UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

IMPLEMENTACIÓN DE LA CAPA DE PRESENTACIÓN Y NEGOCIO DEL MÓDULO (IPC)

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE

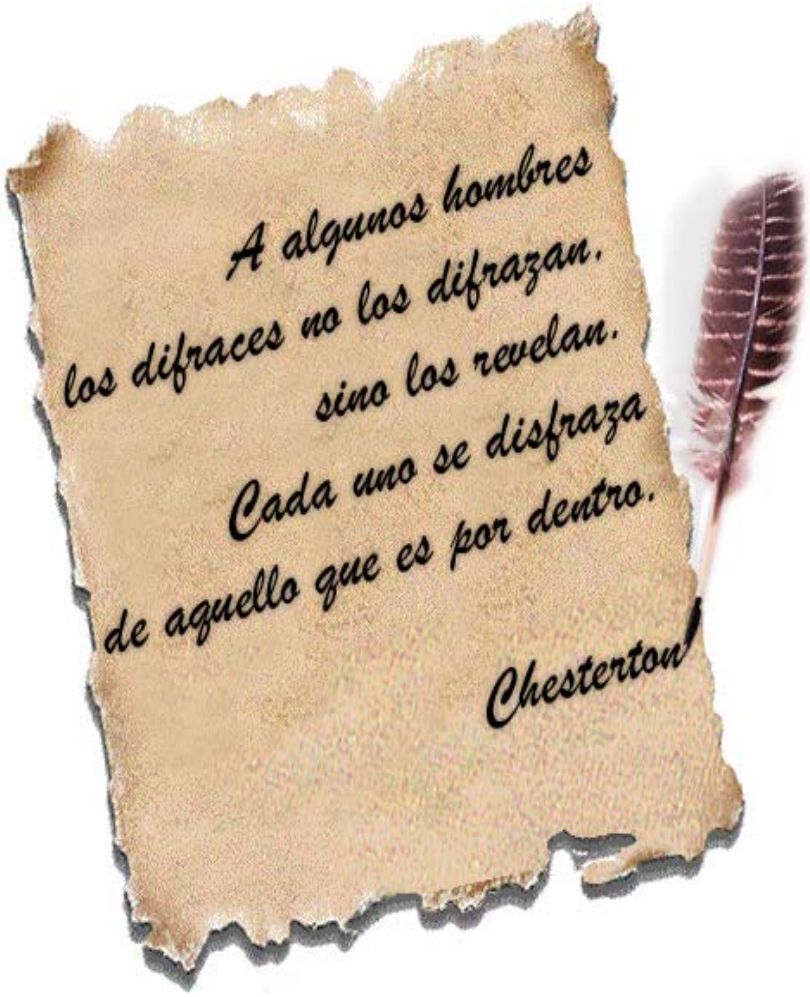
INGENIERO EN CIENCIAS INFORMÁTICAS

AUTOR: IROCHY ORO HURTADO

TUTOR: LIC. MERLYN AVILÉS ESPINOSA

Ciudad de La Habana. Cuba

Junio de 2007


A algunos hombres
los difraces no los difrazan.
sino los revelan.
Cada uno se disfraza
de aquello que es por dentro.
Chesterton

Declaración de autoría.

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Irochy Oro Hurtado	Lic. Merlyn Avilés Espinosa
_____	_____
Autor	Tutor

Datos de contacto.

Datos del **tutor**.

Nombre: Merlyn Avilés Espinosa.
Área: Universidad de las Ciencias Informáticas. Facultad 3.
Cargo: Profesor.
Apartamento: 113102
Teléfono: 8372688
Correo: merlyna@uci.cu

Datos del **autor**.

Nombre: Irochy Oro Hurtado.
Área: Universidad de las Ciencias Informáticas. Facultad 3.
Apartamento: 11210
Teléfono: 8358707
Correo: ioro@estudiantes.uci.cu

Agradecimientos.

A **mi abuela,** por educarme desde pequeño, por formar en mí todos los valores que hoy tengo, y aunque no se encuentre en estos momentos a mi lado, sus últimas palabras fueron que estudiara y llegará a ser alguien en la vida, y siempre pensé mucho en esto, sentía que no podía fallarle.

A **mis padres,** agradecerles por darme la vida, por depositar toda su confianza en mí, y darme fuerzas cada vez que lo necesité, guiándome siempre por buen camino.

A **mis hermanos,** por tenerme siempre presente, siendo yo el mayor de todos, sentí que debía ser un ejemplo para ellos y me esforcé mucho para lograrlo.

A **mis tías,** que siempre estuvieron al tanto de todo, con sus llamadas, guiándome en todo momento.

A **todos mis amigos,** que compartieron junto a mi, tanto buenos y malos momentos, que discutían conmigo a diario de casi todo, seguro que voy a extrañar eso.

A **mi mejor amigo Yuniesky La Rosa Pérez** al que quiero como un hermano, al que tuve desde pequeño a mi lado y creyó siempre en mi, gracias por estar en mi vida y espero que el destino nunca nos separe.

A **mi novia Meylin Rodríguez Cabrales**, por su amor y paciencia. Gracias por su apoyo y ayuda para la realización de este trabajo.

