ésta plataforma desembocó en el panorama empresarial en el año 2001 y ofrece a los desarrolladores algunas ventajas interesantes, entre las que se pueden mencionar el soporte para múltiples lenguajes, una perfecta integración con el resto de los productos de Microsoft, dispone del Visual Studio .Net como una herramienta muy potente que ofrece un entorno homogéneo de desarrollo, los desarrolladores poco experimentados pueden utilizar lenguajes como Visual Basic .NET que hacen muy sencilla la creación de aplicaciones empresariales. (Ávila, 2007)

Con esta plataforma Microsoft incursiona de lleno en el campo de los Servicios Web y establece el XML como norma en el transporte de información en sus productos y lo promociona como tal en los sistemas desarrollados utilizando sus herramientas. .NET intenta ofrecer una manera rápida y económica pero a la vez segura y robusta de desarrollar aplicaciones o como la misma plataforma las

denomina, “soluciones”, permitiendo a su vez una integración más rápida y ágil entre empresas y un acceso más simple y universal a todo tipo de información desde cualquier tipo de dispositivo.

A largo plazo Microsoft pretende reemplazar la API Win32 o Windows API con la plataforma .NET. Esto es debido a que la API Win32 o Windows API fue desarrollada sobre la marcha, careciendo de documentación detallada, uniformidad y cohesión entre sus distintos componentes, provocando múltiples problemas en el desarrollo de aplicaciones para el sistema operativo Windows. La plataforma .NET pretende solventar la mayoría de estos problemas proveyendo un conjunto único y expandible con facilidad, de bloques interconectados, diseñados de forma uniforme y bien documentados, que permitan a los desarrolladores tener a mano todo lo que necesitan para producir aplicaciones sólidas.

Debido a las ventajas que la disponibilidad de una plataforma de este tipo puede darle a las empresas de tecnología y al público en general, muchas otras empresas e instituciones se han unido a Microsoft en el desarrollo y fortalecimiento de la plataforma .NET, ya sea por medio de la implementación de la plataforma para otros sistemas operativos aparte de Windows (Proyecto Mono de Ximian/Novell para Linux/MacOS X/BSD/Solaris), el desarrollo de lenguajes de programación adicionales para la plataforma (ANSI C de la Universidad de Princeton, NetCOBOL de Fujitsu, Delphi de Borland, entre otros) o la creación de bloques adicionales para la plataforma (como controles, componentes y bibliotecas de clases adicionales); siendo algunas de ellas software libre, distribuibles bajo la licencia GPL. (Cuevas Guerrero)

Entre las principales desventajas de esta plataforma se pueden mencionar que no soporta múltiples sistemas operativos y que por ser exclusiva de Microsoft es ésta sola empresa la única que puede añadir y quitar características según crea necesaria. (Ávila, 2007)

En el caso específico de C#, Microsoft lo diseñó desde su base para aprovechar el nuevo entorno .NET Framework. Como C# forma parte de este nuevo mundo .NET, deberá comprender perfectamente lo que proporciona .NET Framework y de qué manera aumenta su productividad. (Ferguson, y otros, 2003)

La base de la plataforma.NET la constituye el "framework" o marco de trabajo, y este denota la infraestructura sobre la cual se reúnen un conjunto de lenguajes, herramientas y servicios que simplifican el desarrollo de aplicaciones en entorno de ejecución distribuido.

Los principales componentes del marco de trabajo son (Ver Figura 1.5):

- El conjunto de lenguajes de programación.
- El Entorno Común de Ejecución para Lenguajes o CLR por sus siglas en inglés.
- La Biblioteca de Clases Base o BCL.

- El entorno ASP.NET.

(Cuevas Guerrero)

Figura 1.5. Marco de Trabajo de .Net

.NET Framework fue diseñado con tres objetivos en mente. Primero, debía lograr aplicaciones Windows mucho más estables, aunque también debía proporcionar una aplicación con un mayor grado de seguridad. En segundo lugar, debía simplificar el desarrollo de aplicaciones y servicios Web que no sólo funcionen en plataformas tradicionales, sino también en dispositivos móviles. Por último, el entorno fue diseñado para proporcionar un solo grupo de bibliotecas que pudieran trabajar con varios lenguajes.

Los componentes del .NET Framework proveen los "ladrillos" necesarios para construir las aplicaciones Web, los servicios Web y cualquier otra aplicación dentro de Visual Studio .NET. Microsoft ofrece herramientas de desarrollo de extrema productividad que se centran en la fase de construcción de códigos del ciclo de vida de la aplicación.

Con la introducción de Microsoft Visual Studio 97, esta perspectiva del ciclo de vida de la aplicación comenzó a ampliarse, incluyendo compatibilidad para el análisis, el diseño y el desarrollo basado en equipos. En la actualidad, con Visual Studio .NET 2005, Microsoft garantiza nuevas características del ciclo de vida empresarial que ayudan a las organizaciones a programar, analizar, diseñar, generar, probar y coordinar los equipos que producen aplicaciones y servicios Web XML. (Ferguson, y otros, 2003) (Ávila, 2007)

1.10.1. Lenguaje de Programación C-Sharp (C#)

(Charte Ojeda, 2002) (Ferguson, y otros, 2003)

C# (C-Sharp) es un lenguaje de programación Orientado a Objetos desarrollado por Microsoft el cual fue presentado al público por primera vez en la Professional Developer's Conference en Orlando, Florida, en el verano del año 2000. Combina las mejores ideas de diferentes lenguajes como C, C++, Java y Visual Basic con las mejoras de productividad de .NET Framework, aunque también ha sido influenciado por Modula 2 y SmallTalk. Brinda una experiencia de codificación muy productiva tanto para los nuevos programadores como para los más experimentados.

El principal objetivo de Microsoft fue la creación del primer lenguaje Orientado a Componentes, al estilo de Visual Basic, pero flexible y potente como C++ pero sin muchas de sus complejidades. Se diseñó de modo que retuviera casi toda la sintaxis de C y C++. Los programadores que estén familiarizados con estos dos lenguajes pueden escoger el código C# y empezar a programar de forma relativamente rápida. Sin embargo, la gran ventaja de C# consiste en que sus diseñadores decidieron no hacerlo compatible con los anteriores C y C++.

C# elimina las cosas que hacían que fuese difícil trabajar con estos dos lenguajes, por ejemplo, los punteros. Como todo el código C es también código C++, éste tenía que mantener todas las rarezas y deficiencias de C, sin embargo, C# parte de cero y sin ningún requisito de compatibilidad, así que mantiene los puntos fuertes de sus predecesores y descarta las debilidades que complicaban las cosas a los programadores de C y C++.

Cuenta con la mayoría de las características de orientación a objetos de C++. Su entidad de primer nivel es la *clase,* de la cual se crearían objetos y se derivarían nuevas clases. No pueden faltar por supuesto, la encapsulación, herencia y polimorfismo. Las clases pueden agruparse en *espacios de nombres o namespaces.* No existen los conceptos de definición e implementación como elementos separados en una clase, todo se realiza en la misma, simplificando la codificación.

C# asimila, e incluso supera, las tradicionales facilidades que da Visual Basic a la hora de usar y diseñar componentes sin perder un ápice de potencia y flexibilidad. Se puede usar C# para fabricar los componentes que encapsulan la lógica de negocio y acceden a bases de datos, las interfaces de usuario que actúan como clientes de dichos componentes y el código de servidor que actúa detrás de las interfaces Web.

La destrucción de objetos y liberación de la memoria asociada se produce de manera automática gracias a la existencia de un recolector de basura. El compilador no genera código ejecutable en ningún caso, sino que produce código MSIL (Microsoft Intermediate Language) por lo que el

programador no tiene que preocuparse del sistema operativo o procesador en el que va a ejecutarse su aplicación.

1.11. Patrones de Diseño

1.11.1. Abstract Factory (Fábrica Abstracta)

(Dodero, y otros, 2002-2003) (Gamma, y otros, 1998)

Es un patrón que ofrece una interfaz para la creación de familias de productos relacionados o dependientes sin especificar las clases concretas a las que pertenecen. Es conocido también como *Kit*. Se debe usar cuando:

- Un sistema debe ser independiente de cómo se crean, se componen y se representan sus productos.
- Un sistema se debe configurar con una de entre varias familias de productos.
- Una familia de productos relacionados están hechos para utilizarse juntos (hay que hacer que esto se cumpla).
- Se desea ofrecer una biblioteca de productos a partir de su interfaz sin revelar su implementación.

Estructura (Ver Anexo 1)

Las clases participantes en la estructura son las siguientes:

AbstractFactory: Declara una interfaz para la creación de objetos de productos abstractos.

ConcreteFactory: Implementa las operaciones para la creación de objetos de productos concretos.

AbstractProduct: Declara una interfaz para los objetos de un tipo de productos.

ConcreteProduct: Define un objeto de producto que la correspondiente factoría concreta se encargará de crear, a la vez que implementa la interfaz de producto abstracto.

Client: Utiliza solamente las interfaces declaradas en la factoría abstracta y en los productos abstractos.

Las clases AbstractFactory a menudo son implementadas con un método factoría (Factory Method[9]), pero también usando prototipos (Prototype[10]). Una factoría concreta a veces es un Singleton.

[9] Patrón de Creación que define una interfaz para crear un objeto, pero deja a las subclases decidir qué clases van a instanciar.

[10] Patrón de Creación que especifica los tipos de objetos a crear usando una instancia prototípica y crea nuevos objetos mediante la copia de estos prototipos.

1.11.2. Singleton

(Gamma, y otros, 1998) (Dodero, y otros, 2002-2003)
El patrón Singleton asegura que sólo se pueda crear una instancia de una clase, y ofrece un punto de acceso global a ella. Se debe usar cuando:

- Existe la necesidad de que una clase se instancie una sola vez de modo que todos los clientes puedan acceder a esa única instancia desde un punto conocido.
- La instancia única se puede ampliar mediante subclases y los clientes deben ser capaces de utilizar las instancias de las subclases sin tener que modificar su código.

Estructura (Ver Anexo 2)

La clase participante en la estructura es la siguiente:

Singleton: Es el responsable de la creación de su única instancia. Define una operación para crear instancias que ofrece a todos los clientes la misma y única instancia. Éste método debe ser estático.

Diferentes patrones pueden ser implementados usando el Singleton; entre ellos Abstract Factory, Builder[11] y Prototype.

1.11.3. Facade (Fachada)

(Gamma, y otros, 1998)
El patrón Facade tiene como propósito ofrecer un interfaz de acceso único a un conjunto de interfaces en un subsistema. Define una interfaz de alto nivel que hace al subsistema fácil de usar. Se debe usar cuando:

- Se quiera proporcionar una interfaz sencilla para un subsistema complejo.
- Se quiera desacoplar un subsistema de sus clientes y de otros subsistemas, haciéndolo mas independiente y portable.
- Se quiera dividir los sistemas en niveles: las fachadas serían el punto de entrada a cada nivel.

Estructura (Ver Anexo 3)

Las clases participantes en la estructura son las siguientes:

Fachada: Conoce las clases del subsistema responsables de un pedido y delega las peticiones de los clientes en los objetos del subsistema.

[11]Patrón de Creación que separa la construcción de un objeto complejo de su representación para que éste mismo proceso de construcción pueda crear diferentes representaciones.

Clases del subsistema: Implementan la funcionalidad del subsistema y llevan a cabo las peticiones que les envía la Fachada, aunque no la conocen.

Abstract Factory puede ser usado junto con Facade proporcionando una interfaz para crear objetos subsistemas. Por lo general, sólo un objeto Facade es requerido. De este modo, los objetos Facade son a menudo Singleton.

1.12. Herramienta de Modelado Enterprise Architect

(Sparx Systems, 2007)
Enterprise Architect es una herramienta comprensible de diseño y análisis UML, que cubre el desarrollo de software desde el paso de los Requerimientos a través de las etapas del Análisis, Diseño, Pruebas y Mantenimiento. EA es una herramienta multiusuario, basada en Windows, diseñada para ayudar a construir software robusto y fácil de mantener. Ofrece una salida de documentación flexible y de alta calidad.

El Lenguaje Unificado de Modelado provee beneficios significativos para ayudar a construir modelos de sistemas de software rigurosos y donde es posible mantener la trazabilidad de manera consistente. Enterprise Architect soporta este proceso en un ambiente fácil de usar, rápido y flexible.

Sus bases están construidas sobre la especificación de UML 2.0. Tiene soportes para los diferentes diagramas de éste (Diagrama de Clases, de Objetos, Secuencia, Componentes, Casos de Uso etc.)

Provee una trazabilidad completa desde el análisis de Requerimientos hasta los artefactos de Análisis y Diseño, a través de la Implementación y el Despliegue. Combinados con la ubicación de recursos y tareas incorporados, los equipos de Administradores de Proyectos y Calidad están equipados con la información que ellos necesitan para entregar proyectos en tiempo.

Enterprise Architect suministra una generación poderosa de documentos y herramientas de reporte con un editor de plantilla completo WYSIWYG. Ayuda a administrar la complejidad con herramientas para rastrear las dependencias, tiene soporte para modelos muy grandes y control de versiones con proveedores CVS o SCC.

Soporta generación e ingeniería inversa de código fuente para muchos lenguajes populares, incluyendo C++, C#, Visual Basic.Net, Java, Delphi, Visual Basic y PHP. También hay disponibles Add-ins gratis para CORBA y Python. Cuenta con un editor de código fuente con "resaltador de sintaxis" incorporado.

EA soporta transformaciones de Arquitectura avanzada dirigida por Modelos (MDA) usando plantillas de transformaciones de desarrollo y fáciles de usar. Con transformaciones incorporadas para DDL, C#,

Java, EJB y XSD, se pueden desarrollar soluciones complejas desde los simples "Modelos Independientes de Plataforma" (MIP) que son el objetivo en "Modelos Específicos de Plataforma" (MEP). Un Modelo Independiente de Plataforma se puede usar para generar y sincronizar múltiples modelos, proveyendo un aumento de productividad significativo.

1.13. Sistema Gestor de Bases de Datos Oracle 10G Enterprise Edition

(Oracle, 2006)

Oracle Database 10G Enterprise Edition es ideal para empresas que necesitan soportar altos volúmenes de procesamiento de transacciones on-line y consultas constantes a aplicaciones de almacenamiento de datos. Éste provee probada escalabilidad en todas las configuraciones del hardware, y puede ser usado para administrar grandes cantidades de información, con altos niveles de garantía de seguridad en la industria.

Proporciona beneficios de disponibilidad únicos que protegen los datos de costosos errores humanos, reduce el tiempo de inactividad asociado con el mantenimiento de rutina, e incluye capacidades de auto-gestión para ayudar a disminuir los costos operacionales. Está disponible en todos los sistemas operativos que soportan Oracle, incluyendo Windows, Linux and Unix, y es soportado en todas las configuraciones de hardware.

Oracle Database 10G Enterprise Edition soporta todos los tipos de datos relacionales estándar, así como el almacenamiento nativo de XML, texto, documentos, imágenes, audio, video y localización de los datos. El acceso a los datos almacenados es a través de interfaces tales como SQL, JDBC, SQLJ, ODBC, OLE DB y ODP.NET, SQL/XML, XQuery, and WebDAV. La lógica de negocio desplegada en la base de datos puede ser escrita en Java y PL/SQL.

Sustenta consultas y transacciones distribuidas entre dos o más bases de datos, e incluye el soporte para conectarse por ODBC a bases de datos de terceras partes comunes. Además provee un framework para la captura, establecimiento y procesamiento de eventos en una base de datos, tales como los que son causados por cambios en los datos o creados por aplicaciones de negocio.

Puede también ser usado como el centro de almacenamiento de datos coordinado replicado en el ambiente de oficina, en conjunto con una Standard Edition local o una base de datos Standard Edition One. La replicación entre varios servidores Peer-to-Peer es también soportada entre dos o más Enterprise Editions de una base de datos Oracle.

Para la demanda de ambientes de procesamiento de transacciones online, Oracle Database 10G Enterprise Edition sustenta el despliegue de grandes números de usuarios por la utilización de filas de

bloqueo únicas por niveles y versiones múltiples de consistencia de lectura, permitiendo que una aplicación rápida aumente fácilmente de decenas a decenas de miles de usuarios en línea.

Lidera la industria por las características de seguridad que contempla, tales como columna de seguridad, columna de filtrado, encriptación de datos, autenticación por proxy, contexto de aplicación y roles de aplicación seguro que se adicionan a los cuatro rasgos de seguridad que se adicionan a los comúnmente conocidos, tales como auditoría, chequeo de la complejidad de contraseñas, soporte robusto para los roles en la base de datos, procedimientos almacenados y funciones.

Oracle Advanced Security protege la privacidad y confidencialidad de los datos sobre la red direccionando data sniffing, data loss, replay and person-in-the-middle attacks. Toda la comunicación con una base de datos Oracle puede ser encriptado con Oracle Advanced Security. Oracle Advanced Security también provee poderosas soluciones de autenticación para Oracle Database 10G. Oracle Label Security provee una solución ideal para los clientes que necesitan proteger información sensible o privada. Basado en niveles múltiples de seguridad de la tecnología, Oracle Label Security restringe el acceso a los datos usando etiquetas sensibles y acreditaciones.

Para la amplia administración de la empresa, las cuentas de usuarios y las autorizaciones pueden ser manejadas centralmente con Oracle Database 10G enterprise user security y Oracle Identify Management, eliminando la necesidad de esquemas individuales de usuarios de la base de datos y haciendo fácil la administración de las autorizaciones por la organización.

1.14. Framework de Persistencia NHibernate

NHibernate es un framework de ORM (Object Relational Mapping), basado en el popular Hibernate surgido en la comunidad Java en el año 2002, que tiene como función principal mapear los objetos desde una aplicación .Net a una Base de Datos Relacional (DB2, Oracle, SQL Server, PostgreSQL, MySQL).

NHibernate para poder conocer la correspondencia que existe entre los objetos y las tablas, lo hace por medio de la configuración de mapeo. Esta configuración se puede hacer de forma programática o mediante archivos de mapeo XML (mapping files). Estos archivos poseen la información necesaria para poder saber en qué tabla se tiene que guardar cada objeto o en qué tabla se tiene que buscar para obtener un determinado objeto. La extensión de estos archivos XML es .hbm.xml. Por ejemplo Cuenta.hbm.xml, Cliente.hbm.xml, etc. (Quintana, 2007)

Utilizar NHibernate ofrece las siguientes ventajas (2007):

- Persistencia Transparente: Los objetos del dominio no saben nada acerca de la base de datos donde son persistidos, el framework lo resuelve en forma automática utilizando archivos de mapeo expresados en XML.
- Soporte de Polimorfismo: Se pueden cargar jerarquías de objetos en forma polimórfica.
- Soporte de los tres niveles de mapeo de Herencia: Se puede mapear toda una jerarquía de clases a una sola tabla, crear una tabla por cada clase concreta o crear una tabla por cada escalón de la jerarquía.
- Soporte completo de asociaciones: Los frameworks de ORM (Object Relational Mapping) soportan el mapeo de todos los tipos de relaciones que pueden existir en un modelo de objetos del dominio (asociaciones 1...1; 1...N; N...M, unidireccionales y bidireccionales).
- Soporte de carga de objetos Proxy: Se pueden cargar objetos que sólo contengan la clave del objeto completo.
- Soporte de múltiples dialectos SQL: Las aplicaciones pueden persistir sus datos en SQL Server, en Oracle, en MySQL, etc. simplemente cambiando la configuración correspondiente.

La arquitectura propuesta está enmarcada en la solución general para la Automatización de los Registros y Notarias de la República Bolivariana de Venezuela y de ella hereda una serie de mecanismos y características tales como: el Lenguaje de Programación (C#), el IDE[12] de desarrollo (Visual Studio .Net), las Políticas de Seguridad, el Gestor de Base de Datos, la Topología de Red. No obstante, se incorporan y modifican algunos elementos relacionados fundamentalmente con la lógica de persistencia y la estructura de capas del modelo propuesto inicialmente.

Conclusiones

Como resultado de la investigación y el análisis bibliográfico realizado, a lo largo de este capítulo, han sido expuestos los principales puntos de interés relacionados con el Proceso Presupuestario, los objetivos, las etapas, los aspectos generales de éste en Venezuela, etc. Se plantea también la necesidad de contar con un Sistema Integrado de Administración Financiera y objetivos de éste. No se deja de mencionar la solución informática llamada SIGECOF (Sistema Integrado de Gestión y Control de las Finanzas Públicas), pero que no cumple con los parámetros necesarios en su funcionamiento. La metodología, las herramientas a utilizar y los patrones también tienen su pequeño espacio en el capítulo donde se exponen sus principales características y funcionamiento, así como los artefactos resultantes y trabajadores participantes en las disciplinas de Diseño e Implementación.

[12] Integrated Development Environment (Entorno de Desarrollo Integrado).

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

Introducción

En el siguiente capítulo se describe la arquitectura que utiliza el sistema y se abordan las capas por las que está compuesta brindándose un breve análisis de éstas. Se exponen las diferentes clases del diseño junto con la explicación de cada una de ellas y se argumentan los Patrones de Diseño. Se hace referencia a los principales Casos de Uso del sistema junto con sus Diagramas de Clases y de Secuencia. También se analizan un grupo de componentes que han sido desarrollados para el sistema y las ventajas que éstos poseen. Se plantea un Estándar de Codificación que contiene un conjunto de reglas a seguir en la implementación del software. Además, se tienen el Modelo de Datos, Diagrama de Componentes y de Despliegue del sistema.

2.1. Breve descripción de la Arquitectura

Cuando una persona nombra la palabra “arquitectura”, la primera definición que viene a la mente es que “*la Arquitectura es el arte de proyectar y construir edificaciones de diferentes tipos*”. Se piensa en la estructura física de la construcción, pero la arquitectura es mucho más que eso, es la forma en la que los diferentes componentes del edificio se integran para formar un todo unido de manera tal que se cumplan los objetivos que se trazaron para realizar la obra.

En Informática, la Arquitectura de Software según (Pressman, 2002 pág. 238) es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos.

La arquitectura constituye un modelo relativamente pequeño e intelectualmente comprensible de cómo está estructurado el sistema y de cómo trabajan juntos sus componentes facilitando así la comunicación entre todas las partes implicadas en el desarrollo del software

La arquitectura del Sistema de Administración Financiera del Proyecto Registros y Notarías de la República Bolivariana de Venezuela se encuentra organizada desde dos enfoques, Vertical y Horizontal, explicados a continuación.

2.1.1. Enfoque Horizontal

El sistema está dividido en once módulos principales donde cada uno responde a un conjunto de funcionalidades y casos de uso específicos del cliente.

Estos interactúan y comparten datos de interés en dependencia de la funcionalidad de cada uno y siguiendo un estricto régimen de seguridad de la información. Los módulos son:

- Administración
- Presupuesto
- Contabilidad
- Recaudación
- Requisiciones
- Compras y Servicios
- Retenciones
- Tesorería
- Fondo en Anticipo
- Fondo de Caja Chica
- Ejecución y Cierre de Presupuesto

2.1.2. Enfoque Vertical

El desarrollo de cada uno de los módulos responde a un modelo multicapas. Este tipo de sistema puede proporcionar varias configuraciones diferentes. En una arquitectura de n niveles como esta, las acciones de la aplicación están lógicamente divididas por funciones.

En este caso se decidió concebir 5 capas, las cuales se muestran a continuación:

- Capa de Presentación
- Capa Lógica del Negocio
- Capa de Fachada
- Capa Acceso a Datos
- Capa de Datos

En la Figura 2.1 se muestra la representación del enfoque vertical de la arquitectura del sistema

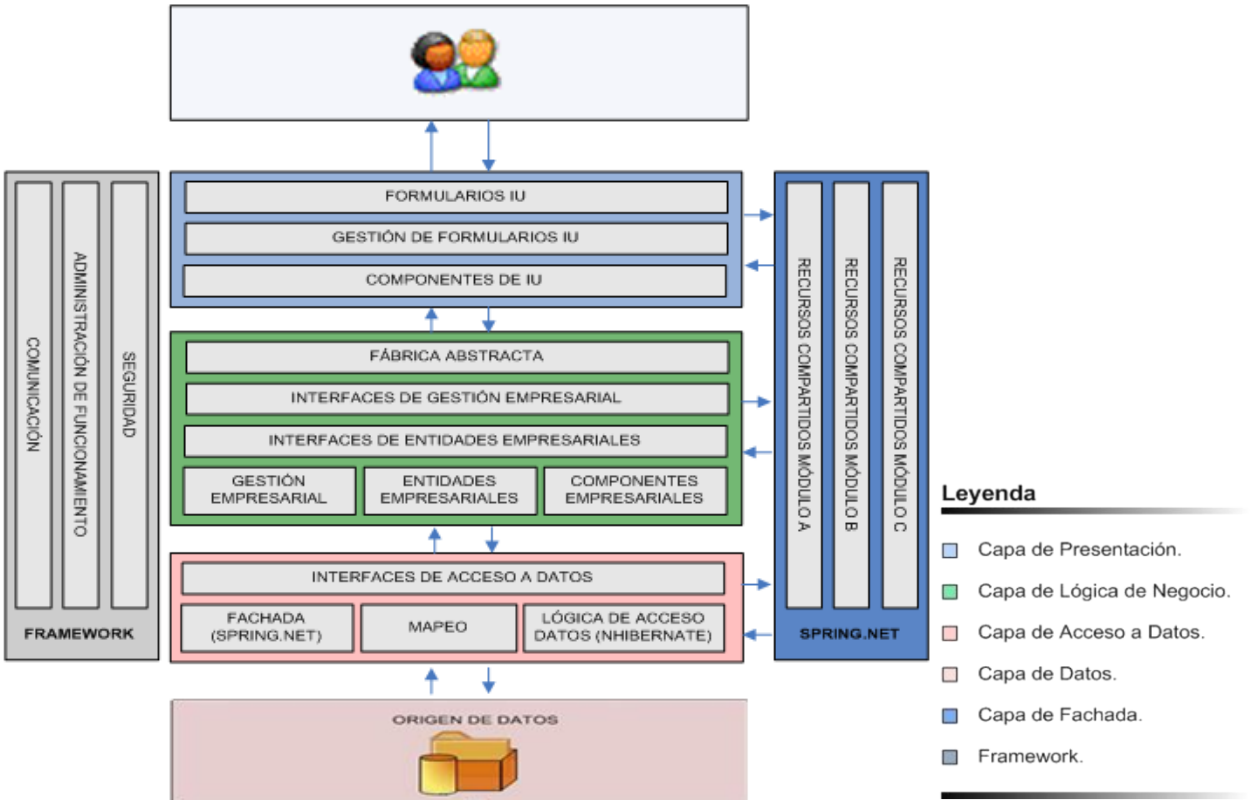


Figura 2.1. Representación del enfoque vertical de la arquitectura

2.2. Modelo basado en capas

Se define al Estilo en Capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas. El uso de arquitecturas en capas, explícitas o implícitas, es frecuentísimo hoy en día.

El módulo Presupuesto utiliza el Estilo Arquitectónico basado en Capas, el cual permite distribuir el trabajo en varios niveles y si es necesario hacer algún cambio sólo se hace en el nivel requerido sin tener que afectar a otro.

2.2.1. Capa de Presentación

Aquí se encuentran los Formularios de Interfaz de Usuario, los cuales responden a un comportamiento y diseño estándar. Su uso se basa en la explotación de estos formularios que realmente provee un framework facilitando el desarrollo del sistema. Este framework se basa en “acciones” contempladas en la Gestión de Formularios de Interfaz de Usuario y cada acción se corresponde con funcionalidades de captura o visualización de los datos que tiene asociado un formulario cuya forma depende de la funcionalidad específica para la cual fue concebida dicha acción.

Una acción es una clase que representa la ejecución de alguna tarea concreta. Esta clase debe heredar de la clase Acción o AcciónSegura (que proporciona el framework), puede tener o no una forma visual asociada y en el caso que la tenga, se deben programar dentro de la misma todos los eventos que puedan ocurrir y que se deseen usar a partir de la interacción con la forma visual.

Los Componentes de Interfaz de Usuario son recursos utilizados para encapsular una función determinada y que serán utilizados por más de un proceso en el módulo, el hecho de que estén en la Capa de Presentación se debe a que trabajan con los formularios directamente, ya sea de forma visual o con los datos que se encuentran relacionados en estos a través de otros componentes o controles. Aquí pueden encontrarse clases de negocio e inclusive elementos de acceso a datos, distribuidos según la estructura que tiene la arquitectura general.

2.2.2. Capa Lógica del Negocio

El diseño de esta capa depende directamente del negocio específico del módulo. El Negocio recibe datos o información capturada en las interfaces de usuario, gestiona o procesa la misma, de ser necesario solicitándola a la Capa de Acceso a Datos y finalmente se la envía a la de Presentación nuevamente para que ésta la presente al usuario en el punto donde se inició la petición.

Las “entidades” del negocio son clases objeto-valor que representan los datos con los que se van a trabajar en cada uno de los procesos que se están automatizando. Cada entidad se encarga de procesar sus propios datos o valores sin interactuar con los demás elementos del negocio, con esto se garantiza independencia y encapsulamiento de la información.

Los “gestores” tienen como objetivo agrupar una serie de entidades en un fin común, y así, obtener funcionalidades donde intervienen una serie de datos que se encuentran en dichas clases objeto-valor. Los Componentes de Negocio son recursos utilizados para encapsular una función determinada y que serán utilizados por más de un proceso de negocio en el módulo, estas funcionalidades son netamente de este nivel, no tienen Presentación pero sí pueden utilizar recursos del Acceso a Datos.

Algo fundamental es la utilización de las “interfaces” como recurso que se usa para brindar funcionalidades que serán implementadas. La mayoría de las interfaces van a estar en el Negocio ya que aquí se encuentran, en su mayoría, la implementación de estas funcionalidades, ya sea en las entidades, como en los gestores.

2.2.3. Capa de Fachada

Es una representación del patrón Fachada a gran escala, este nivel es un recurso utilizado para mantener una representación de los demás módulos en el que se está implementando, siempre y

cuando lo necesite. Es decir, cada módulo que se vaya a comunicar con el otro debe dar las funcionalidades que necesita, las cuales se van a agrupar aquí. Al emplear esta forma de comunicación entre los módulos garantizamos abstracción y enajenación apoyándonos en Spring.net.

2.2.4. Capa Acceso a Datos

Es la más crítica y sensible pues controla todo lo relacionado a la información que se encuentra en la fuente de almacenamiento (Capa de Datos). Al ser la capa inferior, no conoce los niveles superiores, sólo se limita al manejo de la información, ya sea para persistirla o proporcionarla para su procesamiento y propagación por la aplicación. Aquí se propone el uso de NHibernate 1.1.

La Fachada es un recurso usado para buscar la abstracción, es decir, brindar una especie de interfaz con la cual se va a interactuar cada vez que se necesite comunicarse, en este caso, con el Acceso a Datos logrando una completa enajenación ante cualquier modificación que pueda ocurrir.

Esta estructura responde a la implementación del patrón Fachada y todas las clases que conformen este ensamblado serán estáticas debido a que no es necesario instanciarlas, solamente acceder a sus funcionalidades que realmente son un puente de acceso al resto.

El mapeo es un elemento de uso exclusivo de NHibernate que está aislado en un ensamblado porque básicamente son ficheros XML de configuración que son independientes de cualquier implementación, por tanto resulta muy útil tenerlos separados de todo código y así se evita recompilar toda una capa ante cualquier modificación en una configuración de estos XML.

En la Lógica de Acceso a Datos recaen todas las funcionalidades del Acceso a Datos ya que éste ensamblado implementa la totalidad de las operaciones de persistencia y obtención de datos explotando los recursos que brinda NHibernate, como el trabajo con procedimientos almacenados y métodos de persistencia o consultas.

Las Interfaces de Acceso a Datos representan las funcionalidades que brinda este nivel, su uso e importancia es la misma que las de la capa Lógica del Negocio.

2.2.5. Capa de Datos.

Es aquella que corresponde a los almacenes de datos, a ella pertenecen las bases de datos disponibles en los servidores de bases de datos.

2.3. Diseño del sistema

El Diseño es el centro de atención al final de la Fase de Elaboración y el comienzo de las iteraciones de Construcción. El Diseño se encuentra en el núcleo técnico de la Ingeniería del Software. Una vez

que se analizan y especifican los requisitos del sistema, el Diseño del software es la primera de las tres actividades técnicas (diseño, generación de código y pruebas) que se requieren para construir y verificar el software. Cada actividad transforma la información de manera que de lugar por último a un software de computadora validado.

El Diseño de un software es tanto un Proceso como un Modelo. El "Proceso" de Diseño es una secuencia de pasos que hacen posible que el diseñador describa todos los aspectos del software que se va a realizar mediante el cual los requisitos se traducen en un "plano" para construir el software. Un conocimiento creativo, gran experiencia en el tema, un sentido de lo que hace que un software sea bueno, y un compromiso general con la calidad son factores críticos de éxito para un diseño competente.

El "Modelo" es el equivalente a los planes de un arquitecto para una casa. Comienza representando la totalidad de lo que se va a construir y refina lentamente lo que va a proporcionar la guía para construir cada detalle. De manera similar, el Modelo de Diseño que se crea para el software proporciona diversos enfoques diferentes del sistema (Pressman, 2002).El punto de partida para la elaboración de este diseño fueron las especificaciones de los casos de uso del sistema, los requisitos asociados a éstos y las reglas generales del negocio.

2.3.1. Clases del Diseño

El diseño del sistema se ajusta a la arquitectura definida por el proyecto desde su etapa inicial. De esta forma, son identificadas siete tipos de clases: Formularios, Acciones, Gestores, Interfaces, Entidades, Fachadas y de Acceso a Datos.

2.3.1.1. Formularios

Estas son clases que contienen los controles y componentes visuales necesarios para construir las interfaces que son utilizadas para la entrada de datos o para mostrar registros. Sus nombres comienzan con las letras "Frm" por lo que se pueden distinguir fácilmente de otras. Todos los atributos de estas clases son públicos, de manera tal que las acciones que utilizan estas vistas puedan acceder a ellos en un momento determinado. Carecen de cualquier tipo de lógica de negocio o dependencia con éste por lo que pueden ser utilizadas por varias acciones diferentes, que representan procesos de un entorno de negocio específicos.

A continuación, se muestra en la Figura 2.2 un ejemplo de clase Formulario

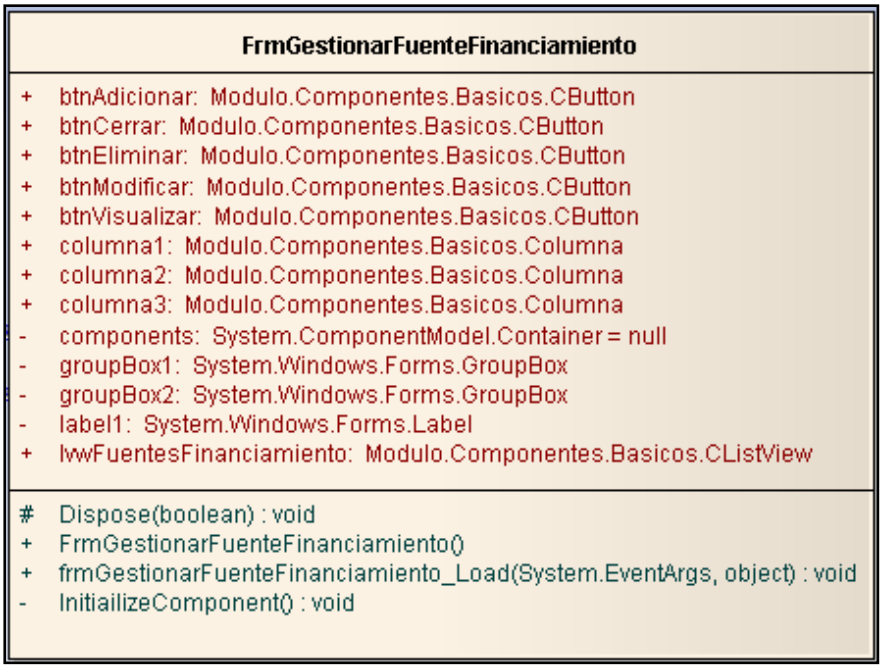


Figura 2.2. Clase del Formulario FrmGestionarFuenteFinanciamiento

De forma general estas vistas pueden heredar de tres tipos básicos de formularios que se incluyen dentro del framework Sistema, éstos son: *Sistema.Interfaz.frmAreaTrabajo, Sistema.Interfaz.frmConfirmar y Sistema.Interfaz.frmMensaje.*

Sistema.Interfaz.frmAreaTrabajo

Estas interfaces se utilizan para mostrar diferentes tipos de información y constituyen el área de trabajo fundamental. Se muestran empotradas en la forma principal por cuestiones prácticas y de diseño gráfico, tienen una dimensión aproximada de (845, 713) píxeles.

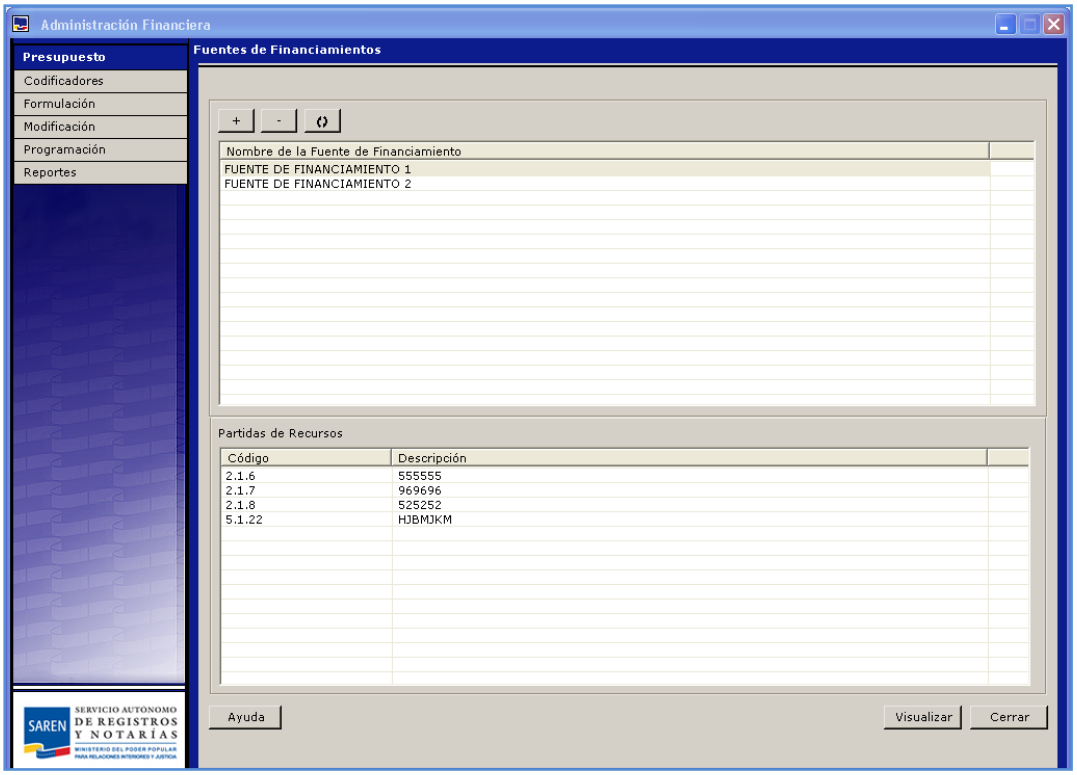


Figura 2.3. Interfaz Gestionar Fuentes de Financiamiento

Sistema.Interfaz.frmConfirmar

La función principal de este tipo de formularios es proporcionar el mecanismo para realizar preguntas y solicitar confirmaciones en determinadas situaciones de los procesos del negocio. Pueden ser configurados y adaptados para una situación específica según sea el caso. Sus dimensiones pueden ser variables.

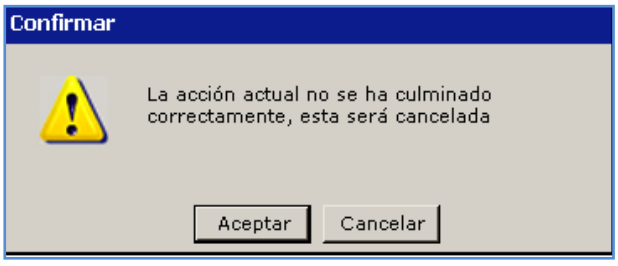


Figura 2.4. Interfaz Confirmar

Sistema.Interfaz.frmMensaje

Son utilizados con frecuencia para crear pequeños formularios que se superponen al área de trabajo que está activa en ese momento y se subordinan mayormente a éstas. Su complejidad y dimensiones pueden variar dependiendo del uso específico que se le de.

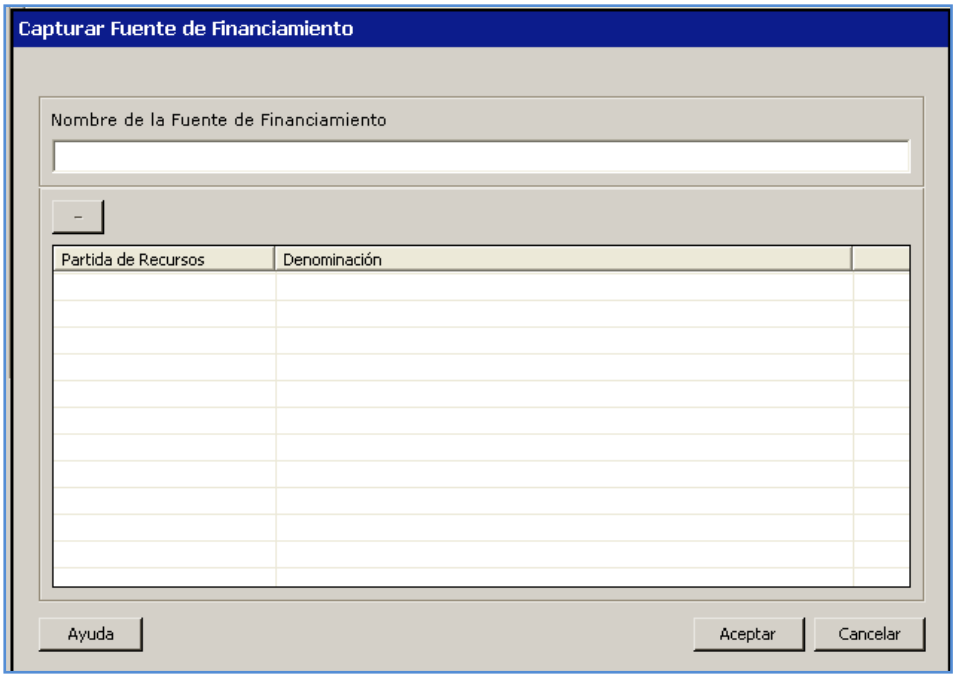


Figura 2.5. Formulario Capturar Fuente de Financiamiento

2.3.1.2. Acciones

Las acciones constituyen procesos específicos o pasos dentro de éstos, generalmente tienen asociada un área de trabajo particular y pueden usar además otros formularios auxiliares. Están nombradas de forma tal que se puedan identificar fácilmente, pues a su nombre se le anteponen las letras “Acc”.