A **Juan Carlos Suárez López**, amigo y guía para mí en todo momento. Gracias por brindarme su experiencia e inteligencia. Gracias por compartir tu conocimiento conmigo y ayudarme a hacer de este trabajo lo que hoy es.

A **todos ustedes,** de todo corazón muchas gracias.

Dedicatoria.

A mí Abuela.

A mís padres, por todo su apoyo.

A toda mí familia, por su confianza depositada.

A mí novia.

A mís amigos.

A los que me han ayudado.

A todos ustedes, porque son parte de mí vida.

Tabla de contenidos.

Índices de figuras.

Índices de tablas.

Resumen.

La existencia en Cuba de dos tipos de monedas, el peso cubano y el peso convertible (CUC), y a su vez de cuatro tipos de mercados: Formal (estatal), Informal (llamado mercado negro), Agropecuario y mercado en Divisa, han conllevado a que existan variedades en los precios para un mismo artículo, haciéndose el trabajo de cálculo de precios algo tedioso, por lo que se convierte en una necesidad realizar una operación que permita sondear los precios de los productos en los diferentes mercados, esta operación es llamada Normalización. La Oficina Nacional de Estadísticas (ONE) creó un sistema que les permite controlar todo lo referente a cálculos e índices de precios, dicho sistema posee algunas limitaciones: no permite adicionar nuevos productos o eliminarlos, haciendo empalmes con años anteriores, lo que implica que estos procesos se están realizando manualmente. Las carencias que presenta el sistema actual evidencia que es de **vital importancia** desarrollar un nuevo sistema más flexible que permita el control del cálculo del índice de precios al consumidor, así como gestionar la canasta de productos.

El trabajo surge a raíz de la necesidad que tiene la ONE hoy día, de adquirir un sistema fiable, que brinde funcionalidades para gestionar nuevos productos, calcular índices de precios y realizar comparaciones de productos por provincias, etc.

Este trabajo diploma contiene la implementación de las capas de Presentación y Negocio para el módulo nacional Índices de Precios al Consumidor ***(IPC)***, el cual permitirá las funcionalidades de Seguridad, Gestión de Usuarios y Productos, Emisión de tablas e Índices de Precios, además de proporcionar un manejo fácil mediante las interfaces que estarán contempladas en la capa de Presentación.

Palabras Claves.

***IPC**:* Índices de Precios al Consumidor.

***ONE**:* Oficina Nacional de Estadísticas.

Fusión: Procesamiento periódico de precios, a los productos provenientes de los respectivos mercados a nivel nacional.

Interfaz: Medio que permite la interacción entre dos elementos.

Capa: Conjunto de subsistemas que comparten la misma generalidad y jerarquía.

Implementación: Término que se refiere a la forma en que se van a codificar las capas de Presentación y Negocio.

Casos de usos: Fragmentos de funcionalidad dentro de un sistema que reportan beneficios para los usuarios.

Instancia: Copia en memoria de una clase con valores en los atributos.

PSP: Proceso Personal de Software.

TSP: Proceso de Software en Equipo.

Abstract.

The existence in Cuba of two types of currencies Cuban peso and (CUC), and in turn four types of markets: Formal (state), Informal (called black market), Agricultural and market in Foreign currency, they have borne to that varieties exist in the prices for oneself article, being made the work of calculation of prices something tedious, for what becomes a necessity to carry out an operation that allows to sound the prices of the products in the different markets, this operation it is called Normalization. The Cuban company (ONE) National Office of Statistical created a system that allows them to control all it with respect to calculations and indexes of prices, this system possesses some limitations like to add new products or to eliminate making connections with previous years, what implies that these processes are being carried out by hand, that which tells us that it is of vital importance to develop a new more flexible system that allows in a better way the control of the calculation from the index of prices to the consumer.

The work arises soon after the necessity that has the NOE nowadays, of acquiring a reliable system that offers functionalities to negotiate new products, to calculate indexes of prices and to carry out comparisons of products for counties, etc.

This work diploma contains the implementation of the layers of Presentation and Business for the module national Indexes of Prices to the Consumer (IPC), which will allow the functionalities of Security, Administration of Users and Products, Emission of charts and Indexes of Prices, besides providing an easy handling by means of the interfaces that will be contemplated in the layer of Presentation.

Introducción.

Hoy en día los sistemas informáticos están presentes en todas las esferas de la sociedad, en la política, en lo social, en la cultura, en la economía, en fin en todas sin excepción. El uso de aplicaciones informáticas ha sido de carácter **vital** para realizar trabajos que se tornan complejos, tediosos, que requieren de mucho tiempo y detalle, los cuales si fueran realizados por el hombre estarían expuestos a cometer grandes errores y muchas veces pues simplemente con llevaría a la perdida de tiempo, Lo que parece algo muy factible.