Estas clases representan la base del funcionamiento del sistema porque la gestión del acceso y control de éste se realiza a través de roles bien definidos que están asociados a estas acciones (en la Figura 2.6 se muestra un ejemplo de una clase de este tipo). Pueden heredar de dos tipos de acciones genéricas: *AccionSegura y Accion.*

Sistema.Interfaz.Acciones.Accion

El acceso a estas acciones no se controla, simplemente representan procesos simples del negocio que no requieren restricción alguna.

Sistema.Interfaz.Acciones.AccionSegura

Se realiza un control estricto con el usuario que está accediendo a ellas y las acciones que realiza sobre la misma. Deben ser asociadas a los roles pertinentes a través de configuraciones de la aplicación. En conclusión, las acciones seguras sólo son accesibles y visibles para los usuarios que tienen el privilegio de acceso a las mismas.

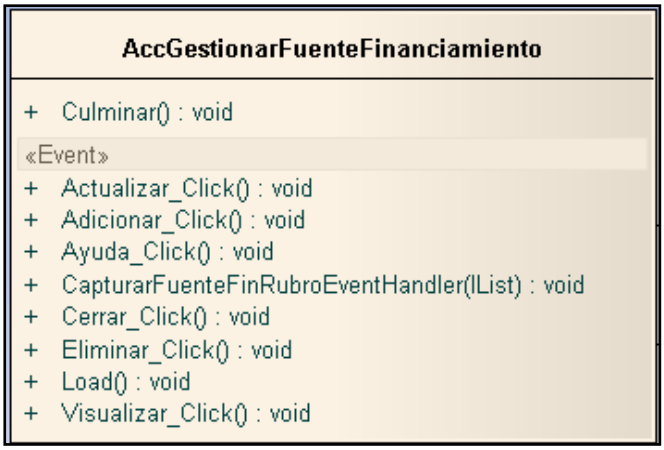


Figura 2.6. Clase acción Gestionar Fuentes de Financiamiento

2.3.1.3. Gestores

Son las clases que contienen la lógica del negocio y manipulan las operaciones internas del sistema y los datos. Sus nombres comienzan con las letras “Gtr” para identificarlas fácilmente. Se construyeron gestores por caso de uso, y sólo en los casos de varias entidades involucradas en un mismo proceso se crearon gestores para los mismos. En la Figura 2.7 se muestra un ejemplo de una clase gestora.

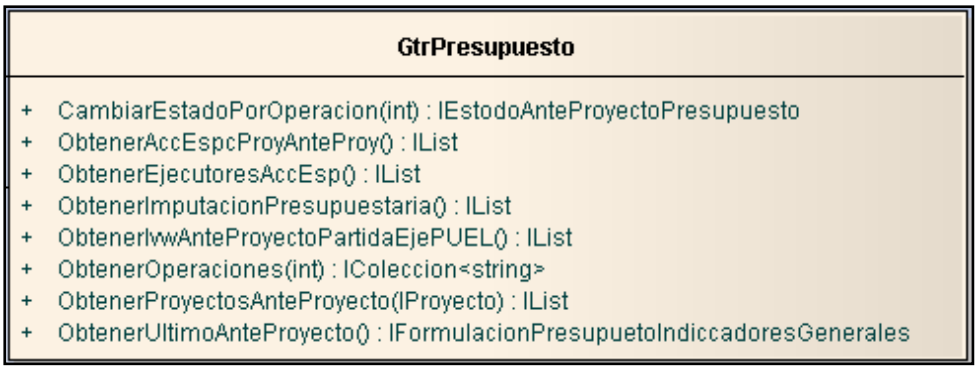


Figura 2.7. Clase gestora GtrPresupuesto

2.3.1.4. Interfaces

Las interfaces son un recurso de la arquitectura para alcanzar desacoplamiento y abstracción entre las funcionalidades que representan y su implementación. La implementación la pueden realizar las entidades, fundamentalmente están constituidas por propiedades, y determinados métodos. Se desarrolla una interfaz por cada entidad. Comienzan con la letra “I” para poder identificarse mejor.

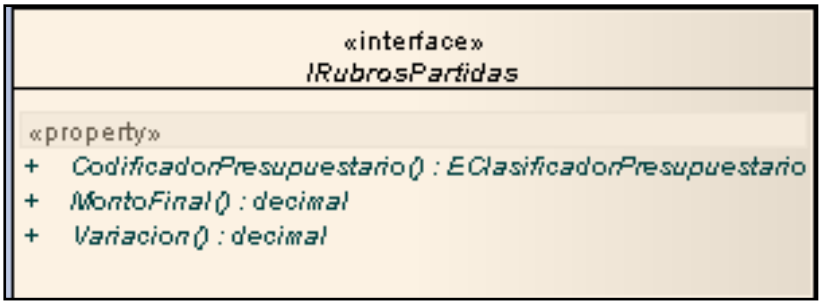


Figura 2.8. Ejemplo de una clase interfaz (en este caso IRubrosPartidas)

2.3.1.5. Entidades

Las entidades son las clases que representan objetos o conceptos del negocio y mayormente se identifican por su nombre (en la Figura 2.9 se muestra un ejemplo de una clase entidad). Se identifican comenzando con la letra “E”.

Además de contener toda la información del sistema, las entidades realizan algunos cálculos y operaciones sobre los datos que poseen, siempre evitando que sean muy complejos, pues en ese caso sería tarea de una clase gestora desempeñar dicha función.

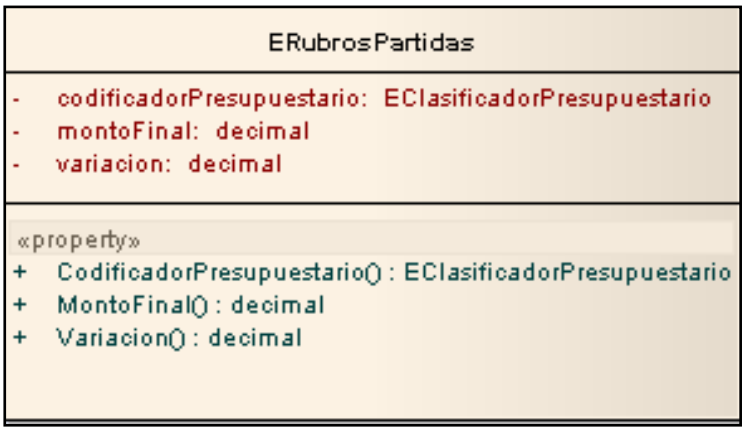


Figura 2.9. Clase entidad ERubrosPartidas

2.3.1.6. Fachadas

Las clases que sirven de fachada son un punto intermedio entre los objetos del negocio y los objetos persistentes que existen en la Capa de Acceso a Datos. Por convenio se agrega al principio del nombre de estas clases la palabra “Fachada”.

Se construyó en la mayoría de los casos una clase fachada para cada grupo de entidades que forman un proceso del negocio aunque existen excepciones por la complejidad de algunos objetos. A continuación se muestra en la Figura 2.10 un ejemplo de una clase de este tipo.

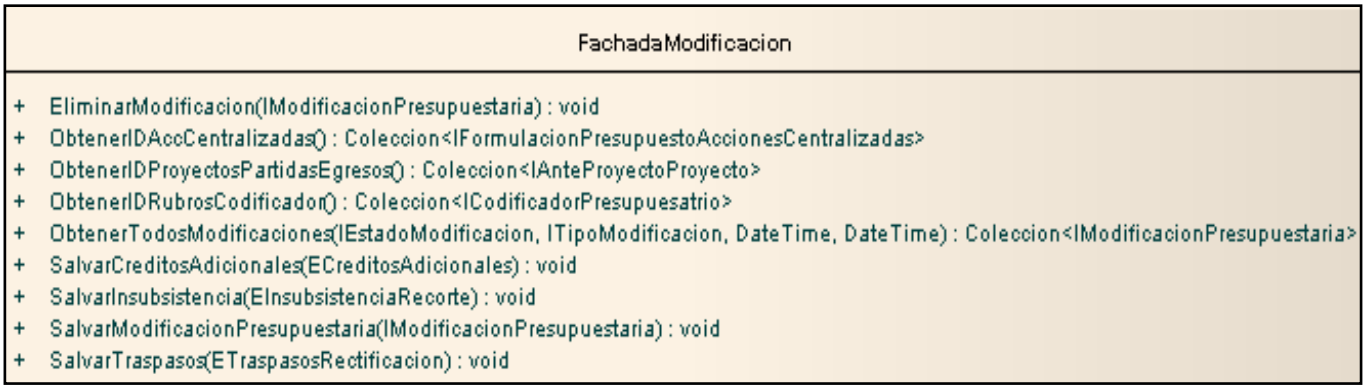


Figura 2.10. Clase FachadaModificación

2.3.1.7. DAO's (Objetos de Acceso a Datos)

Los Objetos de Acceso a Datos son instancias que encapsulan todas las funcionalidades destinadas a realizar todas las operaciones de Base de Datos que requiere la aplicación, dígase inserción, actualización, eliminación de elementos, así como llamadas a consultas y procedimientos almacenados. Sus nombres comienzan con las letras "Dao". En la Figura 2.11 se muestra un ejemplo de una clase de este tipo.

Estos objetos se encuentran preferiblemente en la Capa de Acceso a Datos, donde abstraen todo el negocio de la lógica de persistencia del resto de las capas proporcionándoles sólo un servicio, sin importar como éstos están implementados.

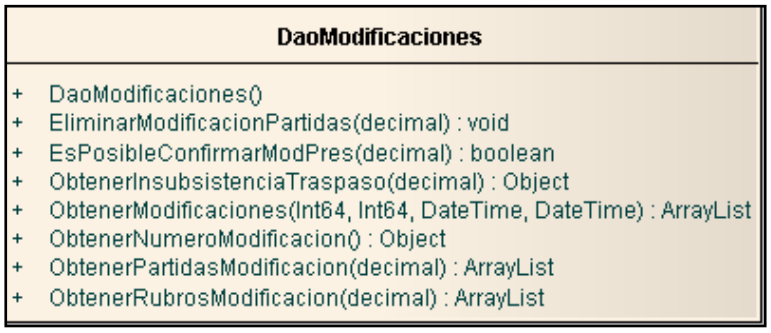


Figura 2.11. Clase DaoModificaciones

2.3.2. Patrones de Diseño

La utilización de patrones en una arquitectura resulta una decisión muy acertada y eficiente puesto que constituyen buenas prácticas y soluciones a problemas muy comunes en el desarrollo de cualquier aplicación.

En el Módulo Presupuesto, se utiliza el framework Spring.Net el cual ayuda a construir aplicaciones empresariales sobre la plataforma .NET, basado en el framework Spring de Java del cual hereda los

principales conceptos. Precisamente al utilizar Spring.Net se aseguran una serie de puntos importantes en toda aplicación, puntos que comprende el framework y no requieren casi ningún código por parte de los desarrolladores. Uno de ellos es que implementa una gran variedad de Patrones de Diseño de una forma bastante sencilla y práctica, entre ellos: Fábrica (Factory), Fábrica Abstracta (Abstract Factory), Constructor (Builder), Decorador (Decorator), Service Locator, Singleton (Solitario), etc. Un punto importante en la solución es la utilización del patrón Facade (Fachada).

El **Solitario** es un Patrón de Creación cuyo propósito es garantizar que una clase sólo tenga una única instancia, proporcionando un punto de acceso global a la misma y que el acceso a ésta esté más controlado. Se usa generalmente en las clases gestoras de negocio, ya que es necesario tenerlas una sola vez instanciadas, y no pueden ser estáticas porque ellas cumplen otras necesidades o responsabilidades con otros objetos.

Permite que se reduzca el espacio de nombres (namespace) frente al uso de variables globales y refinamientos en las operaciones y en la representación mediante la especialización por herencia. Es fácilmente modificable para permitir más de una instancia y, en general, para controlar el número de las mismas (incluso si es variable).

Fachada es un Patrón de Estructura con el objetivo de proporcionar una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. La “Fachada” satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas.

Separa al cliente de los componentes del subsistema reduciendo el número de objetos con los que el cliente trata y facilitando entonces el uso del subsistema. También se promueve un acoplamiento débil entre éste y sus clientes, eliminándose o reduciéndose las dependencias. No existen obstáculos para que las aplicaciones usen las clases del subsistema que necesiten, de esta forma se puede elegir entre facilidad de uso y generalidad.

Fábrica Abstracta se usa generalmente para la creación de los objetos, ya que en la aplicación se trabaja con interfaces y no con objetos. Dentro de ella también se utiliza el patrón Solitario.

Método Plantilla se emplea generalmente en las acciones, o en la Capa de Presentación, presentando los algoritmos a nivel general y que el desarrollador en su propia implementación sea capaz de generar su propio algoritmo de trabajo.

2.3.3. Principales Casos de Uso del Sistema

Debido a la cantidad de casos de usos definidos en su totalidad para el sistema, solamente se muestran aquí los arquitectónicamente significativos (aquellos que presentan un alto grado de prioridad

para el cliente y los más complejos) identificados por el Arquitecto de Software junto con sus Diagramas de Clases y de Secuencia correspondientes (por la complejidad de algunos de ellos y la gran extensión que poseen, es necesario que aparezcan como anexos en el documento o ficheros de una carpeta adjunta a la tesis). Se agrupan en cuatro unidades diferentes: Formular Presupuesto en la Unidad Administradora Central, Formular Presupuesto en la Unidad Ejecutora Local/Unidad Administradora Desconcentrada, Gestionar Modificaciones Presupuestarias y Programación Financiera. También se exponen los Actores del Sistema[13] y una breve descripción de ellos en la siguiente tabla:

Actor del Sistema	Descripción
Responsable de la Unidad de Planificación y Presupuesto.	Es la persona encargada de la Unidad de Planificación y Presupuesto en la UAC[14]. Es el responsable de llevar a cabo todo el trabajo del Presupuesto a nivel Central.
Funcionario de Planificación y Presupuesto.	Es la persona encargada de dirigir en la UEL[15] y en la UAD[16] todos los procesos contables de las mismas.
Funcionario Administrativo en la UEL.	Es el encargado de realizar todas las operaciones administrativas en la UEL.

Tabla 2.1. Actores del Sistema y su descripción

2.3.3.1. Formular Presupuesto en la Unidad Administradora Central

Es iniciado por el Responsable de la Unidad de Planificación y Presupuesto y consiste en formular el Anteproyecto de Presupuesto a nivel central en la Unidad Administradora Central. Contiene cinco casos de uso significativos para la arquitectura, los que se muestran a continuación:

***Gestionar Fuentes de Financiamientos*:** Este caso de uso consiste en asociar las fuentes de financiamiento para crear el ante-proyecto de presupuesto en la Unidad Administradora Central, además de seleccionar por cada fuente de financiamiento el rubro de cada una de las mismas.

***Formular Presupuesto por Proyecto*:** Este caso de uso consiste en formular el presupuesto de un proyecto determinado en la Unidad Administradora Central.

[13] Constituyen una idealización de una persona externa, de un proceso, o de una cosa que interactúa con un sistema, un subsistema, o una clase. Puede ser un ser humano, otro sistema informático, o un cierto proceso ejecutable.
[14] Unidad Administradora Central
[15] Unidad Ejecutora Local
[16] Unidad Administradora Desconcentrada

***Formular Presupuesto por Acción Centralizada*:** Este caso de uso consiste en formular el presupuesto de una Acción Centralizada determinada en la Unidad Administradora Central.

***Gestionar Formulación de un Proyecto*:** Este caso de uso consiste en configurar los datos necesarios para crear el Presupuesto de un Proyecto.

***Gestionar Formulación de una Acción Centralizada*:** Este caso de uso consiste en configurar los datos necesarios para crear el Presupuesto de una Acción Centralizada.

En la Figura 2.12, se muestra el Diagrama de Clases "Fuentes de Financiamientos". Los demás se encuentran dentro de la carpeta adjunta a la tesis "Formular Presupuesto en la UAC"

- **Diagrama de Clases "Formulación del Presupuesto Proyecto"** (Ver fichero "DC_Formulación del Presupuesto Proyecto" en la carpeta adjunta a la tesis).
- **Diagrama de Clases "Formulación del Presupuesto Acciones Centralizadas"** (Ver fichero "DC_Formulación del Presupuesto Acciones Centralizadas" en la carpeta adjunta a la tesis).
- **Diagrama de Clases "Formulación del Presupuesto Fuentes de Financiamiento"** (Ver fichero "DC_Formulación del Presupuesto Fuentes de Financiamiento" en la carpeta adjunta a la tesis).
- **Diagrama de Clases "Formulación del Presupuesto Indicadores Generales"** (Ver fichero "DC_Formulación del Presupuesto Indicadores Generales" en la carpeta adjunta a la tesis).
- **Diagrama de Secuencia "Fuentes de Financiamientos"** (Ver Anexo 4)
- **Diagrama de Secuencia "Formulación del Presupuesto Proyecto"** (Ver carpeta "DS_Formulación del Presupuesto Proyecto"en la carpeta adjunta a la tesis).
- **Diagrama de Secuencia "Formulación del Presupuesto Acciones Centralizadas"** (Ver carpeta "DS_Formulación Presupuesto Acciones Centralizadas" en la carpeta adjunta a la tesis).
- **Diagrama de Secuencia "Formulación del Presupuesto Fuentes de Financiamiento"** (Ver carpeta "DS_Formulación Presupuesto Fuentes de Financiamiento" en la carpeta adjunta a la tesis).
- **Diagrama de Secuencia "Formulación del Presupuesto Indicadores Generales"** (Ver carpeta "DS_Formulación del Presupuesto Indicadores Generales" en la carpeta adjunta a la tesis).

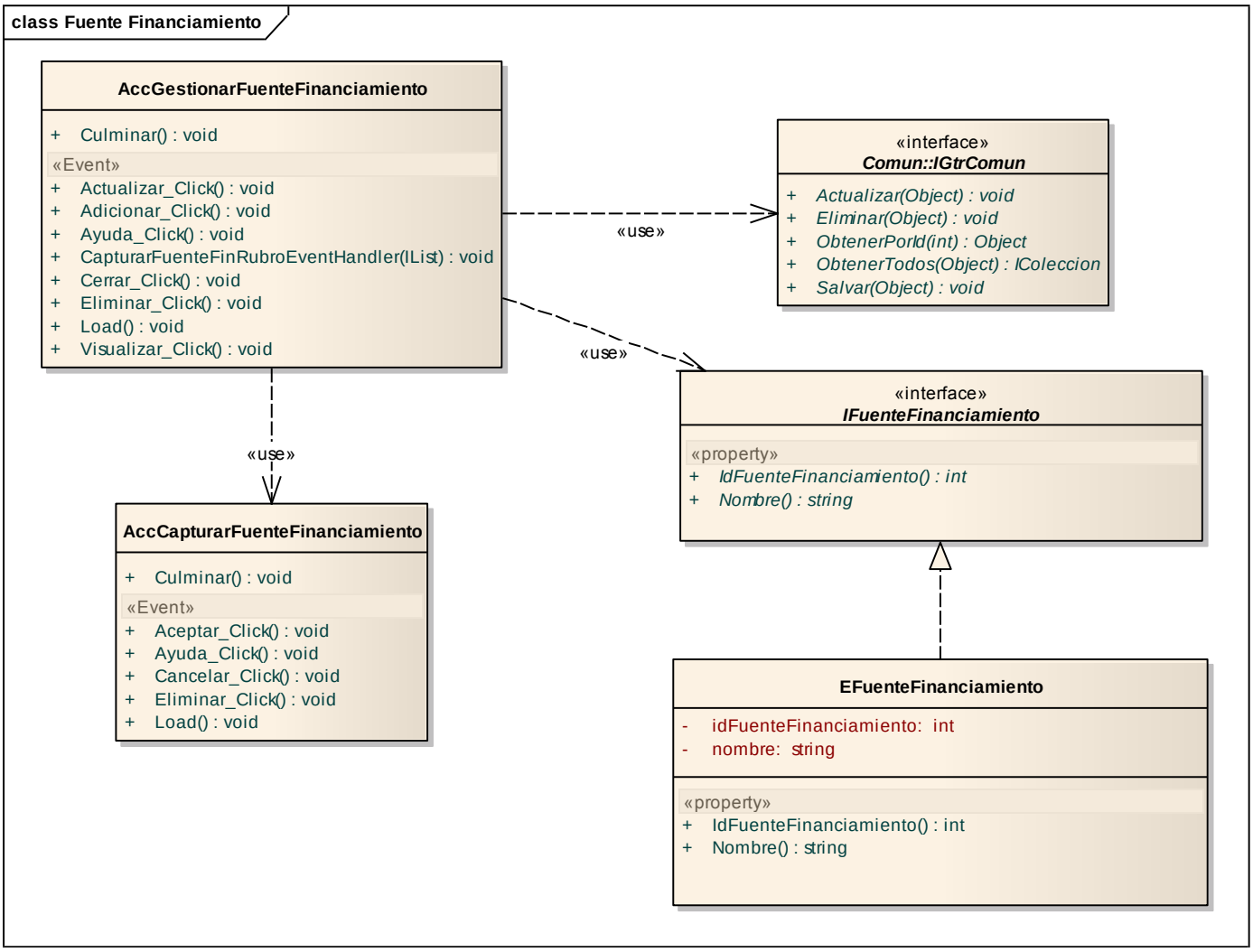


Figura 2.12. Diagrama de Clases "Fuentes de Financiamientos"

2.3.3.2. Formular Presupuesto en la Unidad Ejecutora Local/Unidad Administradora Desconcentrada

Es iniciado por el Funcionario Administrativo en la Unidad Ejecutora Local y consiste en formular el Presupuesto en la Unidad Ejecutora Local/Unidad Administradora Desconcentrada. Contiene dos casos de uso importantes para la arquitectura, los que se muestran a continuación.