Se ha dado cuenta la humanidad la necesidad de informatizarnos cada día más, automatizar los procesos que enmarcan un trabajo o alguna situación determinada, los cuales garantizan un aprovechamiento óptimo del tiempo de trabajo, reducción de errores en un gran margen, permitiendo muchas facilidades, logrando una interacción más amena y cómoda entre los usuarios y su puesto de trabajo. Dado esto se puede afirmar que si se crearan aplicaciones informáticas se resolverán muchos problemas que hoy en día se presentan, por ejemplo, particularmente en nuestro país tenemos dos monedas y varios mercados, problema que solo existe en nuestro territorio, por tanto para hacer cálculos de índices de precios se torna algo complejo y sabiendo además que en cada mercado existen tanto productos iguales como productos diferentes es verdaderamente imposible realizar dichos cálculos de forma manual, además de todos los procesos que deben realizar como comparaciones entre provincias, teniendo en cuenta fechas, provincias, y mercados, es una **necesidad vital** la creación de una aplicación para realizar todas estas operaciones. Ahora bien si es cierto que la creación de sistemas facilita todas estas operaciones, también debemos destacar que lograr un sistema completo y que satisfaga las necesidades del cliente es bastante complejo, la empresa ONE cuenta en la actualidad con un sistema debido a la necesidad que tiene de calcular índices de precios, comparar productos, entre otras cosas, y dicho sistema no cumple con todas las operaciones y funcionalidades que la empresa necesita por tanto, urge la necesidad de un sistema que resuelva esta problemática. Ya que el cálculo de índice de precios es de **importancia vital** pues da la medida, de cómo ondean los precios hoy en día, con respecto a varios periodos anteriores, llegando a conclusiones, siendo posible la toma de medidas si fuese necesario, para cada día poder mejorar la situación económica de cada uno de nuestros habitantes, después de analizar esto se observa varias deficiencias las cuales dan origen a un problemática.

La **situación problemática:** esta dada porque la empresa ONE cuenta con un sistema poco flexible para realizar el proceso del cálculo del ***IPC***, el cual presenta varias deficiencias, pues no brinda la funcionalidad de adicionar, eliminar productos, realizar empalmes con respecto a años anteriores, entre otras cosas, ya que no se realizó una arquitectura adecuada, y no se realizó de manera eficiente la implementación de la capa de Negocio.

A partir del análisis a esta problemática se puede formular el siguiente **problema científico:** ¿Cómo lograr un nuevo sistema que permita de manera eficiente el cálculo del Índice de Precios al Consumidor ***IPC***?

Donde el **objeto de estudio** a tratar será: Implementación de las capas de Presentación y Negocio en aplicaciones para la automatización.

El **objetivo general** de este trabajo diploma es desarrollar las capas de Presentación y Negocio creando un nuevo sistema para el cálculo de ***IPC***, permitiendo una aplicación más segura, eficiente, que erradique las deficiencias del sistema actual.

Para dar cumplimiento a este objetivo, se proponen los siguientes **objetivos específicos:**

- Definir las clases que intervendrán en el sistema así como sus relaciones.
- Definir roles con sus respectivos privilegios garantizando la seguridad del sistema.
- Dar cumplimiento a todos los requisitos funcionales de una forma eficiente.
- Realizar pruebas a las soluciones propuestas.

Del anterior objeto de estudio se deriva el siguiente **campo de acción:** Implementación de las capas de Presentación y Negocio en Microsoft Visual C # para la automatización de cálculos estadísticos para la ONE en Cuba.

Lo que conlleva a la **hipótesis** siguiente: Con la implementación, de forma correcta, de las capas de Presentación y Negocio se logrará un nuevo sistema más flexible, que permitirá de manera eficiente el proceso del cálculo de Índice de Precios al Consumidor ***IPC***.

Y para lograr de manera exitosa el cumplimiento de la hipótesis expuesta anteriormente, se hizo necesario el cumplimiento de las **tareas de la investigación** que se presentan a continuación.

- Análisis y estudio de los requisitos funcionales y no funcionales para la implementación del módulo ***IPC***.
- Realizar un estudio de la tecnología o lenguaje utilizado.
- Programar orientado a objetos en el lenguaje Microsoft Visual C#.
- Estudio de la herramienta NUnit Framework, para la realización de pruebas de validación.

Para la realización de forma correcta de la implementación de las capas que formarán el modulo ***IPC***, se requiere de conocimiento, se analizará al detalle todos los requisitos funcionales, para llegar a tener una idea del tiempo de trabajo y poder desglosarlo en partes, por tanto se hace necesario la utilización de la **metodología de la investigación** científica, apoyado en algunos de los métodos existentes hoy en día.

Métodos Teóricos:

- **Histórico - Lógico:** permitiéndonos hacer un estudio del comportamiento actual del desarrollo de sistemas de gestión, dándole solución al problema dado.
- **Sistémico**: es uno de los métodos más importantes, permite desglosar los elementos que contendrá el sistema así como asignar las relaciones y jerarquías que existen entre ellos.

Métodos Empíricos:

- **Observación:** mediante este método percibimos y planificamos como quedaría concebido el sistema.
- **Medición:** para medir la eficiencia de los algoritmos, así como el tiempo de desarrollo del sistema.

La culminación exitosa de este trabajo diploma permitirá lograr resultados satisfactorios, dando lugar a importantes **aportes prácticos:** El desarrollo de las capas de Presentación y Negocio para el módulo nacional ***IPC*** propuesto en el trabajo diploma contiene un gran valor práctico que se manifiesta en su aplicación:

- El desarrollo de estrategias de integración que permitieron una solución eficiente a las exigencias trazadas y una mejor viabilidad y organización en la programación.
- El proceso de la captura de errores de las distintas excepciones lanzadas por el sistema y la base de datos, asegurando un correcto funcionamiento y especificándole al usuario el tipo de error cometido.
- El sistema de seguridad con el que cuenta la aplicación, pretende brindar la integridad de los datos, así como la reusabilidad de la misma, puesto que puede ser perfectamente utilizado este sistema en la creación de otras aplicaciones.

El **trabajo diploma** esta estructurado en tres capítulos:

Capítulo 1: Fundamentación Teórica.

Se hace un estudio del arte de los sistemas estadísticos para la automatización de índices de precios al consumidor, así como las principales tendencias de la programación hoy en día, y sus características, especialmente de la Programación Orientada a Objetos (POO), se analiza también las ventajas que da la utilización de estándares de programación, la importancia que tiene para el desarrollo de aplicaciones el uso del Proceso Personal de Software (PSP), y el análisis de la eficiencia de los algoritmos, así como una breve descripción de la plataforma utilizada, que actualmente brinda un entorno cómodo y fácil de trabajo.

Capítulo 2: Descripción y análisis de la solución propuesta.

En este capítulo se realiza una descripción de la propuesta de solución al problema planteado en el diseño teórico de la investigación, para ello se expresa una solución concreta del problema, plasmando de esta manera los artefactos generados para dar cumplimiento al módulo nacional ***IPC***, empezando por la

arquitectura definida por el arquitecto, seguido una valoración crítica del modelo de diseño propuesto por el diseñador, estrategias de integración, patrones utilizados, clases u operaciones que permiten dar respuesta de forma rápida y segura a los requisitos planteados por el cliente, la captura de errores, así como la utilización de las distintas actividades y normas planteadas dentro del PSP para el control y planificación de los desarrolladores durante el ciclo de implementación del software.

Capítulo 3: Validación de la solución propuesta.

Este capítulo muestra la validación realizada a la solución propuesta en el capítulo anterior, para ello se analiza la solución concreta del problema, presentando y describiendo las clases de prueba que verifican la validez de los casos de usos (CU) que conforman el módulo nacional ***IPC***, además de una breve descripción de la herramienta utilizada, NUnit Framework.

Capítulo 1: Fundamentación Teórica.

1.1 Introducción.

En este capítulo se hará un estudio del estado del arte, el uso de las Tecnología de la Informáticas y las Comunicaciones *(TIC)*, una breve reseña histórica, se hablará del ***IPC*** y como se ve esto en el mundo actual, así como maneja la empresa ONE en Cuba esta importante estadística, se darán características especiales de la Programación Orientada a Objetos y sus ventajas, cálculos de eficiencia de algoritmos, y una concepción de la importancia del PSP, también se detallará la herramienta de desarrollo a utilizar, las facilidades que esta proporciona. Entre otras cosas.

1.2 Uso de las TIC. Reseña Histórica.

Se sabe de antemano que en tiempos anteriores para realizar ciertos tipos de trabajos se necesitaba de mucho esfuerzo y tiempo, el nivel de cometer errores era grande y por tanto la calidad y aprovechamiento era malo, hoy en día se ha podido resolver esta problemática, con la aparición de la computadora y los software que en ella se ejecutan, con el uso de las *TIC,* las condiciones de trabajo han mejorado en un gran nivel. En los primeros días de la computación, las computadoras se usaban principalmente en aplicaciones militares, pues su costo era muy alto. Hoy en día están presentes prácticamente en todas las actividades del hombre. Sin duda alguna, su mayor aplicación se da en ambientes de negocios, en el procesamiento de datos relevantes para actividades administrativas. Esto no hubiese sido posible sin la dedicación y entrega de un grupo de Personas que años tras años lo han entregado todo por mejorar cada día más esta rama. Personas que están integradas a lo que podemos llamar grupos de desarrollo de software las cuales juegan determinados roles dentro del grupo como Analistas, Diseñadores, Arquitectos, Programadores entre otros, dando lugar a la obtención de productos de calidad mediante la programación por mencionar alguna, tarea que desempeña el Programador. [MARE, 1999]

1.3 Índices de precios al consumidor *IPC.*

IPC es la abreviatura de Índice de Precios al Consumidor, Índice de Precios de Consumo o Índice de Precios al Consumo (CPI en inglés).
Es un índice de precios el valor numérico que representa una medida relativa de diferencia de precios de bienes y servicios en diferentes situaciones temporales. Esto es, medir las variaciones de precios de una muestra de productos, (conocida como "canasta" o "cesta") con respecto a un período base determinado. De esta forma se pretende medir, mensualmente, la evolución del nivel de precios de bienes y servicios de consumo en un país. El objetivo es medir las diferencias de nivel de precios de una canasta con respecto a una región determinada. [ONE, 2000]