***Formular Presupuesto por Proyecto en la UEL/UAD*:** Este caso de uso consiste en la formulación del Proyecto de Presupuesto desde las Unidades Ejecutoras Locales o las Unidades Administradoras Desconcentradas.

Formular Presupuesto por Acciones Centralizadas en la UEL/UAD: Este caso de uso consiste en modificar las Acciones Centralizadas para el Anteproyecto de Presupuesto en la Unidad Ejecutora Local/Unidad Administradora Desconcentrada.

En la Figura 2.13 se muestra el Diagrama de Clases "Formulación del Presupuesto UEL/UAD - Indicadores Generales". Los demás se encuentran dentro de la carpeta adjunta a la tesis "Formular Presupuesto en la UEL-UAD"

- **Diagrama de Clases "Proyecto(s)"** (Ver fichero "DC_Proyecto(s)" en la carpeta adjunta a la tesis).
- **Diagrama de Secuencia "Formulación"** (Ver fichero "DS_Formulación"en la carpeta adjunta a la tesis).
- **Diagrama de Secuencia "Proyecto(s)"** (Ver carpeta "DS_Proyecto(s)" en la carpeta adjunta a la tesis).

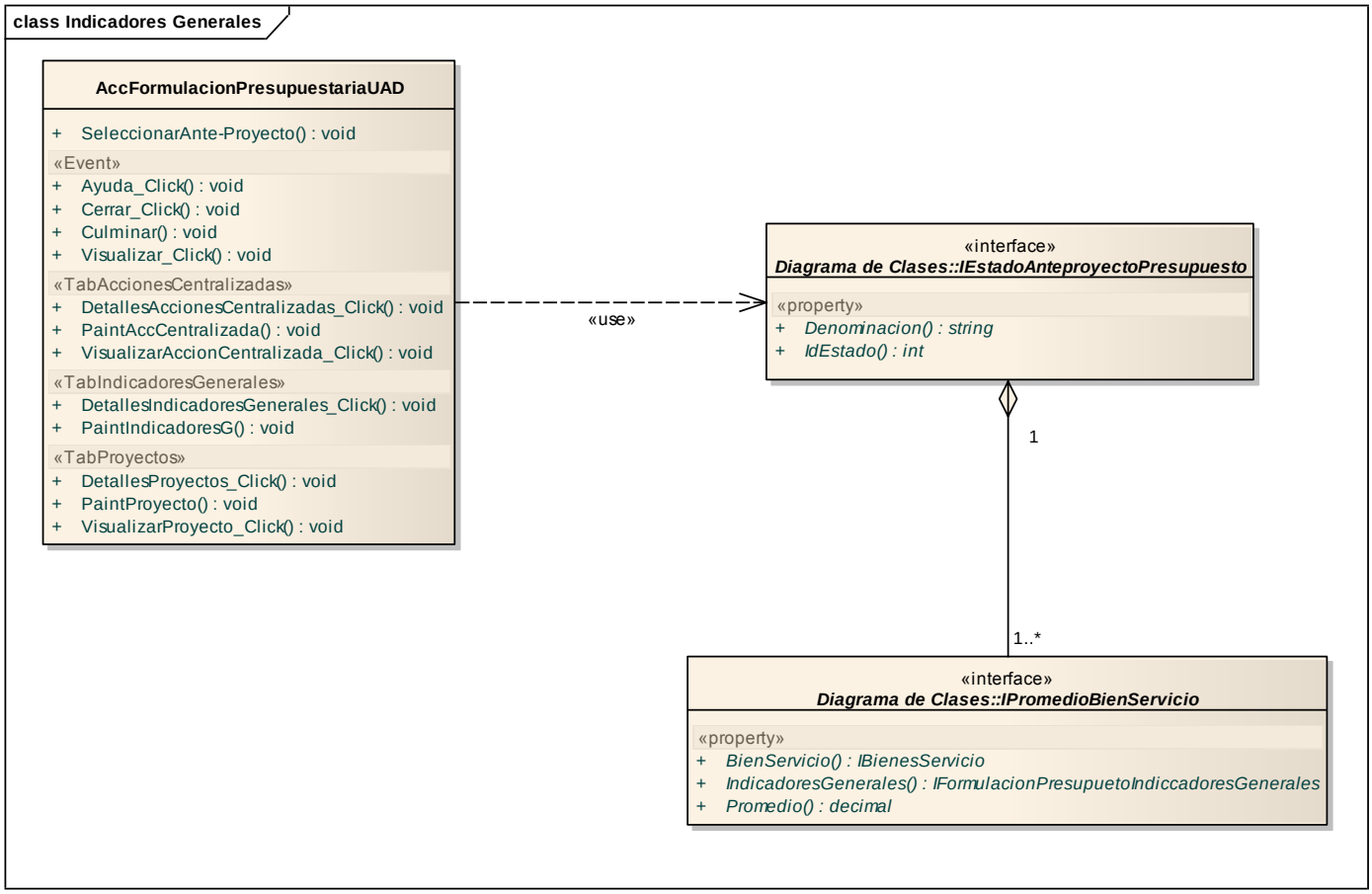


Figura 2.13. Diagrama de Clases "Formulación del Presupuesto UEL/UAD - Indicadores Generales"

2.3.3.3. Gestionar Modificación Presupuestaria.

Es iniciado por el Funcionario Administrativo en la Unidad Ejecutora Local y consiste en gestionar las Modificaciones Presupuestarias solicitadas por las Unidades Ejecutoras Locales. Contiene un caso de uso extendido el cual se muestra a continuación:

Crear Modificación Presupuestaria según tipo de modificación: Este caso de uso consiste en crear la Modificación Presupuestaria dependiendo del tipo que sea.

Los Diagramas de Clases y de Secuencia se encuentran en la carpeta adjunta al documento "Gestionar Modificación Presupuestaria"

- **Diagrama de Clases "Gestionar Modificaciones Presupuestarias"** (Ver fichero "DC_Gestionar Modificaciones Presupuestarias" en la carpeta adjunta a la tesis).
- **Diagrama de Secuencia "Modificaciones Presupuestarias"** (Ver carpeta "DS_Gestionar Modificaciones Presupuestarias" en la carpeta adjunta a la tesis).

2.3.3.4. Programación Financiera

Es iniciado por el Funcionario de Planificación y Presupuesto y consiste en hacer una programación del año que se va a presupuestar. Contiene dos casos de uso arquitectónicamente significativos, los que se describen a continuación:

Programar la Ejecución Financiera por Trimestre: Este caso de uso consiste en hacer una Programación por Trimestre del año que se va a presupuestar teniendo en cuenta la Ejecución Financiera por Proyectos y por Acciones Centralizadas.

Programar Ingresos y Desembolsos mensualmente: Este caso de uso consiste en hacer una Programación Mensual de los Ingresos y Desembolsos del año que se está presupuestando.

Los Diagramas de Clases y de Secuencia se encuentran en la carpeta adjunta al documento "Programación Financiera"

- **Diagrama de Clases "Gestionar Programación Financiera"** (Ver fichero "DC_Gestionar Programación Financiera" en la carpeta adjunta a la tesis).
- **Diagrama de Clases "Gestionar Programación Mensual de Ingresos y Desembolsos"** (Ver fichero "DC_Gestionar Programación Mensual de Ingresos y Desembolsos" en la carpeta adjunta a la tesis).
- **Diagrama de Secuencia "Programación Financiera-Ejecución Financiera"** (Ver carpeta "DS_Programación Financiera-Ejecución Financiera" en la carpeta adjunta a la tesis).

- **Diagrama de Secuencia "Programación Mensual Ingresos-Desembolsos"** (Ver carpeta "DS_Programación Mensual Ingresos-Desembolsos" en la carpeta adjunta a la tesis).

2.3.4. Modelo de Datos

El Modelo de Datos puede ser inicialmente creado a través de la Ingeniería Inversa de datos persistentes almacenados (Base de Datos) ya existentes o puede ser inicialmente creado a partir de un conjunto de Clases del Diseño persistentes en el Modelo de Diseño.

El Modelo de Datos desarrollado para el módulo Presupuesto ha sido concebido tomando en cuenta las Clases del Diseño persistentes en el Modelo de Diseño. El mismo está dividido en cuatro unidades diferentes:

- Codificadores
- Modificación Presupuestaria
- Formulación del Presupuesto
- Programación de la Ejecución Financiera

Codificadores

Entre sus funciones, define los estados por los que puede pasar un Proyecto, y las áreas estratégicas que éstos pueden abarcar. Registra la información referente a los Proyectos, así como las Acciones Centralizadas (generaliza las Acciones Centralizadas y los Proyectos en una sola entidad).

Modificación Presupuestaria

Registra los tipos de modificaciones que se le pueden hacer al presupuesto, así como el estado en que éstas se encuentran, registra además los comportamientos que puede sufrir una Partida en una determinada modificación, al igual que los cambios que sufre una Partida al ser sometida a una modificación del presupuesto.

Formulación del Presupuesto

Registra la transición de los estados del Anteproyecto de Presupuesto con una breve descripción a la vez que los va definiendo. Registra los precios promedios de los Bienes y Servicios, como parte de los Indicadores Generales de la Formulación del Anteproyecto de Presupuesto para el Ejercicio Fiscal seleccionado.

Programación

Define los estados por los que puede pasar la Programación del Presupuesto, y los trimestres. Registra la Programación Trimestral de las Metas y la Ejecución Financiera más la Acción Específica de una Acción Centralizada, sus partidas, la Acción Específica de un Proyecto y sus Partidas.

- **Modelo de Datos del módulo "Presupuesto"** (Ver carpeta "Modelo de Datos Presupuesto" en la carpeta adjunta a la tesis).

2.3.5. Diagrama de Despliegue

El Diagrama de Despliegue constituye un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos (representan recursos de cómputo, normalmente un procesador o un dispositivo hardware similar). Se utiliza como entrada fundamental en las actividades de Diseño e Implementación ya que la distribución del sistema final ejerce una gran influencia en su diseño. Puede describir diferentes configuraciones de red, incluidas las configuraciones para pruebas y para simulación.

Seguidamente, en la Figura 2.14, se muestra el Diagrama de Despliegue perteneciente al módulo "Presupuesto".

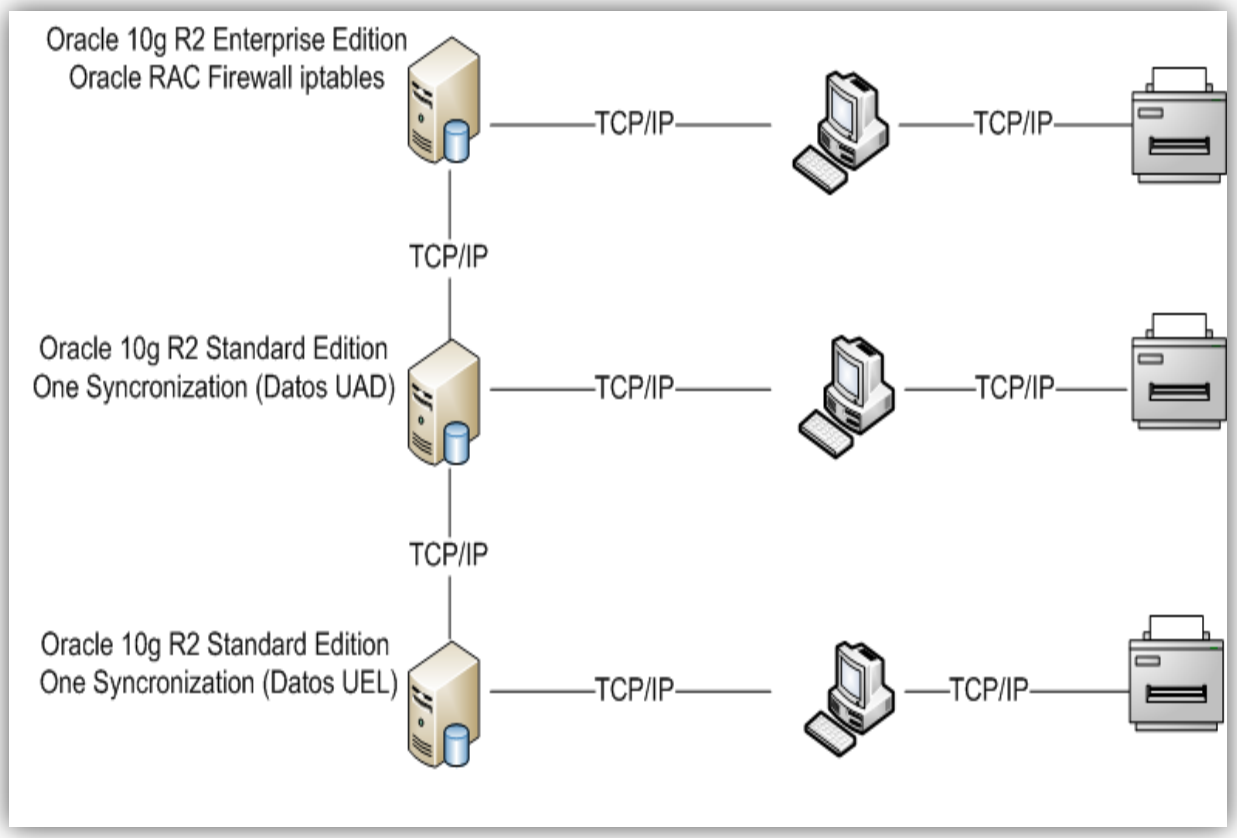


Figura 2.14. Diagrama de Despliegue correspondiente a "Presupuesto"

2.4. Implementación del sistema

Según RUP, la Implementación es el centro durante la Fase de Construcción, aunque también se lleva a cabo durante la Fase de Elaboración para crear la línea base ejecutable de la arquitectura y durante la de Transición para tratar defectos que se hayan encontrado en ese momento, en caso de que existan.

El resultado del Diseño constituye la entrada fundamental de la Implementación comenzándose a implementar el sistema en términos de componentes, es decir, ficheros de código fuente o binario, scripts, ejecutables o similares. La mayor parte de la arquitectura es capturada durante el Diseño, siendo el propósito fundamental de la Implementación desarrollar la arquitectura y el sistema como un todo. Aquí se implementan las clases elaboradas durante el Diseño como componentes de fichero que contienen código fuente. (Jacobson, y otros, 2004)

2.4.1. Componentes desarrollados

Se cuenta con una serie de componentes visuales y no visuales que facilitan la programación, realizando todas las restricciones y validaciones en tiempo de diseño de la aplicación. En tiempo de ejecución, estas validaciones se verificarán de forma autónoma por los componentes para que sean cumplidas, lo que facilita al programador la posibilidad de sólo dedicarse a la programación de las funcionalidades del caso de uso, olvidándose por completo de las validaciones de los datos entrados por el usuario.

Reducen considerablemente el número de líneas de código haciendo que éste sea más comprensible para los programadores. Entre estos componentes se encuentran:

CTextBox: Encargado de la entrada de datos alfanuméricos y numéricos, para entrar cadenas (string), números (int16, int32, int 64, double, decimal y monedas). Permite realizar validaciones mediante condiciones (CCondicion), con las cuales se puede validar que los valores estén en un rango determinado o sean distintas de un valor determinado, dando la posibilidad de poner condiciones complejas. Presenta varias propiedades nuevas que son muy utilizadas en la aplicación, las fundamentales son:

- *ColorMal*: Se utiliza para resaltar datos inválidos o cuando se deja en blanco el CTextBox en tiempo de ejecución. Está por defecto inicializada en “rojo” para indicar que no se ha llenado el campo.
- *Condicion*: Es donde se asocia el componente CCondicion que necesita.

- *DígitosDecimales*: Permite establecer la cantidad de dígitos decimales permitidos, en caso de que el campo asociado sea numérico.
- *Filtro*: Esta se considera una de las propiedades más importantes desarrolladas en el componente ya que permite filtrar los datos estableciendo en tiempo de diseño, los diferentes caracteres convenientes en el campo que se quiera representar, por ejemplo: si se va a representar el nombre de una persona, en el filtro se le valida que solamente se le puedan entrar letras y el espacio, no permitiendo los números, símbolos y otros caracteres especiales.
- *MultiEspacio*: Esta no permite entrar varios espacios consecutivos, si se inicializa en "False".
- *Propiedad*: En esta se indica la propiedad (Property) asociada al campo que se quiera representar.

CCheckBox: Encargado de la entrada de parámetros booleanos, verdadero o falso.

- *Propiedad*: En esta se especifica la propiedad (Property) asociada al campo que se quiere representar.

CListView: Es Orientado a Objetos y se encarga de visualizar y manipular de forma automática colecciones de objetos. Entre las propiedades desarrolladas se encuentran:

- *ActivarEn*: Se usa para activar el CListView en la forma establecida.
- *ActivarPrimera*: Con esta se especifica si se activa la primera fila con un clic o no.
- *Caminar*: Establece si se puede caminar con la tecla "Enter" o no.
- *ControlColumns:* Cuando el ListView es editable (lo que significa que se pueden establecer los valores de sus columnas dentro de él mismo) es donde se utiliza esta propiedad la cual da la posibilidad de establecer los controles que se van a asociar a cada columna en dicho ListView.

CManagerTextCheckBox: Encargado de manipular todos los controles (CTextBox y CCheckBox) asociados al mismo como un todo, permitiendo conocer el estado de los datos, si son o no válidos. Presenta la propiedad *Forma* la cual da la posibilidad de asociarle la forma donde se van a encontrar todos los componentes (CTextBox y CCheckBox) al cual él va a estar relacionado y *SetFocus donde se e*stablece el foco sobre el componente con error.

CCondicion: Encargado de crear las validaciones y determinar que sean cumplidas.

- *Asociación*: Indica si se asocian por And/Or las Reglas/SubReglas.
- *Reglas*: Colección de reglas. Las reglas pueden ser un operador de comparación más un valor, los operadores pueden ser:
    1. Mayor

2. Menor
3. Igual
4. MayorIgual
5. MenorIgual
6. NoIgual

Ejemplo Regla: (NoIgual,15 && Mayor, 10 && Menor,100)

- *GReglas*: Colección de sub-reglas, cada sub-regla puede añadir nuevas reglas y sub-reglas. Ejemplo:(Mayor, 5 || MenorIgual, 30 || (NoIgual, 20 && NoIgual,15))

CButton: Se integra con el CManagerTextCheckBox, permitiendo así validar antes de ejecutar el evento si se cumplen las validaciones de los componentes de entrada (CTextBox, CCheckBox). Entre las propiedades nuevas desarrolladas, las más importantes son:

- *CListViewAsociado*: Es donde se le indica el ListView al cual va a estar asociado ese botón para realizar su acción.
- *CListViewAsociadoCount*: Esta propiedad deshabilita el botón si está vacío el ListView.
- *LanzarExcepManager*: Lanza una excepción si el manager no es válido.
- *CManagerTextCheckBox*: Es donde se le asocia el manager con el cual va a relacionarse, en caso que sea necesario.

Nomenclador: Facilita el trabajo con las *tablas nomencladoras*[17] de la BD asegurando un conjunto de funcionalidades que resultan útiles para el trabajo con este tipo de datos en el proceso de Implementación. Su representación visual es igual a la del componente “ComboBox” que viene predeterminado en Visual Studio. Net Entre las propiedades que presenta, las más importantes son:

- *TipoOrigenDatos*: Aquí se especifica el origen de los datos que va a cargar el nomenclador, ya sea de una Vista o de un Procedimiento Almacenado.
- *NombreOrigenDatos*: Esta especifica el nombre de la tabla de donde cargará los datos el Nomenclador.
- *ColumnaMostrar*: Especifica el nombre de la columna que se quiere visualizar.
- *ColumnaIdentificador*: Especifica el nombre de la columna que es llave primaria en la tabla seleccionada.
- *CapturarParámetros*: Esta se utiliza para capturar los parámetros del procedimiento almacenado, en caso de que los tenga.

[17] Tablas estáticas generalmente gestionadas por el Administrador de Base de Datos y que habitualmente contienen pocos datos.

- *Activo*: Activa el Nomenclador para su uso.
- *FilaEnBlanco*: Aquí se especifica si el componente tendrá una primera fila vacía.
- *TextoFilaEnBlanco*: En caso de tener una fila en blanco, esta propiedad da la posibilidad de indicarle un texto para una mejor comprensión.

En las siguientes figuras se observa la diferencia del número de líneas de código usando los componentes y sin usarlos. En la Figura 2.14 (a y b) se aprecia un fragmento de código perteneciente al evento "CClick" de un botón donde no se hace uso de los componentes mencionados anteriormente. La Figura 2.15 (a y b), por otra parte, muestra como quedaría el mismo código usando estos componentes.

```
private void btnAceptarrr_CClick(object sender)
{
try
{
    decimal monto = LlenarConMontos();

    if(monto == Convert.ToDecimal(objForma.txtMontoTotal.Text))
    {
        MostrarReloj();
        objAnteProyectoAccionCentralizada.ObjAccionCentralizada.Codigo = objForma.txtCodigo.Text;
        objAnteProyectoAccionCentralizada.ObjAccionCentralizada.Denominacion = objForma.txtDenominacio.Text;
        objAnteProyectoAccionCentralizada.Responsable = objForma.txtResponsable.Text;
        objAnteProyectoAccionCentralizada.BienServicio = objForma.txtBienesServ.Text;
        objAnteProyectoAccionCentralizada.UnidadMedida = objForma.txtUniadM.Text;
        objAnteProyectoAccionCentralizada.ObjAccionCentralizada.Descripcion = objForma.txtDescripcion.Text;
        objAnteProyectoAccionCentralizada.MetaTotal = Convert.ToDecimal(objForma.txtMetaTotal.Text);
        objAnteProyectoAccionCentralizada.MontoTotal = Convert.ToDecimal(objForma.txtMontoTotal.Text);

        objAnteProyectoAccionCentralizada.ListaAnteProyectoAccionEspAccionCent =
            listaAnteProyectoAccionEspAccionCent;
        Fabrica.GetInstancia.CrearGtrAnteProyectoAccCentralizada().SalvarAnteAccionCent
            (objAnteProyectoAccionCentralizada);
```

Figura 2.14. a) Fragmento de código sin usar los componentes (47 líneas de código)

```
        if(capturarAnteProyectoAccionCentralizada != null)
        {
            capturarAnteProyectoAccionCentralizada(this, objAnteProyectoAccionCentralizada,
                ObtenerListaFF());
        }

        Mensajeria.Instancia.MostrarMensaje(15);
        OcultarReloj();
    }
    else
    {
        objForma.txtMontoTotal.BackColor = Color.Red;
        ArrayList datos = new ArrayList();
        datos.Add(objAnteProyectoAccionCentralizada.ObjAnteProyectoPresupuesto.ObjEjercicioFiscal.
            AnoEjercicioFiscal);
        datos.Add(monto.ToString("N"));
        Mensajeria.Instancia.MostrarMensaje(143, datos);
    }

}
catch(Exception ex)
{
    throw ex;
}
}
```

Figura 2.14. b) Fragmento de código sin usar los componentes (47 líneas de código) (continuación)

```
private void btnAceptarrr_CClick(object sender)
{
    try
    {
        decimal monto = LlenarConMontos();


        if(monto == Convert.ToDecimal(objForma.txtMontoTotal.Text))
        {
            MostrarReloj();
            objForma.mngAccCentralizada.GetValor(objAnteProyectoAccionCentralizada);
            objAnteProyectoAccionCentralizada.ListaAnteProyectoAccionEspAccionCent = objForma.
                lvwAccEspecificas.Objetos;
            Fabrica.GetInstancia.CrearGtrAnteProyectoAccCentralizada().SalvarAnteAccionCent
                (objAnteProyectoAccionCentralizada);

            if(capturarAnteProyectoAccionCentralizada != null)
            {
                capturarAnteProyectoAccionCentralizada(this, objAnteProyectoAccionCentralizada,
                    ObtenerListaFF());
            }
```

Figura 2.15. a) Fragmento de código usando los componentes (39 líneas de código)

```
            Mensajeria.Instancia.MostrarMensaje(15);
            OcultarReloj();
        }
        else
        {
            objForma.txtMontoTotal.BackColor = Color.Red;
            ArrayList datos = new ArrayList();
            datos.Add(objAnteProyectoAccionCentralizada.ObjAnteProyectoPresupuesto.ObjEjercicioFiscal.
                AnoEjercicioFiscal);
            datos.Add(monto.ToString("N"));
            Mensajeria.Instancia.MostrarMensaje(143, datos);
        }

    }
    catch(Exception ex)
    {
        throw ex;
    }
```

Figura 2.15. b) Fragmento de código usando los componentes (39 líneas de código) (continuación)

En la Figura 2.16 (a y b) se muestra el fragmento del evento “Load” de un formulario sin hacer uso de los componentes desarrollados y en la Figura 2.17 se observa este mismo evento pero haciendo uso de estas nuevas funcionalidades.

```
private void objForma_Load(object sender, EventArgs e)
    {
        objForma.txtCodigo.Text = objAnteProyectoAccionCentralizada.ObjAccionCentralizada.Codigo ;
        objForma.txtDenominacio.Text = objAnteProyectoAccionCentralizada.ObjAccionCentralizada.Denominacion;
        objForma.txtResponsable.Text = objAnteProyectoAccionCentralizada.Responsable;
        objForma.txtBienesServ.Text = objAnteProyectoAccionCentralizada.BienServicio;
        objForma.txtUniadM.Text = objAnteProyectoAccionCentralizada.UnidadMedida;
        objForma.txtDescripcion.Text = objAnteProyectoAccionCentralizada.ObjAccionCentralizada.Descripcion;
        objForma.txtMetaTotal.Text = objAnteProyectoAccionCentralizada.MetaTotal.ToString();
        objForma.txtMontoTotal.Text = objAnteProyectoAccionCentralizada.MontoTotal.ToString();

        ArrayList asignar  = new ArrayList();

        if((objAnteProyectoAccionCentralizada.ListaAnteProyectoAccionEspAccionCent != null) &&
            (objAnteProyectoAccionCentralizada.ListaAnteProyectoAccionEspAccionCent.Count > 0))
            asignar = objAnteProyectoAccionCentralizada.ListaAnteProyectoAccionEspAccionCent;
```

Figura 2.16. a) Fragmento de código sin usar los componentes (34 líneas de código)

```
    else
    {
        asignar = Fabrica.GetInstancia.CrearGtrAnteProyectoAccCentralizada().
            ObtenerAnteProyectoAccEspAccCentralizada(objAnteProyectoAccionCentralizada.
                                                    IdAnteProyectoAccionCentralizada);
        for (int i = 0; i < asignar.Count; i++)
            (asignar[i] as IAnteProyectoAccionEspAccionCent).ObjAnteProyectoAccionCentralizada =
                objAnteProyectoAccionCentralizada;
    }

    foreach (IAnteProyectoAccionEspAccionCent a in asignar)
    {
        ListViewItem item = new ListViewItem(new string[]{a.BienServicio,a.MontoTotal.ToString(),
                                                          a.MetaTotal.ToString(),a.UnidadMedida,
                                                          a.Responsable});
        objForma.lvwAccEspecificas.Items.Add(item);
    }

}
```

Figura 2.16. b) Fragmento de código sin usar los componentes (34 líneas de código)(continuación)

```
private void objForma_Load(object sender, EventArgs e)
{
    objForma.mngAccCentralizada.Add(objForma.txtCodigo);
    objForma.mngAccCentralizada.Add(objForma.txtDenominacio);
    objForma.mngAccCentralizada.Add(objForma.txtResponsable);
    objForma.mngAccCentralizada.Add(objForma.txtBienesServ);
    objForma.mngAccCentralizada.Add(objForma.txtUniadM);
    objForma.mngAccCentralizada.Add(objForma.txtDescripcion);
    objForma.mngAccCentralizada.Add(objForma.txtMetaTotal);
    objForma.mngAccCentralizada.Add(objForma.txtMontoTotal);

    objForma.mngAccCentralizada.SetValor(objAnteProyectoAccionCentralizada);

    ArrayList asignar  = new ArrayList();
    if((objAnteProyectoAccionCentralizada.ListaAnteProyectoAccionEspAccionCent != null) &&
        (objAnteProyectoAccionCentralizada.ListaAnteProyectoAccionEspAccionCent.Count > 0))
        asignar = objAnteProyectoAccionCentralizada.ListaAnteProyectoAccionEspAccionCent;
    else
    {
        asignar = Fabrica.GetInstancia.CrearGtrAnteProyectoAccCentralizada().
            ObtenerAnteProyectoAccEspAccCentralizada(objAnteProyectoAccionCentralizada.
                                                     IdAnteProyectoAccionCentralizada);
        for (int i = 0; i < asignar.Count; i++)
            (asignar[i] as IAnteProyectoAccionEspAccionCent).ObjAnteProyectoAccionCentralizada =
                objAnteProyectoAccionCentralizada;
    }

    objForma.lvwAccEspecificas.Objetos = asignar;

}
```

Figura 2.17. Fragmento de código usando los componentes (29 líneas de código)

2.4.2. Estándares de Codificación

Los Estándares de Codificación son reglas específicas a un lenguaje que reducen perceptiblemente el riesgo de que los desarrolladores introduzcan errores. Durante el desarrollo de un software, estos estándares ayudan a los ingenieros a producir un código de alta calidad y a entender y a utilizar el código de sus colegas. Pero también realzan considerablemente la capacidad de mantenimiento y reusabilidad a largo plazo del producto final. (SYNSPACE AG, 2005)

A partir de esto se utilizó en la solución Estándares de Codificación puesto que reducen la posibilidad de introducir errores en el código así como precisar algunos de éstos que estén ocultos o sean inesperados, pero, sobre todo, porque hace al código más legible y fácil de leer ayudando así al posterior mantenimiento del software.

2.4.2.1. Reglas a seguir

1. Los nombres de las clases deben ser sustantivos o frases de sustantivos y no se deben usar prefijos.
2. Para las clases interfaces, utilizar sustantivos, frases en sustantivos o adjetivos que describan comportamiento.
3. Todas las clases deben tener prefijos que las identifiquen (como se muestran en la siguiente tabla) y seguidos de una letra mayúscula que sería la primera del nombre de la clase.

Clases	Prefijos
Formularios	Frm
Acciones	Acc
Interfaces	I
Entidades	E
Gestores	Gtr
Objeto de Acceso a Datos	Dao

4. En los enumerados se recomienda no poner prefijos(sufijos) a los valores ni al identificador, el cual tiene que estar en singular.
5. Para los campos de solo lectura y constantes se deben utilizar sustantivos o frases de éstos.
6. Para los parámetros y campos no constantes se deben usar nombres descriptivos que deben ser lo suficientemente explícitos para determinar el significado de la variable y su tipo, preferentemente su significado.

7. Utilizar i, j, k, l, m, n para contadores en ciclos triviales, en caso contrario usar nombres de variables más descriptivos.
8. Los métodos deben ser nombrados con verbos o frases verbales.
9. Las propiedades deben ser nombradas utilizando sustantivos o frases en sustantivo considerando siempre que sean nombradas con el mismo nombre de su tipo.
10. Nombrar los manejadores de eventos con el sufijo EventHandler y las clases que sean para pasar argumentos con EventArgs. Denominar los eventos que utilizan el concepto de presente y pasado utilizando el tiempo en que ocurren y usar dos parámetros nombrados "sender" y "e".
11. No hacer público ninguna instancia o campo de una clase, deben ser privados.
12. Excepto el código relacionado con la interfaz gráfica de la aplicación, todos los nombres de variables y campos que contengan elementos de interfaz como botones, etiquetas, etc., deben tener al principio la abreviatura del tipo como se muestra en la siguiente tabla:

Tipo	Abreviatura
System.Windows.Forms.Button	btn
System.Windows.Forms.TextBox	txt
System.Windows.Forms.ComboBox	cbx
System.Windows.Forms.ListBox	lbx
System.Windows.Forms.ListView	lvw
System.Windows.Forms.DateTimePicker	dtp
System.Windows.Forms.Label	lbl

13. Todas las declaraciones se deben hacer en idioma Español para un mejor entendimiento y utilizando el estilo Pascal[18]; excepto los campos protegidos y privados, las variables locales y los parámetros, que deben ser en estilo Camello[19].

2.4.3. Diagrama de Componentes

El uso más importante que tiene este tipo de diagrama en el Proceso Unificado de Desarrollo de Software es mostrar la estructura de alto nivel del Modelo de Implementación, específicamente los Subsistemas de Implementación y sus dependencias de importación, además, que éstos estén organizados en capas.

También se usan para mostrar ficheros de código fuente y sus dependencias de compilación; ficheros de aplicación y sus dependencias en tiempo de ejecución; relaciones derivadas entre ficheros de código fuente y ficheros resultantes de la compilación; dependencias de implementación entre

[18] Capitaliza la primera letra de cada palabra, ejemplo: AccProyecto.
[19] Capitaliza la primera letra de cada palabra, excepto para la primera palabra, ejemplo: objProyecto.

Elementos de Implementación y Elementos de Diseño que ellos implementan. (Rational Software Corporation, 2003)

A continuación se muestra, en la Figura 2.18 el Diagrama de Componentes correspondiente al módulo "Presupuesto".

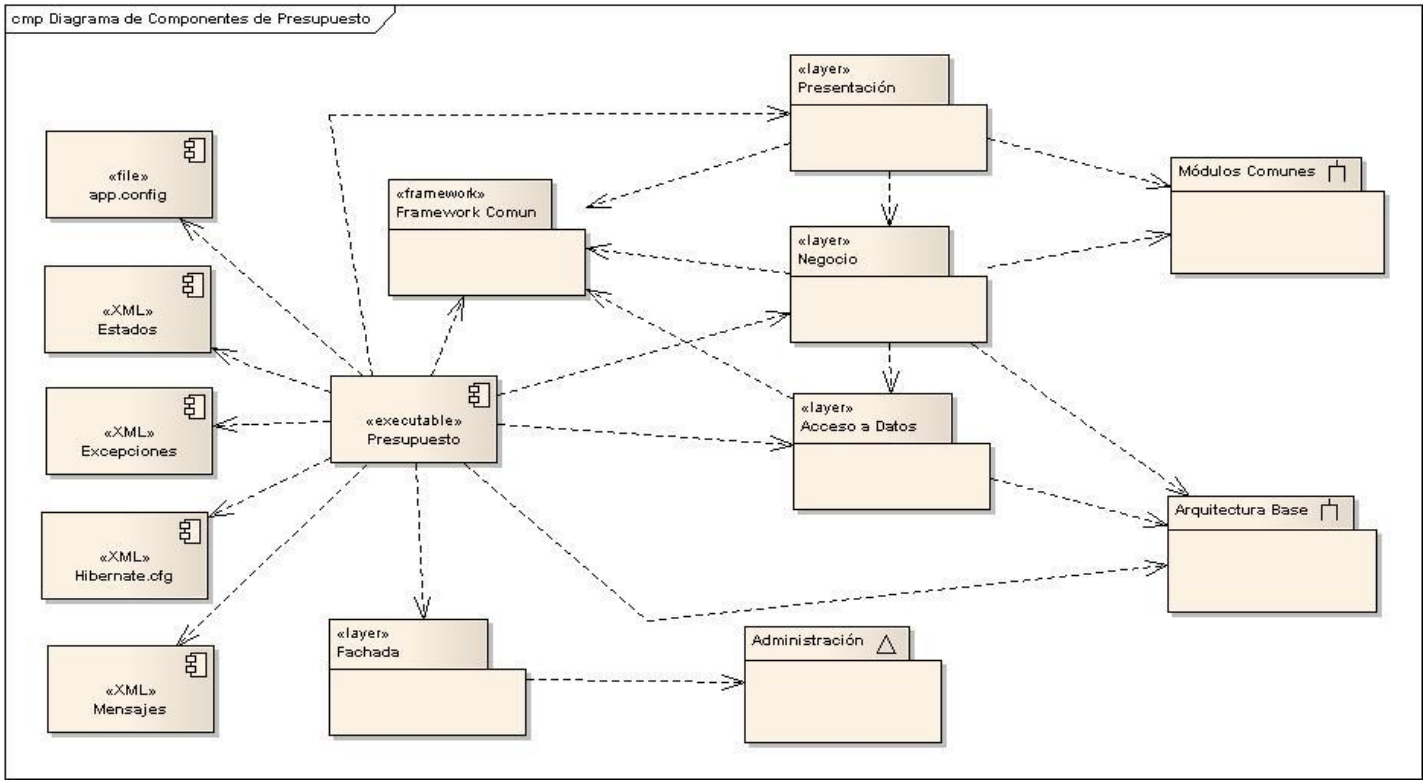


Figura 2.18. Diagrama de Componentes correspondiente al módulo "Presupuesto"

Conclusiones

A lo largo de este capítulo se revelaron los resultados de las disciplinas de Diseño e Implementación en el proceso de desarrollo del módulo "Presupuesto". Se desarrollaron los Diagramas de Clases y de Secuencia de los Casos de Uso fundamentales en el sistema, así como los tipos de clases que existen en el diseño de la solución y el Modelo de Datos. Se expuso también un grupo de componentes desarrollados para ser utilizados en el sistema, además de una serie de reglas que se asumieron para la codificación del software. Como complemento se brinda también el Diagrama de Componentes y el de Despliegue.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

Introducción

En el presente capítulo se establecen criterios asociados a la validación del sistema propuesto, partiendo del estudio y puesta en práctica de los métodos de Caja Negra y Caja Blanca. Para lograr este propósito se formularon los casos de pruebas afines con los Casos de Usos más significativos para la arquitectura, teniendo en cuenta para su construcción la Técnica de Partición de Equivalencia. Se aplicaron además técnicas referentes a las Pruebas de Caja Blanca, en las que se hace referencia principalmente a la Prueba del Camino Básico y la Métrica de la Complejidad Ciclomática, definiendo el grado de complejidad de dicho sistema.

3.1. Pruebas de Caja Negra

La Prueba de Caja Negra es aquella que se lleva a cabo sobre la interfaz del software, centrándose en los requisitos funcionales de éste, para demostrar que las funciones del sistema sean operativas, que las entradas se acepten de forma correcta y que se produzcan resultados adecuados. Este tipo de pruebas examina algunos aspectos del modelo fundamental del sistema sin tener mucho en cuenta la estructura lógica interna del software.

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

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 que los métodos de Caja Blanca.

Dentro de las Técnicas de Prueba de Caja Negra se utilizó la de Partición de Equivalencia. Ésta divide el dominio de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba.

El diseño de casos de prueba para este método se basa en una evaluación de las clases de equivalencia para una condición de entrada. Una *clase de equivalencia* representa un conjunto de estados válidos o inválidos para diferentes *condiciones de entrada*[20].

[20]Constituye un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

Para definir las clases de equivalencia se tuvieron en cuenta las siguientes reglas:

- Si una condición de entrada especifica un rango, entonces se confeccionan una clase de equivalencia válida y dos inválidas.
- Si una condición de entrada requiere un valor específico, se identifica una clase de equivalencia válida y dos inválidas.
- Si una condición de entrada especifica un miembro de un conjunto, se identifica una clase válida para cada uno de ellos y una clase inválida.
- Si una condición de entrada es lógica, se define una clase válida y una inválida.

(Pressman, 2002)

3.2. Casos de Prueba principales

Luego de tener las clases válidas e inválidas definidas, se procedió a definir los Casos de Prueba, para los Casos de Uso más significativos que se encuentran organizados en cuatro unidades diferentes: Formular Presupuesto en la UAC, Formular Presupuesto en la UEL/UAD, Gestionar Modificaciones Presupuestarias y Programación Financiera.

3.2.1. Formular Presupuesto en la UAC

3.2.1.1. CU Gestionar Fuentes de Financiamiento

A este Caso de Uso se le realizaron las pruebas de Adicionar, Modificar y Eliminar una Fuente de Financiamiento.

CPR 1: Adicionar una Fuente de Financiamiento

Flujo Central:

1. El responsable ordena gestionar las Fuentes de Financiamientos.
2. El sistema muestra la Interfaz.
3. El responsable ordena adicionar una Fuente de Financiamientos.
4. El sistema muestra la Interfaz Fuente de Financiamiento para seleccionar la Fuente de Financiamientos.
5. El responsable selecciona la Fuente de Financiamiento.
6. El sistema muestra todos los rubros asociados a la Fuente de Financiamientos seleccionada.
7. El responsable edita el monto de cada Rubro.
8. El responsable ordena aceptar la operación.

9. El sistema acepta la operación adicionándose en la interfaz anterior la Fuente de Financiamiento terminando así el caso de uso.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El botón Adicionar (+) está activado.		El sistema permite adicionar.	Satisfactorio	
El campo monto sólo permite números		El sistema sólo permite números en el campo monto	Satisfactorio	
	El campo monto permite letras, caracteres especiales y símbolos. Ej: "@^[¿abc	El sistema no permite letras, caracteres especiales y símbolos en el campo monto	Satisfactorio	
La interfaz Seleccionar Fuente de Financiamiento se cierra al oprimir el botón Cancelar.		El sistema cierra la interfaz	Satisfactorio	
El sistema guarda los cambios al oprimir el botón Aceptar de la interfaz Seleccionar Fuente de Financiamiento.		El sistema guarda los cambios efectuados en los montos.	Satisfactorio	

CPR 2: Eliminar una Fuente de Financiamiento.

Flujo Central:

1. El responsable selecciona la Fuente de Financiamiento que desea eliminar.
2. El responsable ordena eliminar la Fuente de Financiamiento seleccionada.
3. El sistema elimina la Fuente de Financiamiento.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El botón Eliminar (-) permanece inhabilitado hasta que se seleccione la Fuente de Financiamiento.		Seleccionar la Fuente de Financiamiento antes de oprimir el botón Eliminar.	Satisfactorio.	
Al oprimir el botón eliminar el sistema muestra un mensaje de confirmación		El sistema muestra un mensaje de confirmación	Satisfactorio	

Al oprimir el botón Aceptar del mensaje de confirmación el sistema elimina la fuente de financiamiento.		El sistema elimina la fuente de financiamiento.	Satisfactorio	

CPR 3: Modificar una Fuente de Financiamiento.