1.4 Sistemas actuales para cálculos de *IPC* en el mundo.

La creación de software para cálculos estadísticos ha sufrido un gran aumento en los últimos años. Este crecimiento se debe al incremento proporcional de las empresas en el mundo. El perfeccionamiento empresarial a nivel mundial constituye una constante evolución, y de ese modo, las organizaciones necesitan de una automática evaluación de todos los aspectos implicados en sus procesos. Precisamente los sistemas estadísticos permiten acceder a un mejor conocimiento de la información contenida en los datos mediante metodologías y procesos de recogida, análisis e interpretación. La evolución del software estadístico en los últimos años ha significado un importante ahorro en tiempo, en precisión y en calidad de los resultados en los procesos de las empresas. [ALFARO, 2001]

En la actualidad, la mayoría de los sistemas estadísticos presentan funcionalidades que permiten el trabajo con indicadores visuales capaces de mostrar al usuario resultados en gráficas, de una forma más amigable. Esta característica la presentan con mayor frecuencia los sistemas que abarcan la estadística descriptiva, o sea, los que trabajan con modelos matemáticos, los que calculan estimaciones, etc. [ALFARO, 2001]

Sin embargo, muchos de los sistemas que se usan para calcular ***IPC*** han evolucionado poco dentro del campo estadístico. Esto se debe a que la metodología del ***IPC*** no ha variado ni ha sufrido cambios en los últimos 15 años, pero eso si, cada país tiene una forma independiente de recolectar los datos, de procesar las encuestas, entre otros procesos. A continuación se analiza uno de los sistemas utilizados para calcular ***IPC***. [ALFARO, 2001]

En Perú hay implementado un sistema el cual se llama ***Procesamiento y Difusión del IPC*** este sistema trae consigo cuatro subsistemas

- Sub-sistema de entrada de datos inteligente.
- Sub-sistema de consistencia analítica.
- Sub-sistema de entrada de cálculo.
- Sub-sistema de entrada de resultados.

Para la realización del mismo se utilizan los cuatros subsistemas, pero se aclara que el proceso automático solo es en el sub-sistema de cálculo. [ALFARO, 2001]

1.5 Sistema para cálculos de *IPC* en Cuba.

Si se sabe que en el mundo entero existen aplicaciones que realizan cálculos de ***IPC***, entonces porque la empresa ONE no puede adquirir una y resolver su problema, pues, porque como se dijo anteriormente cada país lo hace de manera particular y en Cuba esto se torna un poco más complejo, dado por la existencia de varios mercados y dos tipos de monedas, por lo que la empresa se vio necesitada de ayuda y acudió a la Universidad de las Ciencias Informáticas (UCI), en busca de soluciones, ya que el sistema utilizado para realizar dichos cálculos no es eficiente, siendo así posible la realización de este trabajo diploma.

1.6 ¿Que es Programación?

Se llama programación a un conjunto finito de instrucciones que pueden ejecutarse dando lugar a resultados.

En la actualidad la programación ha evolucionado mucho se ha ido perfeccionando en aras de mejores y más óptimos productos con una mayor calidad. Durante años, los programadores se han dedicado a construir aplicaciones muy parecidas que resolvían una y otra vez los mismos problemas. Para conseguir que los esfuerzos de los programadores puedan ser utilizados por otras personas se creó la Programación Orientada a Objetos. Que es una serie de normas de realizar las cosas de manera que otras personas puedan utilizarlas y adelantar su trabajo, de manera que consigamos que el código se pueda reutilizar. La Programación Orientada a Objetos POO surge en Noruega en 1967 con un lenguaje llamado Simula 67, desarrollado por Krinsten Nygaard y Ole-Johan Dahl, en el centro de cálculo noruego. [*Programación Orientada a Objeto (POO)*, 2007]

1.7 ¿Que es la Programación Orientada a Objetos POO?

La POO es una forma especial de programar, más cercana a como expresaríamos las cosas en la vida real. Es una manera de pensar, otra manera de resolver un problema. [ALVAREZ, 2007]

1.8 Paradigmas de la POO.

La POO no es difícil, pero es una manera especial de pensar, a veces subjetiva de quien la programa, de manera que la forma de hacer las cosas puede ser diferente según el programador. Aunque podamos hacer los programas de formas distintas, no todas ellas son correctas, lo difícil no es programar orientado a objetos sino programar bien. Programar bien es importante porque así nos podemos aprovechar de todas las ventajas de la POO. [FERNÁNDEZ, 2007]

La POO es un paradigma de la programación de computadores; esto hace referencia al conjunto de teorías, estándares, modelos y métodos que permiten organizar el conocimiento, proporcionando un medio bien definido para visualizar el dominio del problema e implementar en un lenguaje de programación la solución a ese problema. [FERNÁNDEZ, 2007]

La POO se basa en el modelo objeto donde el elemento principal es el objeto, el cual es una unidad que contiene todas sus características y comportamientos en sí misma, lo cual lo hace como un todo independiente pero que se interrelaciona con objetos de su misma clase o de otras clase, como sucede en el mundo real. [FERNÁNDEZ, 2007]

Anterior al paradigma de objetos, está el paradigma algorítmico o de procesos, el cual se fundamenta en los procesos o funciones que se llevan a cabo en el mundo real dentro del dominio del problema analizado. Se refiere a lo que entra, como lo maneja el proceso, y lo que sale del proceso. La programación tradicional la sustentan los procesos, algoritmos, bloques de construcción modulares cuya abstracción va de lo general a lo particular, mientras que en la POO tiene como marco de referencia conceptual el objeto, el cual pertenece a una clase que agrupa a todos compañeros con las mismas características y un comportamiento similar. [FERNÁNDEZ, 2007]

Una ventaja de la POO frente al paradigma algorítmico es la facilidad que brinda a través de sus herramientas, de concebir, analizar, modelar, diseñar e implementar el mundo real de manera fiel a como se presenta en la realidad; el paso que hay desde la concepción y asimilación del problema hasta la implementación del mismo es un proceso que se hace de manera casi natural. Esto porque el mundo está lleno de objetos reales, los cuales se puede representar como tales en una solución computarizada. [FERNÁNDEZ, 2007]

1.8.1 El Paradigma Concurrente.

Se dice que dos o mas procesos son concurrentes si están construidos de manera tal que pueden ejecutarse al mismo tiempo y compartiendo recursos. Considerando un entorno multi-thread, cada hilo representa un proceso individual ejecutándose en un sistema. Generalmente, cada hilo controla un único aspecto dentro de un programa. Todos los hilos comparten los mismos recursos. [TESO, 2007]

1.8.2 El Paradigma Lógico.

Una forma de razonar para resolver problemas en matemáticas se fundamenta en la lógica de primer orden. El conocimiento básico de las matemáticas se puede representar en la lógica en forma de axiomas, a los cuales se la agregan reglas formales para deducir cosas verdaderas (teoremas). Los lenguajes que utilizan esta lógica se llaman lenguajes declarativos, porque todo lo que tiene que hacer el programador para solucionar un problema es describirlo vía axiomas y reglas de deducción. [TESO, 2007]

1.8.3 El Paradigma Funcional.

La programación funcional tiene como objeto imitar las funciones matemáticas lo mas posible. Un lenguaje funcional posee la propiedad matemática de transparencia referencial, lo que significa que una expresión representa siempre el mismo valor. Esto permite razonar sobre la ejecución de un programa y demostrar matemáticamente que es correcto. [TESO, 2007]

Las variables de un lenguaje funcional son como las variables en algebra. Inicialmente representan un valor desconocido que, una vez calculado, ya no cambia. En un programa funcional, el orden de evaluación de las subexpresiones no afecta al resultado final, por lo tanto las subexpresiones pueden ejecutarse en forma paralela para hacer más eficiente el programa. [TESO, 2007]

1.9 Características de la POO.

Se enfatizara de forma breve las características de este tipo de programación, y las ventajas que proporciona la abstracción, encapsulación, modularidad, jerarquización, tipificado, concurrencia, y persistencia. [POL, 1996-1997]

1.9.1 Abstracción.

Denota las características esenciales que distinguen a un objeto de otros tipos de objetos, definiendo precisas fronteras conceptuales, relativas al observador.

- Surge del reconocimiento de similaridades entre ciertos objetos, situaciones o procesos en el mundo real.
- Decide concentrarse en estas similaridades e ignorar las diferencias.
- Enfatiza detalles con significado para el usuario, suprimiendo aquellos detalles que, por el momento, son irrelevantes o distraen de lo esencial.
- Deben seguir el "principio de mínimo compromiso", que significa que la interface de un objeto provee su comportamiento esencial, y nada más que eso. Pero también el "principio de mínimo asombro": capturar el comportamiento sin ofrecer sorpresas o efectos laterales. [POL, 1996-1997]

1.9.2 Encapsulación.

Es el proceso de compartimentalización de los elementos de una abstracción que constituyen su estructura y comportamiento. La encapsulación sirve para separar la interface de una abstracción y su implementación.

- Es un concepto complementario al de abstracción.
- La encapsulación esconde la implementación del objeto que no contribuye a sus características esenciales.
- La encapsulación da lugar a que las clases se dividan en dos partes:
    1. Interface: captura la visión externa de una clase, abarcando la abstracción del comportamiento común a los ejemplos de esa clase.
    2. Implementación: comprende la representación de la abstracción, así como los mecanismos que conducen al comportamiento deseado.

    Se conoce también como ocultamiento o privacidad de la información. [POL, 1996-1997]

1.9.3 Modularidad.

Es la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y vagamente conexos.

- Cada módulo se puede compilar separadamente, aunque tengan conexiones con otros módulos.

- En un diseño estructural, modularización comprende el agrupamiento significativo de subprogramas. En diseño orientado a objetos, la modularización debe ceñirse a la estructura lógica elegida en el proceso de diseño.
- Dividir un programa en componentes individualizados reduce en alguna manera su complejidad.
- También se separan los módulos interface de los módulos con implementación. [POL, 1996-1997]

1.9.4 Jerarquización.

Es una clasificación u ordenación de las abstracciones.

- Por jerarquía denotamos el orden de relación que se produce entre abstracciones diferentes.
- Los tipos de jerarquía más útiles:
    1. Herencia (generalización/especialización, padre/hijo, jerarquía del tipo "es un"...). Una clase (subclase) comparte la estructura o comportamiento definido en otra clase, llamada superclase.
    2. Herencia múltiple Una clase comparte la estructura o comportamiento de varias superclases.
    3. Agregación Comprende relaciones del tipo "es parte de" al realizar una descomposición.