Flujo Central:

1. El responsable selecciona la Fuente de Financiamiento que desea modificar.
2. El responsable ordena modificar.
3. El sistema muestra la interfaz Fuente de Financiamiento con los datos de la Fuente de Financiamiento seleccionada.
4. El responsable modifica los datos necesarios.
5. El responsable ordena la opción Aceptar.
6. El sistema actualiza la Fuente de Financiamiento con los datos modificados. Cierra la interfaz volviendo a la interfaz anterior.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
Selecciona la opción cancelar.		No realiza ningún cambio y cierra la vista correspondiente	Satisfactorio	
El botón Modificar (-) permanece inhabilitado hasta que se seleccione la Fuente de Financiamiento.		Seleccionar la Fuente de Financiamiento antes de oprimir el botón Modificar.	Satisfactorio.	
Al oprimir el botón Modificar el sistema muestra una interfaz con todos los datos de la fuente de financiamiento seleccionada.		Muestra la interfaz con los datos a modificar	Satisfactorio	
El sistema sólo permite modificar el monto de la fuente de financiamiento		El sistema permite modificar el monto	Satisfactorio	
El campo monto sólo permite números		El sistema sólo permite números en el campo monto	Satisfactorio	
	El campo monto permite letras, caracteres especiales y símbolos.	El sistema no permite letras, caracteres	Satisfactorio	

	Ej: “@^[¿abc	especiales y símbolos en el campo monto		
El sistema guarda los cambios al oprimir el botón Aceptar de la interfaz Seleccionar Fuente de Financiamiento.		El sistema guarda los cambios efectuados en los montos.	Satisfactorio	

3.2.1.2. CU Formular Presupuesto por Proyecto

A este Caso de Uso se le realizaron las pruebas de Adicionar, Modificar y Eliminar un Proyecto.

CPR 1: Adicionar un Proyecto

Flujo Central:

1. El responsable ordena formular el Presupuesto por Proyecto.
2. El sistema muestra en la interfaz todos los proyectos que han sido creados.
3. El responsable ordena Adicionar un Proyecto.
4. El sistema muestra la interfaz Seleccionar Proyecto para seleccionar los proyectos para la Formulación del Presupuesto.
5. El responsable selecciona los proyectos.
6. El responsable ordena la opción Aceptar.
7. El sistema agrega los proyectos seleccionados a la Formulación del Presupuesto.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario ordena Cancelar		No realiza ningún cambio. Cierra la interfaz correspondiente y regresa a la interfaz anterior	Satisfactorio	
Al oprimir el botón Adicionar(+) el sistema muestra la interfaz correspondiente		El sistema muestra la interfaz para agregar un proyecto.	Satisfactorio	
Al oprimir el botón Aceptar de la interfaz el sistema agrega en la lista anterior el nuevo o los nuevos proyectos		El sistema agrega el nuevo proyecto.	Satisfactorio	

CPR 2: Eliminar un Proyecto.

Flujo Central:

1. El responsable selecciona el proyecto que desea eliminar.
2. El responsable ordena Eliminar.
3. El sistema elimina el proyecto seleccionado.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El botón Eliminar (-) permanece inhabilitado hasta que se seleccione un Proyecto		Se debe seleccionar primero el Proyecto antes de oprimir el botón Eliminar (-).	Satisfactorio	
Al oprimir el botón Eliminar (-) el sistema muestra un mensaje de verificación de la operación.		El sistema muestra un mensaje	Satisfactorio	
Al oprimir el botón Cancelar del mensaje el sistema no realiza ningún cambio.		El sistema no elimina el proyecto.	Satisfactorio	
Al oprimir el botón Aceptar del mensaje, el sistema elimina el Proyecto de la lista anterior.		El sistema elimina el Proyecto	Satisfactorio	

CPR 3: Modificar un Proyecto.

Flujo Central:

1. El responsable ordena Modificar el Proyecto.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El botón Modificar (<>) permanece inhabilitado hasta que se seleccione un Proyecto.		Se debe seleccionar primero el Proyecto antes de oprimir el botón Modificar (<>).	Satisfactorio	
Al oprimir el botón Modificar(<>) el sistema muestra la interfaz correspondiente para modificar los datos		El sistema muestra la interfaz para modificar un proyecto.	Satisfactorio	
El sistema permite modificar los datos de los campos deseados		El sistema permite modificar los datos de los campos deseados.	Satisfactorio	

Al oprimir el botón Aceptar el sistema actualiza los datos		El sistema actualiza los datos modificados.	Satisfactorio	
Al oprimir el botón Cerrar el sistema no modifica los datos y cierra la interfaz		El sistema cierra la interfaz.	Satisfactorio	

3.2.1.3. CU Formular Presupuesto por Acción Centralizada

A este Caso de Uso se le realizaron las pruebas de Adicionar, Modificar y Eliminar una Acción Centralizada.

CPR 1: Adicionar Acción Centralizada.

Flujo Central:

1. El responsable ordena Formular el Presupuesto por Acción Centralizada.
2. El sistema muestra en la interfaz todas las Acciones Centralizadas que han sido creadas.
3. El responsable ordena Adicionar una Acción Centralizada.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
Al oprimir el botón Adicionar el sistema muestra una interfaz donde te permite adicionar la acción centralizada deseada.		El sistema muestra una interfaz	Satisfactorio	
Al oprimir el botón cancelar el sistema cierra la interfaz		No realiza ningún cambio. Cierra la interfaz correspondiente y regresa a la interfaz anterior.	Satisfactorio	
Al oprimir el botón Aceptar de la interfaz el sistema agrega en la lista anterior la nueva o las nuevas Acciones Centralizadas		El sistema agrega la nueva Acción Centralizada.	Satisfactorio	

CPR 2: Eliminar Acción Centralizada.

Flujo Central:

1. El responsable ordena Eliminar la Acción Centralizada seleccionada.
2. El sistema elimina la Acción Centralizada.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El botón Eliminar (-) permanece inhabilitado hasta que se seleccione una Acción Centralizada		Se debe seleccionar primero la acción centralizada antes de oprimir el botón Eliminar (-).	Satisfactorio	
Al oprimir el botón Eliminar (-) el sistema muestra un mensaje de verificación de la operación.		El sistema muestra un mensaje	Satisfactorio	
Al oprimir el botón Cancelar del mensaje, el sistema no realiza ningún cambio.		El sistema no elimina la Acción Centralizada.	Satisfactorio	
Al oprimir el botón Aceptar del mensaje el sistema elimina la Acción Centralizada de la lista anterior.		El sistema elimina la Acción Centralizada	Satisfactorio	

CPR 3: Modificar Acción Centralizada.

Flujo Central:

1. El responsable ordena Modificar la Acción Centralizada.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El botón Modificar (<>) permanece inhabilitado hasta que se seleccione una Acción Centralizada		Se debe seleccionar primero una Acción Centralizada antes de oprimir el botón Modificar (<>).	Satisfactorio	
Al oprimir el botón Modificar(<>) el sistema muestra la interfaz correspondiente para modificar los datos		El sistema muestra la interfaz para modificar una Acción Centralizada	Satisfactorio	
El sistema permite modificar los datos de los campos deseados		El sistema permite modificar los datos de los campos Responsable, Bien o Servicio, Unidad de Medida, Monto total y Meta total, los cuales se encuentran validados	Satisfactorio	
Al oprimir el botón Aceptar el sistema		El sistema actualiza los datos modificados.	Satisfactorio	

actualiza los datos				
Al oprimir el botón Cerrar el sistema no modifica los datos y cierra la interfaz		El sistema cierra la interfaz.	Satisfactorio	

3.3.1.4. CU Gestionar Formulación de un Proyecto

A este Caso de Uso se le realizó la prueba de configurar los datos de un Proyecto.

CPR 1: Configuración de los datos.

Flujo Central:

1. El responsable selecciona el Proyecto que desea modificar.
2. El responsable ordena Modificar.
3. El sistema muestra la interfaz Formulación del Presupuesto por Proyecto con los datos del Proyecto seleccionado.
4. El responsable modifica los datos deseados.
5. El responsable ordena Aceptar.
6. El sistema actualiza el Proyecto con los nuevos datos.
7. El responsable ordena configurar las Acciones Específicas de un Proyecto.
8. El responsable ordena Aceptar la operación.
9. El sistema acepta la operación.
10. El responsable ordena Cerrar la interfaz.
11. El sistema cierra la interfaz, terminando así el Caso de Uso.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario ordena Cancelar		No realiza ningún cambio. Cierra la interfaz correspondiente y regresa a la interfaz anterior	Satisfactorio	
El botón Modificar permanece inhabilitado hasta que se seleccione un Proyecto		El sistema tiene el botón Modificar inhabilitado hasta que se seleccione un Proyecto	Satisfactorio	
Al oprimir el botón Modificar el sistema muestra una interfaz con todos los datos del Proyecto seleccionado.		El sistema muestra la interfaz con los datos de Proyecto	Satisfactorio	

El único campo que se puede modificar es el Monto Anual.		El único campo que se puede modificar es el Monto Anual.	Satisfactorio	

3.3.1.5. CU Gestionar Formulación de una Acción Centralizada

A este Caso de Uso se le realizó la prueba de Adicionar una Acción Centralizada.

CPR 1: Adicionar una Acción Centralizada.

Flujo Central:

1. El responsable ordena Adicionar una Acción Centralizada.
2. El sistema muestra la interfaz Formulación del Presupuesto por Acción Centralizada.
3. El responsable introduce los datos correspondientes.
4. El responsable ordena Aceptar la operación.
5. El sistema cierra la interfaz regresando a la interfaz anterior, terminando así el Caso de Uso.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
Ordena Cancelar		No realiza ningún cambio. Cierra la interfaz correspondiente y regresa a la interfaz anterior.	Satisfactorio	
Al oprimir el botón Adicionar el sistema muestra una interfaz donde te permite adicionar la Acción Centralizada deseada.		El sistema muestra una interfaz	Satisfactorio	
Al oprimir el botón Aceptar el sistema adiciona la Acción Centralizada en el listado anterior		El sistema acepta la operación	Satisfactorio	

3.2.2. Formular Presupuesto en la UEL/UAD

3.2.2.1. CU Formular Presupuesto por Proyecto en la UEL/UAD

A este Caso de Uso se le realizó la prueba de formular presupuesto por proyecto en la UEL/UAD, distribución inter-anual y asignación por fuente de financiamiento.

CPR 1: Formular Presupuesto por Proyecto en la UEL/UAD.

Flujo Central:

1. El Funcionario Administrativo en la UEL solicita la Formulación del Presupuesto por Proyecto.
2. El sistema muestra la interfaz Formulación del Presupuesto por Proyecto con los datos del Proyecto seleccionado y con las siguientes opciones: Acción Específica, Distribución Inter-Anual y Asignación por Fuente de Financiamiento.
3. El Funcionario Administrativo en la UEL selecciona la opción Acción Específica.
4. El sistema muestra una lista de las acciones asociadas a ese Proyecto con los siguientes datos: Código, Denominación, Monto Total y Monto Anual.
5. El Funcionario Administrativo en la UEL selecciona de la lista la Acción Específica que desea modificar.
6. El Funcionario Administrativo en la UEL selecciona la opción Modificar.
7. El sistema manda a ejecutar el caso de uso (Configurar Acción Específica por Proyecto en la UEL/UAD) y hace los cambios pertinentes.
8. El Funcionario Administrativo en la UEL ordena cerrar la interfaz.
9. El sistema cierra la Interfaz.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
Seleccionar un Proyecto y dar clic en el botón "Modificar"		Se muestra la interfaz Formulación del Presupuesto por Proyecto, visualizando los datos correspondientes	Satisfactoria (Todas las funcionalidades se mostraron correctamente)	

CPR 2: Distribución Inter-Anual

Flujo Central:

1. El Funcionario Administrativo en la UEL selecciona la opción Distribución Inter-Anual.
2. El sistema muestra la Distribución Física y Financiera y la Asignación Presupuestaria del Gasto con respecto al año del proyecto.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
Seleccionar opción Distribución		El sistema muestra la Distribución Física y Financiera y la Asignación	Satisfactorio	

Inter-Anual.		Presupuestaria del Gasto con respecto al año del proyecto.		

CPR 3: Asignación por Fuente de Financiamiento

Flujo Central:

1. El Funcionario Administrativo en la UEL selecciona la opción Asignación por Fuente de Financiamiento.
2. El sistema muestra una lista con las Fuentes de Financiamiento asociadas a ese Proyecto.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
Seleccionar opción “Fuente de Financiamiento”		El sistema muestra una lista con las fuentes de financiamiento asociadas a ese proyecto.	Satisfactorio.	

3.2.3. Gestionar Modificaciones Presupuestarias

3.2.3.1. CU Gestionar Modificaciones Presupuestarias

A este Caso de Uso se le realizaron las pruebas de Listar, Eliminar, Modificar, Confirmar y Anular Modificaciones Presupuestarias.

CPR 1: Listar Modificación Presupuestaria.

Flujo Central:

1. El usuario ordena gestionar una Modificación Presupuestaria.
2. El sistema muestra la vista Modificaciones Presupuestarias.
3. El usuario selecciona Adicionar una Modificación.
4. El usuario selecciona los parámetros para hacer una búsqueda y ordena buscar.
5. El sistema muestra una lista de las modificaciones presupuestarias encontradas.
6. El usuario selecciona cerrar.
7. El sistema cierra la vista correspondiente terminando el caso de uso.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones

El usuario ordena cancelar.		No realiza ningún cambio y cierra la vista correspondiente.	Satisfactorio	
El usuario presiona el botón Buscar.		Se muestran todas las Modificaciones Presupuestarias encontradas.	Satisfactorio	

CPR 2: Eliminar Modificación Presupuestaria

Flujo Central:

1. El usuario selecciona de la lista la Modificación Presupuestaria que desea eliminar.
2. El usuario ordena Eliminar.
3. El sistema elimina la Modificación Presupuestaria seleccionada.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario ordena Cancelar.		No realiza ningún cambio y cierra la vista correspondiente.	Satisfactorio	
El usuario presiona el botón <->		El sistema elimina la Modificación Presupuestaria seleccionada.	Satisfactorio	

CPR 3: Modificar Modificaciones Presupuestarias

Flujo Central:

1. El usuario selecciona de la lista la Modificación Presupuestaria que desea modificar.
2. El usuario ordena Modificar.
3. El sistema muestra la ventana correspondiente al Tipo de Modificación seleccionada.
4. El usuario cambia los datos que desea modificar y ordena Aceptar.
5. El sistema actualiza la Modificación Presupuestaria con los nuevos datos y cierra la vista correspondiente.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario ordena cancelar.		No realiza ningún cambio y cierra la vista correspondiente.	Satisfactorio	

El usuario presiona el botón Modificar		El sistema permite modificar el trámite y muestra la vista Modificación Presupuestaria.	Satisfactorio	

CPR 4: Confirmar Modificaciones Presupuestarias

Flujo Central:

1. El usuario selecciona de la lista aquella modificación que desea confirmar.
2. El usuario ordena Confirmar.
3. El sistema cambia el Estatus de la Modificación Presupuestaria a Confirmada y modifica los montos de las partidas o los rubros del Presupuesto.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario ordena cancelar.		No realiza ningún cambio y cierra la vista correspondiente.	Satisfactorio	
El usuario presiona el botón Confirmar		El sistema muestra un mensaje preguntando si está seguro que desea confirmar la modificación y se confirma la Modificación	Satisfactorio	

CPR 5: Anular Modificaciones Presupuestarias

Flujo Central:

1. El usuario selecciona de la lista aquella modificación que desea anular.
2. El usuario ordena Anular.
3. El sistema cambia el Estatus de la Modificación Presupuestaria a Anulada e invierte el proceso hecho anteriormente al confirmar la modificación presupuestaria, es decir se restablecen los montos.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario ordena cancelar.		No realiza ningún cambio y cierra la vista correspondiente.	Satisfactorio	

El usuario presiona el botón Anular		El sistema muestra un mensaje preguntando si está seguro que desea anular la modificación y se anula la Modificación	Satisfactorio	

3.2.3.2. CU Crear Modificación Presupuestaria según tipo de modificación

A este Caso de Uso se le realizaron las pruebas de crear Modificaciones Presupuestarias según el tipo de éstas (Traspasos de Créditos Presupuestarios, Créditos Adicionales, Insubsistencia de Créditos Presupuestarios, Recorte de Créditos presupuestarios y Rectificaciones Presupuestarias).

CPR 1: Seleccionar Traspasos de Créditos Presupuestarios

Flujo Central:

1. El usuario selecciona el tipo de Modificación Presupuestaria “Traspasos de Créditos Presupuestarios”.
2. El sistema muestra la vista “Traspasos de Créditos Presupuestarios”.
3. El usuario selecciona la Declaración de Insubsistencia por la cual va a realizar la Modificación y entra los datos Número, Descripción, Fundamentación y Fecha.
4. El sistema muestra las Partidas Cedentes correspondientes a la Declaración de Insubsistencia seleccionada.
5. El usuario introduce los datos Descripción, Fundamentación y Fecha.
6. El usuario introduce las Partidas Receptoras y ordena Aceptar.
7. El usuario valida que los montos de las Partidas Cedentes y las Partidas Receptoras sean iguales y registra la Modificación Presupuestaria.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario ordena cancelar.		No realiza ningún cambio y cierra la vista correspondiente.	Satisfactorio	
El usuario presiona el botón Aceptar para crear una Modificación Presupuestaria		La Modificación Presupuestaria se crea y se muestra en la vista Modificación Presupuestaria	Satisfactorio	
El sistema muestra un mensaje de diferencias en los montos.		El usuario acepta el mensaje y vuelve a introducir las partidas.	Satisfactorio	

CPR 2: Seleccionar Créditos Adicionales

Flujo Central:

1. El usuario selecciona el tipo de Modificación Presupuestaria “Créditos Adicionales”.
2. El sistema muestra la vista “Créditos Adicionales”.
3. El usuario introduce los datos Número, Descripción, Fundamentación y Fecha.
4. El usuario introduce las partidas correspondientes a ese tipo de modificación y ordena Aceptar.
5. El sistema valida que los montos de partidas y rubros sean iguales y registra la Modificación Presupuestaria.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario ordena cancelar.		No realiza ningún cambio y cierra la vista correspondiente.	Satisfactorio	
El usuario presiona el botón Aceptar para crear una Modificación Presupuestaria		La Modificación Presupuestaria se crea y se muestra en la vista Modificación Presupuestaria	Satisfactorio	
El sistema muestra un mensaje de diferencias en los montos.		El usuario acepta el mensaje y vuelve a introducir las partidas.	Satisfactorio	

CPR 3: Seleccionar Insubsistencia de Créditos Presupuestarios

Flujo Central:

1. El usuario selecciona el tipo de Modificación Presupuestaria “Insubsistencia de Créditos Presupuestarios”.
2. El sistema muestra la vista “Insubsistencia de Créditos Presupuestarios”.
3. El usuario introduce los datos Número, Descripción, Fundamentación y Fecha.
4. El usuario introduce las partidas convenientes a ese tipo de modificación y ordena Aceptar.
5. El sistema valida que el Monto Inicial de la partida sea mayor que el Monto Final y registra la Modificación Presupuestaria.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones

El usuario ordena cancelar.		No realiza ningún cambio y cierra la vista correspondiente.	Satisfactorio	
El usuario presiona el botón Aceptar para crear una Modificación Presupuestaria		La Modificación Presupuestaria se crea y se muestra en la vista Modificación Presupuestaria	Satisfactorio	
El sistema muestra un mensaje de diferencias en los montos.		El usuario acepta el mensaje y vuelve a introducir las partidas.	Satisfactorio	

CPR 4: Recorte de Créditos presupuestarios

Flujo Central:

1. El usuario selecciona el tipo de Modificación Presupuestaria "Recorte de Créditos Presupuestarios".
2. El sistema muestra la vista "Recorte de Créditos Presupuestarios".
3. El usuario introduce los datos Número, Descripción, Fundamentación y Fecha.
4. El usuario introduce las partidas convenientes a ese tipo de modificación y ordena Aceptar.
5. El sistema valida que el Monto Inicial de la partida sea mayor que el Monto Final y registra la Modificación Presupuestaria.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario ordena cancelar.		No realiza ningún cambio y cierra la vista correspondiente.	Satisfactorio	
El usuario presiona el botón Aceptar para crear una Modificación Presupuestaria		La Modificación Presupuestaria se crea y se muestra en la vista Modificación Presupuestaria	Satisfactorio	
El sistema muestra un mensaje de diferencias en los montos.		El usuario acepta el mensaje y vuelve a introducir las partidas.	Satisfactorio	

CPR 5: Rectificaciones Presupuestarias

Flujo Central:

1. El usuario selecciona el tipo de Modificación Presupuestaria “Rectificaciones Presupuestarias”.
2. El sistema muestra la vista “Rectificaciones Presupuestarias”.
3. El usuario introduce los datos Número, Descripción, Fundamentación y Fecha.
4. El usuario introduce las Partidas Receptoras y Cedentes y ordena Aceptar.
5. El sistema valida que los montos de las Partidas Cedentes y las Partidas Receptoras sean iguales y registra la Modificación Presupuestaria.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario ordena cancelar.		No realiza ningún cambio y cierra la vista correspondiente.	Satisfactorio	
El usuario presiona el botón Aceptar para crear una Modificación Presupuestaria		La Modificación Presupuestaria se crea y se muestra en la vista Modificación Presupuestaria	Satisfactorio	
El sistema muestra un mensaje de diferencias en los montos.		El usuario acepta el mensaje y vuelve a introducir las partidas.	Satisfactorio	

3.2.4. Programación Financiera.

3.2.4.1. CU Programar la Ejecución Financiera por Trimestre

A este Caso de Uso se le realizaron las pruebas de Programar, Eliminar, Modificar y Aprobar la Ejecución Financiera por Trimestres.

CPR 1: Programar la Ejecución Financiera por Trimestre.

Flujo Central:

1. El Funcionario de Planificación y Presupuesto ordena programar la ejecución financiera por trimestre.
2. El sistema muestra la vista “Programación Trimestral de la Ejecución Financiera”.
3. El Funcionario selecciona el año presupuestario al cual se le va hacer la programación trimestral de la Ejecución Financiera.
4. El Funcionario ordena Buscar.
5. El sistema muestra los datos de los trimestres asociados a ese año presupuestario.

6. El Funcionario selecciona el trimestre al cual le va hacer la programación trimestral por Proyectos y Acciones Centralizadas.
7. El Funcionario ordena Adicionar metas.
8. El sistema muestra la vista Programación Trimestral de la Ejecución Financiera del Trimestre X.
9. El Funcionario introduce los datos de las metas asociadas a ese trimestre y ordena Aceptar.
10. El sistema registra las metas correspondientes a ese trimestre y cierra la vista correspondiente.
11. El Funcionario ordena cerrar.
12. El sistema cierra la vista correspondiente terminando el caso de uso.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario selecciona el trimestre y ordena adicionar metas(se introducen todos los datos correctamente)		El sistema registra las metas correspondientes a ese trimestre y cierra la vista correspondiente.	Satisfactorio	
	El usuario selecciona el trimestre y ordena adicionar metas(no se introducen todos los datos correctamente)	Se espera que el sistema indique que faltan datos por introducir	Satisfactorio	

CPR 2: Eliminar la Ejecución Financiera por Trimestre

Flujo Central:

1. El Funcionario selecciona la opción Eliminar.
2. El sistema valida que el trimestre no tenga el estatus de Confirmado.
3. El sistema elimina las metas del trimestre seleccionado

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
Selecciona una meta correspondiente a un trimestre que no está Confirmado y da clic en el botón Eliminar.		El sistema elimina la meta seleccionada	Satisfactorio	

	1.Selecciona una meta correspondiente a un trimestre que está Confirmado (trimestre III)	El sistema deshabilita el botón eliminar (--) para que esta funcionalidad no sea solicitada por el usuario	Satisfactorio	

CPR 3: Modificar la Ejecución Financiera por Trimestre

Flujo Central:

1. El Funcionario de Planificación y Presupuesto selecciona la opción Modificar.
2. El sistema muestra la vista “Programación Trimestral de la Ejecución Financiera del Trimestre X.
3. El Funcionario modifica los datos que desea cambiar de las metas y ordena Aceptar.
4. El sistema valida que el trimestre no tenga el estatus de Aprobado.
5. El sistema modifica las metas del trimestre seleccionado.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
Selecciona una meta correspondiente a un trimestre que no está Confirmado y da clic en el botón Modificar.		El sistema modifica las metas del trimestre seleccionado.	Satisfactorio	
	Selección: una meta correspondiente a un trimestre que está Confirmado	El sistema deshabilita el botón Modificar para que esta funcionalidad no sea solicitada por el usuario	Satisfactorio	

CPR 4: Aprobar la Ejecución Financiera por Trimestre

Flujo Central:

1. El Funcionario de Planificación y Presupuesto selecciona la opción Aprobar.
2. El sistema aprueba la programación trimestral de la ejecución financiera de ese trimestre cambiándole el estatus del trimestre a Aprobado.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones

Selecciona un trimestre que no está Confirmado y da clic en el botón Aprobar		El sistema aprueba la programación de ese trimestre cambiándole el estatus del trimestre a Aprobado.	Satisfactorio	
	Selecciona un trimestre que ya está Confirmado e intenta dar clic en el botón Aprobar	El sistema deshabilita el botón Aprobar	Satisfactorio	

3.3. Pruebas de Caja Blanca

La Prueba de Caja Blanca es aquella que se lleva a cabo sobre el código del programa permitiendo examinar la estructura interna de éste. Es un método para diseñar casos de prueba que utiliza la estructura de control del diseño procedimental para obtener casos de prueba que:

1. Garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo del software.
2. Garanticen que se ejerciten todas las decisiones lógicas en sus dos formas: Verdadero o Falso.
3. Garanticen que se ejecuten todos los bucles en sus límites y con sus límites operacionales.
4. Garanticen que se ejerciten las estructuras internas de datos para asegurar su validez.

Normalmente son conocidas como “Pruebas de Cobertura” o de “Caja de Cristal”. Al total de Pruebas de Caja Blanca se le llama “Cobertura” la cual representa un número porcentual que indica cuánto código del programa se ha probado. Las Pruebas de Cobertura consisten en diseñar un plan de pruebas en las que se vaya ejecutando sistemáticamente el código hasta que haya corrido todo o la gran mayoría de él. Deben ser realizadas cuando el software haya sido terminado y no deben ser confundidas con las pruebas que realiza el programador mientras desarrolla su código ya que estas nunca son eficaces por no tener un adecuado diseño.
(Pressman, 2002)

3.3.1. Prueba del Camino Básico

La Prueba del Camino Básico es una técnica de Prueba de Caja Blanca que 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 de caminos de ejecución. Estos casos de prueba son los que garantizan que se ejecute al menos una vez cada sentencia del programa.

Para ilustrar esta técnica se hace uso de los denominados "*grafos de flujo*", aunque no es necesario, constituyen una herramienta muy útil sobre todo cuando la lógica de la estructura de control de un módulo es muy compleja permitiendo trazar los caminos del programa de una forma más fácil.

Los elementos de un grafo de flujo son los siguientes:

Nodos: Representan una o más sentencias procedimentales. Se denominan "nodos predicados" aquellos que contienen una condición (if - else) y se caracterizan porque dos o más aristas surgen de él.

Aristas: Representan el flujo de control. Deben terminar en un nodo aunque el nodo no constituya ninguna sentencia procedimental.

Regiones: Son las áreas delimitadas por aristas y nodos. El área exterior del grafo se cuenta como una región más.

En la Figura 3.1 se muestra un grafo de flujo y sus elementos.

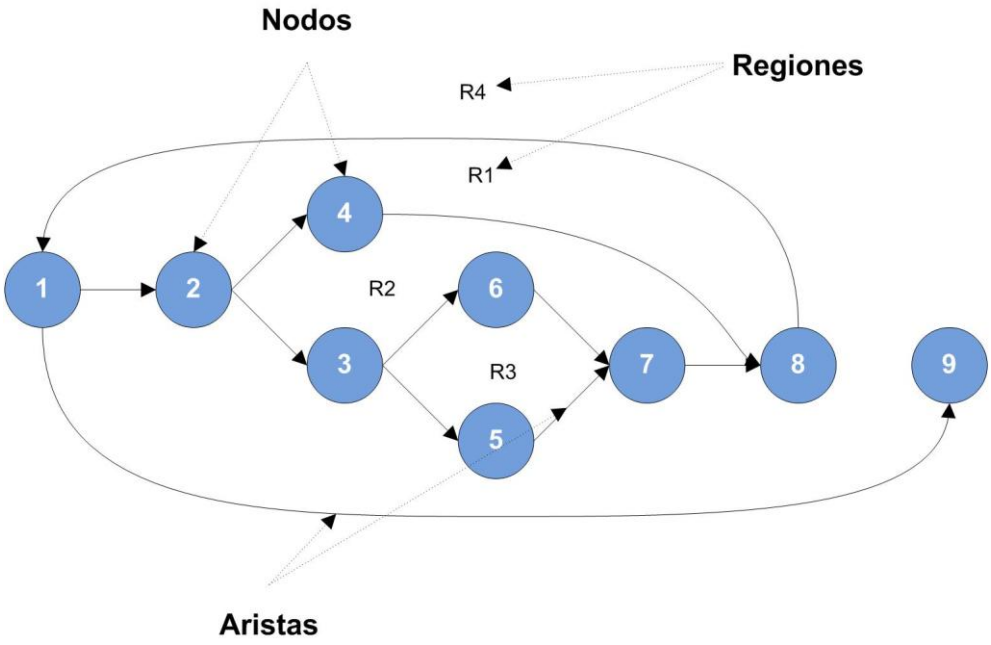


Figura 3.1. Grafo de Flujo con sus elementos

(Pressman, 2002)

3.3.2. Métrica de la Complejidad Ciclomática

La Complejidad Ciclomática es una métrica que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cuando se usa en el contexto de la técnica del Camino Básico, este valor calculado como complejidad ciclomática indica el número de "caminos independientes[21]" del conjunto básico de un programa y da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuten al menos una vez cada sentencia.

Esta métrica es útil para predecir los módulos que son más propensos a error y puede ser usada para planificar pruebas así como para diseñar casos de prueba.

La complejidad se puede calcular de tres formas diferentes:

1. El número de regiones del grafo de flujo es igual a la complejidad ciclomática.
2. Se define a la complejidad ciclomática V(G), de un grafo G, como:

   V(G) = A – N + 2

   donde A es el número de aristas y N el número de nodos del grafo de flujo.
3. También se define a la complejidad ciclomática V(G), de un grafo G, como:

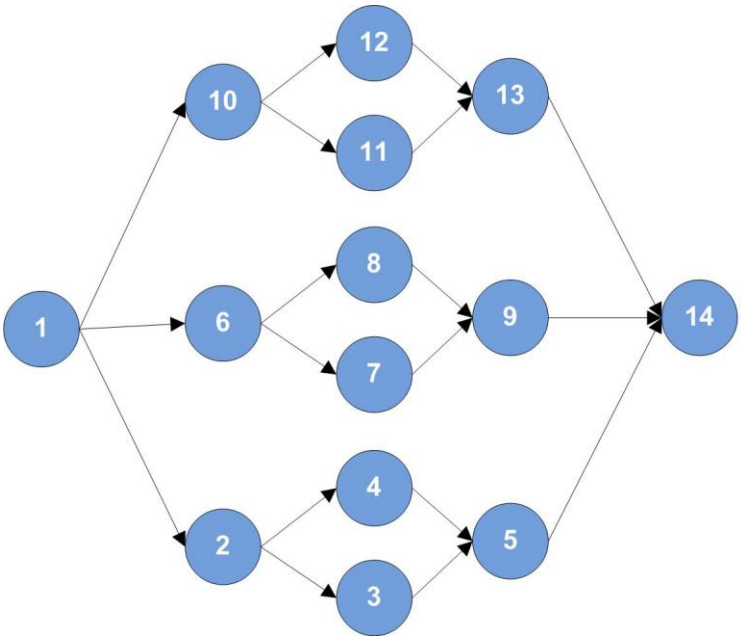
   V(G) = P + 1

   donde P es el número de nodos predicado del grafo de flujo.

Se realizó el cálculo de la complejidad ciclomática en todos los bloques de código dentro del sistema lo que ayudó a reflejar una medida de la complejidad del código escrito. A continuación se muestran algunos fragmentos de código junto con sus grafos de flujo correspondientes y el cálculo de la complejidad ciclomática. Las imágenes que contienen al código fueron necesarias ponerlas como anexos al documento por su gran extensión.

1. Este grafo representa a un método que muestra las listas de Proyectos y Acciones Centralizadas pertenecientes al Anteproyecto de Presupuesto de un año en específico en las UEL/UAD. El código correspondiente se muestra en el Anexo 5

[21] Cualquier camino del programa que introduce al menos un nuevo conjunto de sentencias o una nueva condición. En términos de grafo de flujo, está constituido por lo menos por una arista que no haya sido recorrida anteriormente a la definición del camino.

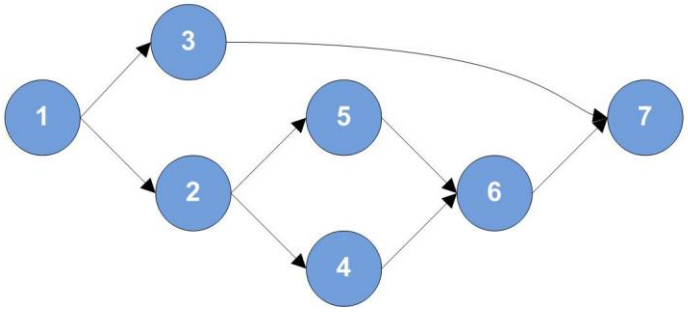


V(G) = A – N + 2
V(G) = 18 – 14 + 2
V(G) =6

Caminos Independientes

- 1-2-3-5-14
- 1-2-4-5-14
- 1-6-7-9-14
- 1-6-8-9-14
- 1-10-11-13-14
- 1-10-12-13-14

2. Este grafo representa al evento que se lanza al oprimir el botón “Confirmar” para confirmar una Modificación Presupuestaria. El código correspondiente se muestra en el Anexo 6

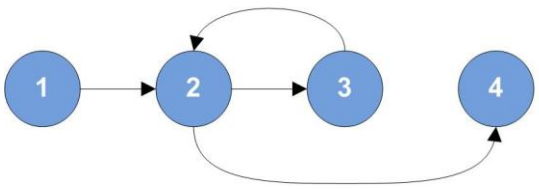


V(G) = A – N + 2
V(G) = 8 – 7 + 2
V(G) = 3

Caminos Independientes

- 1-2-4-6-7
- 1-2-5-6-7
- 1-3-7

3. Este grafo representa a un método que obtiene las Acciones Centralizadas de la base de datos. El código correspondiente se muestra en el Anexo 7

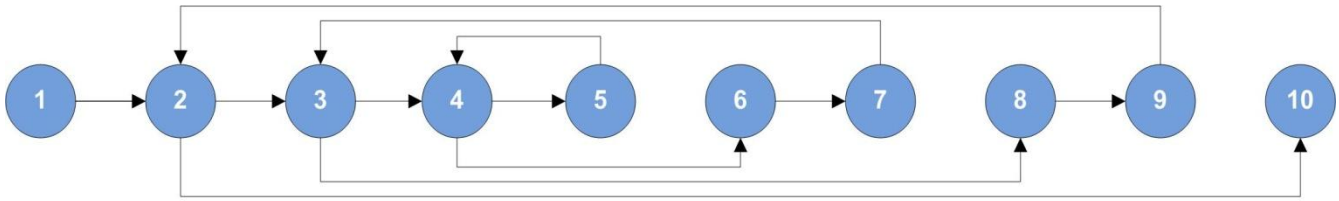


V(G) = A – N + 2
V(G) = 4 – 4 + 2
V(G) = 2

Caminos Independientes

- 1-2-3-2
- 1-2-4

4. Este grafo representa a un método de una clase gestora de la Capa de Negocio que salva las Acciones Centralizadas en la base de datos. El código correspondiente se muestra en el Anexo 8



V(G) = A – N + 2
V(G) = 12 – 10 + 2
V(G) = 4

Caminos Independientes

- 1-2-10
- 1-2-3-8-9-2-10
- 1-2-3-4-6-7-3-8-9-2-10
- 1-2-3-4-5-4-6-7-3-8-9-2-10

Se han medido un gran número de programas a modo de establecer rangos de complejidad que ayuden al Ingeniero de Software a determinar la estabilidad y riesgo de un programa. La medida resultante puede ser utilizada en el desarrollo, mantenimiento y reingeniería para estimar el riesgo, costo y estabilidad del sistema. Algunos estudios experimentales indican la existencia de distintas relaciones entre esta métrica y el número de errores existentes en el código fuente, así como el tiempo requerido para encontrar y corregir esos errores. Se suele comparar la Complejidad Ciclomática obtenida contra un conjunto de valores límites como se observa en la siguiente tabla:

Complejidad Ciclomática	Evaluación del Riesgo
1 - 10	Programa Simple sin mucho riesgo
11 – 20	Más complejo, riesgo moderado
21 – 50	Complejo, programa de alto riesgo
50 +	Programa no testeable, muy alto riesgo

Tabla 3.1. Complejidad Ciclomática vs Evaluación del Riesgo

(Pressman, 2002) (Rizzi)

El cálculo de complejidad más alto que presenta esta propuesta de solución es de 36 el cual se encuentra en el rango de 21 – 50 por lo que se puede llegar a la conclusión de que se cuenta con un programa complejo y de alto riesgo, no obstante, se demostró que las funciones del software son operativas, que las entradas se aceptan de forma adecuada y que se produce una salida correcta, así como que se mantiene la integridad de la información externa.

Conclusiones

En consecuencia al trabajo realizado se ejercitaron completamente los requisitos funcionales del sistema, demostrándose que las funciones del software son totalmente operativas. En efecto, fueron

derivadas un conjunto de reglas en soporte a las técnicas aplicadas de acuerdo a los métodos establecidos, encaminadas a obtener los casos de pruebas asociados a los casos de usos más significativos para la arquitectura. En otros casos se procedió al estudio y análisis de la complejidad lógica del sistema a través de herramientas con las que se arribaron a resultados concretos.

CONCLUSIONES

Con el desarrollo de este trabajo, se arribaron a las siguientes conclusiones:

- A partir de los casos de uso del sistema, los requisitos asociados a éstos, las reglas del negocio y la aplicación de patrones de diseño y arquitectura, se construyó el Modelo de Diseño.
- Con la elaboración del Modelo de Diseño para el módulo Presupuesto, se obtuvieron los Diagramas de Clases, de Secuencia, el Modelo de Datos y el de Despliegue, constituyendo un punto de partida para la realización del Modelo de Implementación.
- Con la aplicación de las Pruebas de Caja Negra se ejercitaron completamente los requisitos funcionales del sistema. Mediante las Pruebas de Caja Blanca se garantizó que se ejecutaran al menos una vez todas las sentencias del programa proporcionando una medida de la complejidad lógica del sistema. Ambas pruebas demostraron que el software funciona correctamente con una alta calidad.
- La implantación de este módulo permitirá a la Dirección Nacional de Registros y del Notariado ejercer autonomía en el país mediante un sistema informático donde se llevará a cabo un control estricto y centralizado de los fondos destinados a cada Registro y Notaría.

RECOMENDACIONES

Con la realización del presente trabajo se recomienda:

- Migrar la solución hacia alguna plataforma de Software Libre, preferentemente al Proyecto Mono, lo que permitirá la reutilización de todo el código desarrollado.
- Integrar el módulo “Presupuesto” a los restantes módulos del Sistema de Administración Financiera.
- Realizar Pruebas de Unidad al sistema como parte de las Pruebas de Caja Blanca para garantizar que el mismo cuente con una alta calidad. También es recomendable que se le hagan otros tipos de pruebas, por ejemplo las de Integración, para comprobar si funciona bien con los diferentes módulos del Sistema de Administración Financiera.

BIBLIOGRAFÍA

Ávila, Rodolfo Pérez. 2007. Análisis, Diseño e Implementación de los Servicios en Línea del Módulo Mercantil del Proyecto de Informatización de los Registros y Notarías. [Trabajo de Diploma para optar por el título de Ingeniero Informático]. Caracas, Venezuela : s.n., Mayo de 2007.

Campeche, Instituto Tecnológico Superior de Calkiní en el Estado de. 2007. ITESCAM. [En línea] 2007. [Citado el: 28 de Noviembre de 2007.] http://www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r26950.DOC.

Charte Ojeda, Francisco. 2002. *Visual C# .Net.* Madrid : Anaya Multimedia, 2002.

Córdova, Johanna. 2007. Presupuesto. [En línea] 2007. [Citado el: 27 de Noviembre de 2007.] http://www.monografias.com/trabajos21/presupuesto/presupuesto.shtml.

Cuevas Guerrero, Daniel. Universidad de Málaga. *Departamento de Lenguajes y Ciencias de la Computación.* [En línea] [Citado el: 7 de Diciembre de 2007.] http://www.lcc.uma.es/~pastrana/EP/trabajos/45.pdf.

Desoft.SA. 2007. Desoft, excelencia en software. [En línea] 2007. [Citado el: 8 de Diciembre de 2007.] http://cursosonline.hab.desoft.cu/file.php/1/Biblioteca_Virtual/Contabilidad/Presupuesto.pdf..

Dodero, Juan Manuel y Fernández Llamas, Camino. 2002-2003. Universidad Carlos III de Madrid. *Laboratorio DEI.* [En línea] 2002-2003. [Citado el: 5 de Marzo de 2008.] http://peterpan.uc3m.es/docencia/p_s_ciclo/tdp/curso0203/apuntes/factory.pdf.

Ferguson, Jeff, Patterson, Brian y Beres, Jason. 2003. *La Biblia de C#.* Madrid : Anaya Multimedia, 2003.

Fonteboa Vizcaíno, Antonio. 2006. Objetivos del Presupuesto. [En línea] 10 de Mayo de 2006. [Citado el: 27 de Noviembre de 2007.] http://www.wikilearning.com/objetivos_del_presupuesto-wkccp-12626-3.htm.

Gamma, Erich, y otros. 1998. *Design Patterns CD.* s.l., Massachusetts : Addison Wesley Longman Inc., 1998. Design Patterns:Elements of Reusable Object-Oriented Software.

Gobierno Bolivariano de Venezuela. 2007. Oficina Nacional de Presupuesto(ONAPRE). [En línea] 2007. [Citado el: 27 de Noviembre de 2007.] http://www.ocepre.gov.ve/conceptos/conceptos.html.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2004. *El Proceso Unificado de Desarrollo de Software.* Ciudad de la Habana : Félix Varela, 2004. Vol. 1.