Relaciones entre los conceptos asociados al modelo de objetos.

- Los conceptos de abstracción y encapsulación son conceptos complementarios: abstracción hace referencia al comportamiento observable de un objeto, mientras encapsulación hace referencia a la implementación que la hace alcanzar este comportamiento.
- Existe una tensión entre los conceptos de encapsulación de la información y el concepto de jerarquía de herencia, que requiere una apertura en el acceso a la información.
- Ofrece mucha flexibilidad, pudiendo disponer de tres compartimentos en cada clase:
    1. Privado: declaraciones accesibles sólo a la clase (completamente encapsulado)
    2. Protegido: declaraciones accesibles a la clase y a sus subclases.
    3. Público: declaraciones accesibles a todos los clientes.

Además de estos tres tipos, soporta la definición de clases cooperativas a las que se les permite acceder a la parte privada de la implementación. [POL, 1996-1997]

1.9.5 Tipificado.

Tipificar es la imposición de una clase a un objeto, de tal modo que objetos de diferentes tipos no se puedan intercambiar, o se puedan intercambiar solo de forma restringida.

- Tipo es una caracterización precisa de las propiedades estructurales y de comportamiento que comparten una colección de entidades.
- Grosso modo, tipo y clase pueden considerarse sinónimos.
- Existen lenguajes fuertemente tipificados (Ada) y débilmente tipificados. Estos últimos soportan polimorfismo, mientras que los fuertemente tipificados no. [POL, 1996-1997]

1.9.6 Concurrencia.

Es la propiedad que distingue un objeto activo de uno no activo. Concurrencia permite que diferentes objetos actúen al mismo tiempo, usando distintos threads de control. [POL, 1996-1997]

1.9.7 Persistencia.

Es la propiedad por la cual la existencia de un objeto trasciende en el tiempo (esto es, el objeto sigue existiendo después de que su creador deja de existir) o en el espacio (esto es, la localización del objeto cambia respecto a la dirección en la que fue creado). [POL, 1996-1997]

1.10 Beneficios de los estándares de codificación.

Esto es uno de los factores claves para obtener calidad en un producto, y se ha venido perfeccionando con el transcurso de los tiempos, en la actualidad se habla mucho acerca del peso que tiene esto para las aplicaciones. [TORRES, 2006]

Facilitando en gran medida el entendimiento de las aplicaciones así como su fácil mantenimiento, pudiendo reutilizar su código en nuevos sistemas. Usando estándares estamos siguiendo un patrón que

nos asegura que un futuro nuestra aplicación seguirá viéndose exactamente igual y funcionando exactamente igual. [TORRES, 2006]

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Importante para cuando en un sistema trabaje más de un programador, parezca que solo uno lo ha hecho todo. [TORRES, 2006]

Al comenzar un proyecto de software, establezca un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. [TORRES, 2006]

Cuando el proyecto de software incorpore código fuente previo, o bien cuando realice el mantenimiento de un sistema de software creado anteriormente, el estándar de codificación debería establecer cómo operar con la base de código existente. [TORRES, 2006]

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. [TORRES, 2006]

Aunque la legibilidad y la mantenibilidad son el resultado de muchos factores, una faceta del desarrollo de software en la que todos los programadores influyen especialmente es en la técnica de codificación. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas. [TORRES, 2006]

Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento. Además, si se aplica de forma continuada un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas, y, posteriormente, se efectúan revisiones del código de

rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener. [TORRES, 2006]

Aunque el propósito principal para llevar a cabo revisiones del código a lo largo de todo el desarrollo es localizar defectos en el mismo, las revisiones también pueden afianzar los estándares de codificación de manera uniforme. La adopción de un estándar de codificación sólo es viable si se sigue desde el principio hasta el final del proyecto de software. No es práctico, ni prudente, imponer un estándar de codificación una vez iniciado el trabajo. [TORRES, 2006]

1.11 Análisis de la eficiencia de algoritmos.

Todos los problemas de la matemática y la lógica podrán ser resueltos mediante algoritmos. Sin embargo no todas las soluciones son igualmente eficientes.
Normalmente los algoritmos de interés reciben una serie de datos de volumen o tamaño N dado. De hecho es posible tener infinidad de algoritmos que resuelvan el mismo problema.
Cada método se tomará más o menos unidades de tiempo en hacerlo. Si el algoritmo tiene que evaluar una muestra de entrada extensa, obviamente, su tiempo de ejecución dependerá de la cantidad de elementos en la muestra de entrada (N). En muchos casos existe una dependencia clara de la cantidad de elementos en la muestra de entrada, en otros la dependencia es del valor de ese dato de entrada. En general, tenemos que el tiempo de ejecución de un programa dependerá del 'tamaño' de los datos de entrada. A esta dependencia se la llama complejidad temporal y se la suele expresar como tiempo de ejecución en alguna unidad de tiempo = T(N). Se la distingue de la complejidad espacial que nos da la dependencia del espacio necesitado por el algoritmo según el tamaño de la muestra de entrada. Realmente en la práctica no se trabaja con T(N) sino con funciones que clasifiquen estas funciones acotando su valor asintóticamente. Se dirá que un algoritmo es eficiente si al aumentar el tamaño de la muestra de la entrada, el tiempo de ejecución crece de forma tan sólo polinomial en función de aquel tamaño. En otro caso, si el crecimiento es de tipo exponencial (más exactamente O(n elevado a k b elevado n)), se dice que es ineficiente o intratable. Esto es así porque en estos últimos algoritmos, tamaños no muy grandes de la entrada dan tiempos de ejecución de días. . . aún en las máquinas más rápidas. El tiempo de ejecución se puede mejorar (quizás con paralelismo, etc.) de forma lineal o