Lanzarini, Laura. Instituto de Investigación en Informática LIDI. *Facultad de Informática.* [En línea] [Citado el: 21 de Febrero de 2008.] http://weblidi.info.unlp.edu.ar/catedras/seminariob/Introduccion.doc.

Martínez, Gerardo Moreno. Ingeniería de Software UML. [En línea] [Citado el: 4 de Diciembre de 2007.] http://www.monografias.com/trabajos5/insof/insof.shtml.

Ministerio de Finanzas, República Bolivariana de Venezuela. 2002. Sistema Integrado de Gestión y Control de las Finanzas Públicas. Entes Descentralizados sin fines empresariales. Marco conceptual. Caracas, Venezuela : s.n., Diciembre de 2002.

—. 1999. Sistema Integrado de Gestión y Control de las Finanzas Públicas. Manual de Modificaciones Presupuestarias y Reprogramación de la Ejecución del Presupuesto de Gastos. Julio de 1999.

Molina Díaz, Yasmany. 2007. Diseño e Implementación de la solución informática para la Gestión Documental de los Registros Públicos de Venezuela. [Trabajo de Diploma para optar por el título de Ingeniero Informático]. Caracas, Venezuela : s.n., Mayo de 2007.

Navarro, Juan José Moreno y Collado, M. Universidad Politécnica de Madrid. *Facultad de Informática, Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software.* [En línea] [Citado el: 22 de Febrero de 2008.] http://lml.ls.fi.upm.es/~jjmoreno/sbc/intro_pbc.ppt.

Oracle. 2006. Oracle.com. [En línea] Abril de 2006. [Citado el: 4 de Marzo de 2008.] http://www.oracle.com/technology/products/database/oracle10g/pdf/DS_General_Oracle_Database10g R2_EE_0605.pdf.

Orellana, Marco. 2006. Universidad del Azuay. *Facultad de Ciencias de la Administración, Ingeniería de Sistemas.* [En línea] Enero de 2006. [Citado el: 21 de Febrero de 2008.] http://uazuay.edu.ec/estudios/sistemas/eventos/cuaderno_docente.doc.

Pressman, Roger S. 2002. *Ingeniería de Software, un enfoque práctico.* Quinta Edición. s.l. : Mc Graw Hill, 2002.

Quintana, Darío. 2007. www.darioquintana.com.ar. *Tutorial del NHibernate- Primeros Pasos.* [En línea] 4 de Junio de 2007. [Citado el: 27 de Febrero de 2008.] http://darioquintana.com.ar/articles/tutorial-de-nhibernate-primeros-pasos.

Rational Software Corporation. 2003. *Rational Unified Process.* 2003. Vol. 2003.06.00.65.

Rizzi, Francisco Marcelo. Instituto Tecnológico de Buenos Aires (ITBA). [En línea] [Citado el: 8 de Mayo de 2008.] http://www.itba.edu.ar/capis/rtis/articulosdeloscuadernosetapaprevia/RIZZI-COMPLEJIDAD.pdf.

Sparx Systems. 2007. Sparx Systems. *Enterprise Architect - Herramienta de diseño UML.* [En línea] 2007. [Citado el: 10 de Marzo de 2008.] http://www.sparxsystems.com.ar/products/ea.html.

SYNSPACE AG. 2005. SYNSPACE AG. [En línea] 17 de Agosto de 2005. [Citado el: 28 de Marzo de 2008.] http://www.synspace.com/ES/Services/tcc.html.

2007. Taringa! *Introducción a NHibernate.* [En línea] 31 de Marzo de 2007. [Citado el: 27 de Febrero de 2008.] http://www.taringa.net/posts/offtopic/131352/nHibernate.html.

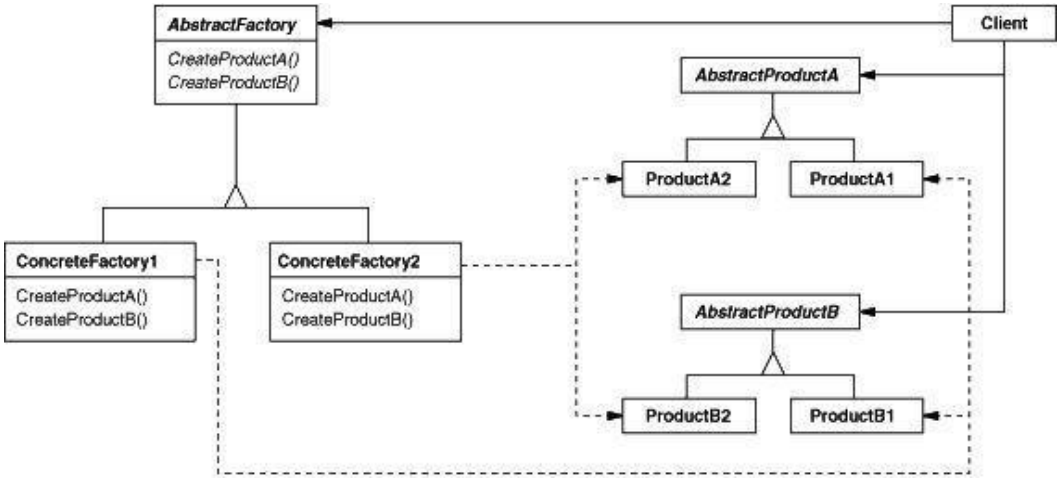
Teso, Leandro del. El Paradigma orientado a objetos. [En línea] [Citado el: 5 de Diciembre de 2007.] http://www.monografias.com/trabajos14/paradigma/paradigma.shtml.

Universidad de Burgos. 1999. Introducción a la Programacion Orientada a Objetos. [En línea] Octubre de 1999. [Citado el: 5 de Diciembre de 2007.] http://pisuerga.inf.ubu.es/lsi/Invest/Java/Tuto/I_1.htm.

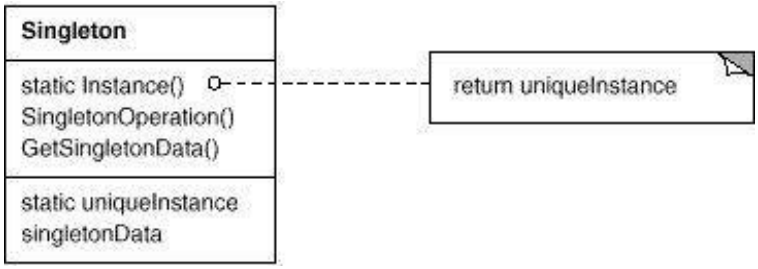
Universidad de Jaén. 2008. Departamanto de Informática. [En línea] 15 de Febrero de 2008. [Citado el: 22 de Febrero de 2008.] http://wwwdi.ujaen.es/asignaturas/progav/progav-tema4.pdf.

ANEXOS

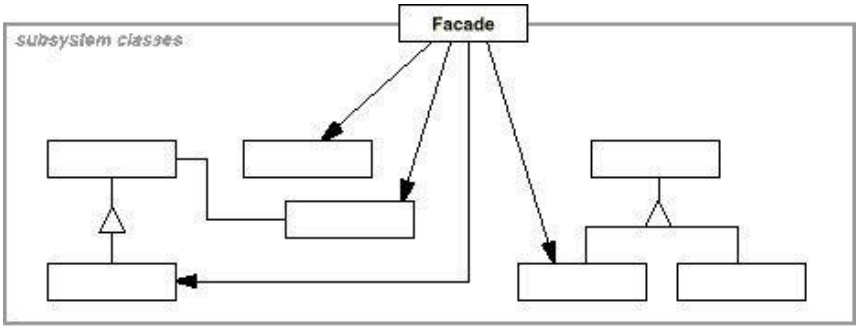
Anexo 1: Estructura del Patrón Abstract Factory (Fábrica Abstracta).



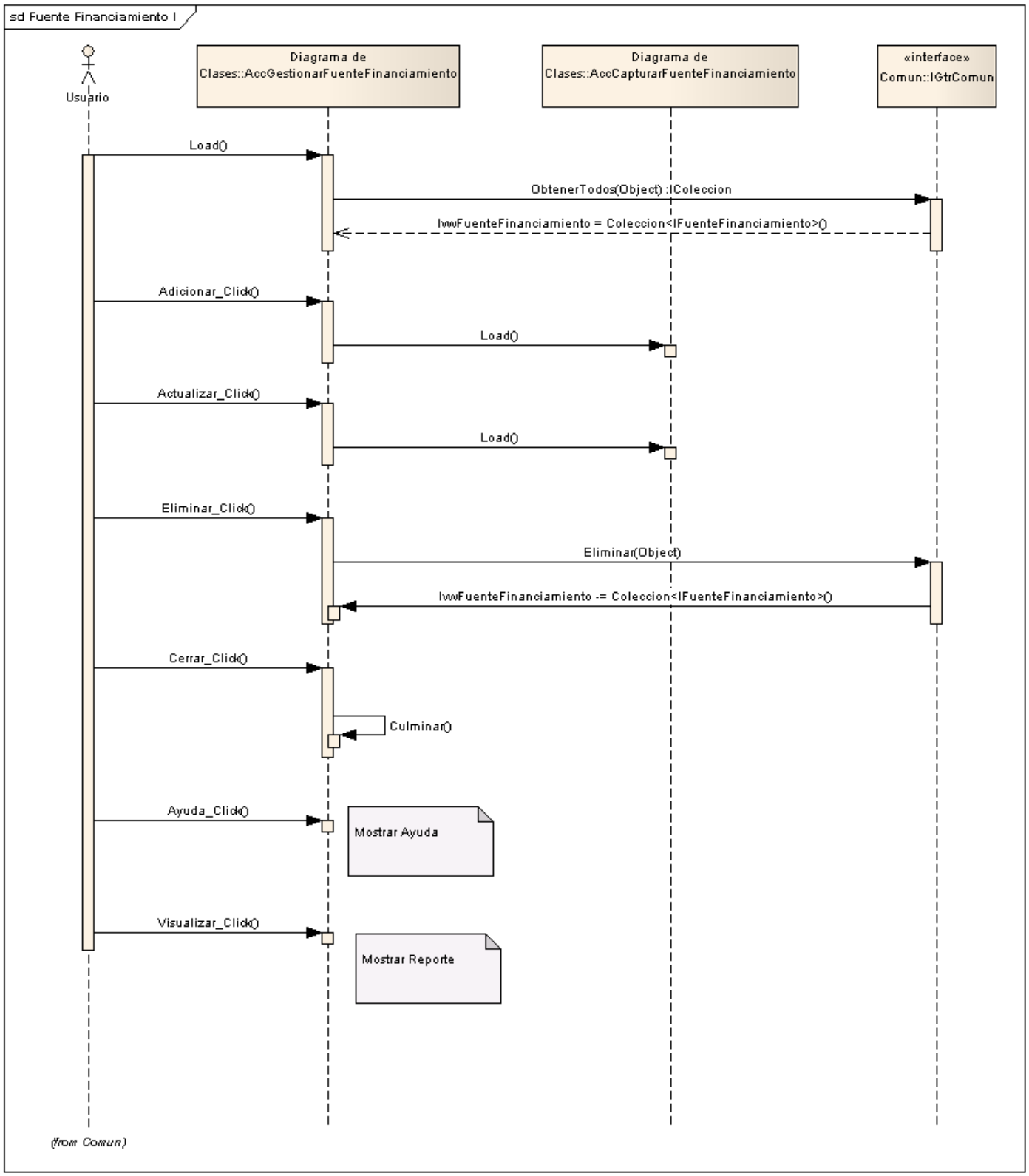
Anexo 2: Estructura del Patrón Singleton.

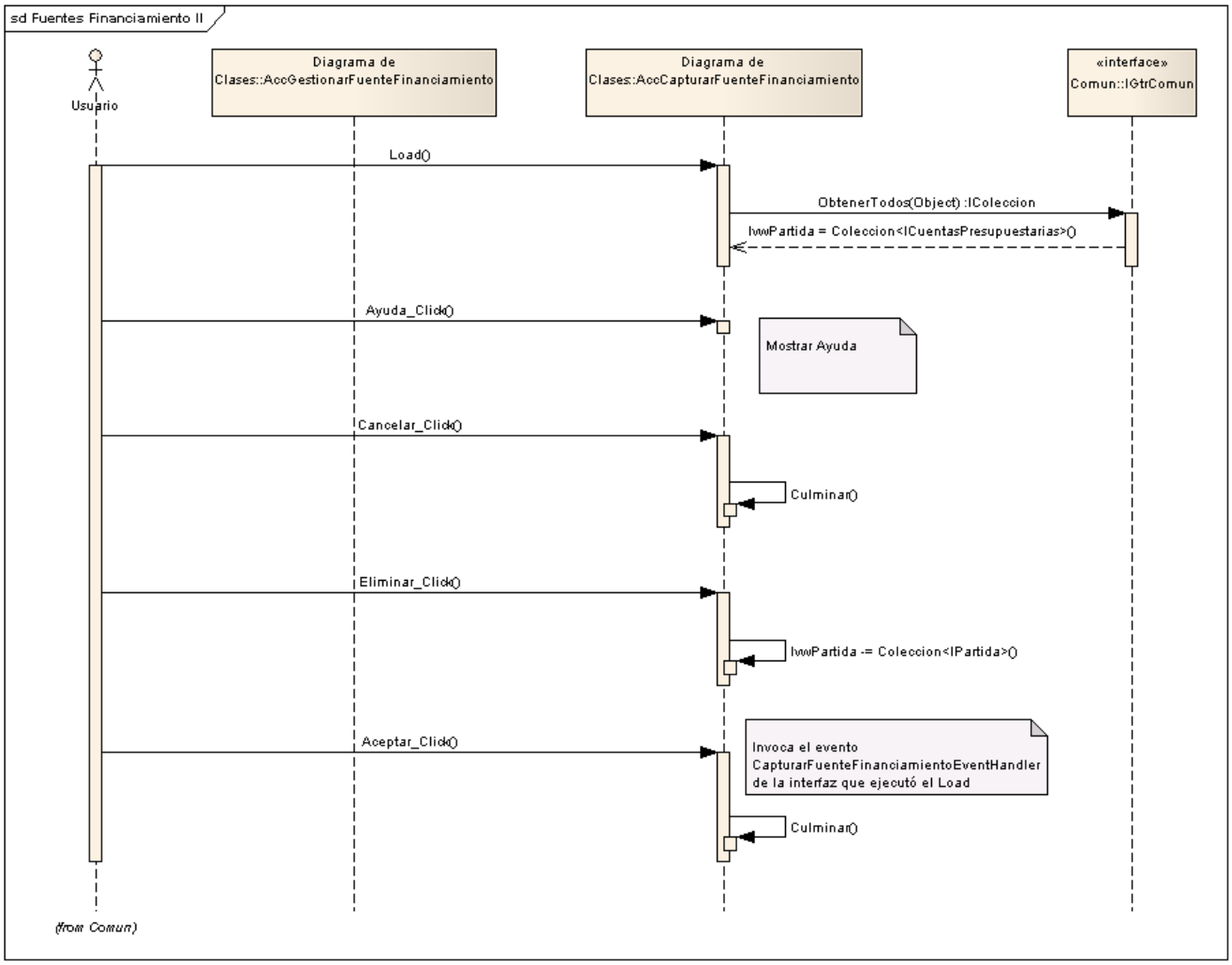


Anexo 3: Estructura del Patrón Facade.



Anexo 4: Diagrama de Secuencia


sd Fuente Financiamiento I
Usuario
Diagrama de Clases::AccGestionarFuenteFinanciamiento
Diagrama de Clases::AccCapturarFuenteFinanciamiento
«interface» Comun::IGtrComun
Load()
ObtenerTodos(Object) :IColeccion
IwwFuenteFinanciamiento = Coleccion<IFuenteFinanciamiento>()
Adicionar_Click()
Load()
Actualizar_Click()
Load()
Eliminar_Click()
Eliminar(Object)
IwwFuenteFinanciamiento -= Coleccion<IFuenteFinanciamiento>()
Cerrar_Click()
Culminar()
Ayuda_Click()
Mostrar Ayuda
Visualizar_Click()
Mostrar Reporte
(from Comun)


sd Fuentes Financiamiento II
Usuario
Diagrama de Clases::AccGestionarFuenteFinanciamiento
Diagrama de Clases::AccCapturarFuenteFinanciamiento
«interface» Comun::IGtrComun
Load()
ObtenerTodos(Object) :IColeccion
lvwPartida = Coleccion<ICuentasPresupuestarias>()
Ayuda_Click()
Mostrar Ayuda
Cancelar_Click()
Culminar()
Eliminar_Click()
lvwPartida -= Coleccion<IPartida>()
Aceptar_Click()
Invoca el evento CapturarFuenteFinanciamientoEventHandler de la interfaz que ejecutó el Load
Culminar()
(from Comun)

Anexo 5: Fragmento de código del método "MostrarListasAnteProyecto(int param)"

```
private void MostrarListasAnteProyecto(int indiceTabControl) 7
{
    switch(indiceTabControl)
    {
        case 0:
            if(Convert.ToInt64(objforma.nmAnteProyecto.IdSeleccionado) != 0)
                objforma.lvwBienesS.Objetos = Fabrica.GetInstancia.CrearGestorPresupuestoUAD().
                    ObtenerBienesServiciosAP(Convert.ToInt64(objforma.nmAnteProyecto.IdSeleccionado));
            break;
        case 1:
            if (Convert.ToInt64(objforma.nmAnteProyecto.IdSeleccionado) != 0)
                objforma.lvwProyectos.Objetos = Fabrica.GetInstancia.CrearGestorPresupuestoUAD().
                    ObtenerProyectoAP(Convert.ToInt64(objforma.nmAnteProyecto.IdSeleccionado));
            break;
        case 2:
            if (Convert.ToInt64(objforma.nmAnteProyecto.IdSeleccionado) != 0)
                objforma.lvwAccionesCent.Objetos = Fabrica.GetInstancia.CrearGestorPresupuestoUAD().
                    ObtenerAccionCentralizadaAP(Convert.ToInt64(objforma.nmAnteProyecto.IdSeleccionado));
            break;
    }
}
```

Anexo 6: Fragmento de código del evento "btnConfirmar_CClick".

```
private void btnConfirmar_CClick(object sender)
{
    objModificacionPresupuesto.ListaModificacionPartidas = new ArrayList();
    if(forma.lvwModificacionesP.SelectedItems.Count > 0)
    {
        if(Mensajeria.Instancia.MostrarMensaje(40))
        {
            MostrarReloj();
            Fabrica.GetInstancia.CrearGtrModificacionesP().ConfirmarAnular(objModificacionPresupuesto, true);
            objModificacionPresupuesto.EstadoModificacion.IdEstadoModificacion = 2;
            objModificacionPresupuesto.EstadoModificacion.Descripcion = "Confirmación";
            forma.lvwModificacionesP.Items.Clear();
            forma.lvwModificacionesP.Objetos = forma.lvwModificacionesP.Objetos = Fabrica.GetInstancia.
                CrearGtrModificacionesP().ObtenerModificaciones(forma.dtpFechaDesde.Value,
                forma.dtpFechaHasta.Value, -1, -1);
            OcultarReloj();
        }
    }
}
```

Anexo 7: Fragmento de código del método "ObtenerAccionesCentralizadas(ArrayList list)"

```
public ArrayList ObtenerAccionesCentralizadas(ArrayList lista)
{
    ArrayList tmp = new ArrayList();

    foreach(IAnteProyectoAccionCentralizada aAC in lista)
        tmp.Add(aAC.ObjAccionCentralizada.IdCategoriaPresupuestaria);

    objDaoComun.IniciarTransaction();
    ArrayList obj = objDaoAnteProyectoAccCentUAC.ObtenerAccionCentnoPertAp(tmp);
    objDaoComun.ReafirmarTransaction();
    return obj;
}
```

Anexo 8: Fragmento de código del método "SalvarAnteAccionCentralizada (IAnteProyectoAccionCentralizada obj)".

```
public void SalvarAnteAccionCent(IAnteProyectoAccionCentralizada objAnteProyectoAccionCentralizada) 4
{
    objDaoComun.IniciarTransaction();
    objDaoComun.SalvarOActualizar(objAnteProyectoAccionCentralizada);

    ArrayList listaAccionesEsp = (ArrayList)objAnteProyectoAccionCentralizada.ListaAnteProyectoAccionEspAccionCent;
    foreach(IAnteProyectoAccionEspAccionCent accEsp in listaAccionesEsp)
    {
        objDaoComun.SalvarOActualizar(accEsp);
        objDaoComun.EliminarColeccionObjetos(accEsp.ListaEliminar);
        foreach(IAnteProyectoEjecutoresAccionCentr ejec in accEsp.ListaAnteProyectoEjecutoresAccionCentr)
        {
            objDaoAnteProyectoAccCentUAC.EliminarPartidasEjecutorAccEspAcc(ejec.
                ListaEliminarAnteProyectoPartidaEjecutoraA);
            objDaoComun.SalvarOActualizar(ejec);
            foreach(IAnteProyectoPartidaEjecutoraA ptda in ejec.ListaAnteProyectoPartidaEjecutoraA)
            {
                objDaoComun.SalvarOActualizar(ptda);
                objDaoComun.SalvarColeccionObjetos(ptda.ListaDesgloseAdicionarBSC);
                objDaoComun.SalvarColeccionObjetos(ptda.ListaDesgloseAdicionarBSFC);
                objDaoComun.SalvarColeccionObjetos(ptda.ListaDesgloseAdicionarMG);
                objDaoComun.EliminarColeccionObjetos(ptda.ListaDesgloseEliminar);
            }
        }
    }
    objDaoComun.ReafirmarTransaction();
}
```