polinomial pero nunca se puede conseguir una mejora en hardware que reduzca la tendencia exponencial. [PÉREZ *et al.*, 2007]

Como hemos visto, en principio puede suponerse que para cualquier problema resoluble (atención, hemos dicho resoluble), existe un algoritmo y una máquina que lo resuelve. La teoría que estudia la posible algoritmización de la solución a un problema se le denomina computabilidad. Otro problema, aparte de la computabilidad, es la complejidad (en términos de espacio y tiempo) de los algoritmos para resolver los problemas, y de los propios problemas (con modelos independientes de la máquina que los ejecute). Hemos visto que existen problemas cuya solución se retrasa en función de “b elevado a n” o incluso “n elevado a n” términos a tener en cuenta, son algoritmos exponenciales. La solución es ineficiente o inadecuada computacionalmente. Otros algoritmos, sin embargo tienen un comportamiento “más sensato”, tardando del orden de “n elevado a c”, se les llama polinómicos y se consideran adecuados y más o menos eficientes. [PÉREZ *et al.*, 2007]

1.11.1 Eficiencia de un programa.

Decimos que un programa es eficiente cuando consume pocos recursos durante su ejecución. Durante su ejecución, un programa consume básicamente dos recursos:

- Tiempo del procesador - coste temporal
- Espacio en memoria - coste espacial [PÉREZ *et al.*, 2007]

1.11.2 Factores de dependencia.

- El algoritmo utilizado.
- El tamaño de la entrada o complejidad del problema a resolver. El tamaño de la entrada es una medida de la cantidad de datos que deberán procesarse.
- Características de la máquina donde se ejecuta
- Calidad del software de desarrollo usado en la implementación [PÉREZ *et al.*, 2007]

1.11.3 Expresiones y sintaxis de la eficiencia.

El tiempo de ejecución y el espacio de memoria requeridos por un programa se expresan en función del tamaño de los datos de entrada n. Esto es así porque el tamaño de los datos es lo que afecta en mayor medida al consumo de recursos de un programa. Si cambiamos de máquina o compilador, el consumo de recursos varía, a lo sumo, en un factor constante.
El hecho anterior se enuncia como el **Teorema de la Invarianza:**

El teorema de la Invarianza: establece que al cambiar de lenguaje, compilador o máquina, el consumo de recursos se multiplica o divide a lo sumo por una constante. Por lo tanto, factores ligados a la máquina donde se ejecuta el programa o al software de desarrollo utilizado, tienen una repercusión moderada en el consumo de recursos. [PÉREZ *et al.*, 2007]

En cambio, la dependencia del tamaño de los datos presenta con frecuencia otro comportamiento. Para muchos algoritmos, al incrementar un poco el tamaño de los datos, el tiempo y la memoria consumidos crecen de forma exagerada: se duplican, se triplican, crecen de forma exponencial, etc. El hecho de expresar la eficiencia de un algoritmo en función del tamaño de los datos de entrada es algo bastante lógico, porque el tamaño de los datos determina, por un lado, la cantidad de instrucciones que tiene que ejecutar el programa, y por otro, la cantidad de elementos que tiene que almacenar. [PÉREZ *et al.*, 2007]

1.11.4 Teoría de los casos.

A la hora de expresar la eficiencia de un programa, aún hay otra decisión a tomar. El consumo de recursos no sólo depende del tamaño de los datos, sino también de las características de los datos. [PÉREZ *et al.*, 2007]

1.11.5 Métodos del cálculo de la eficiencia.

Básicamente hay dos métodos:

- Implementar el programa y probarlo (prueba empírica)
- Analizar el algoritmo en el que se basa el programa (análisis del algoritmo)

La prueba empírica consiste en implementar y ejecutar el programa y medir el tiempo y el espacio en una máquina concreta y con unos datos concretos.
El análisis de la complejidad del algoritmo consiste en estimar el tiempo y el espacio a partir del análisis de las instrucciones que componen el algoritmo en el que se basa el programa. [PÉREZ *et al.*, 2007]

1.11.6 Cálculo de la complejidad temporal.

Básicamente existen dos formas de analizar la complejidad temporal de un algoritmo:

- Contar el número de instrucciones que se ejecutan (en función del tamaño de entrada *n*)
- Analizar el comportamiento asintótico del algoritmo (qué pasa cuando *n* tiene a infinito) [PÉREZ *et al.*, 2007]

1.11.7 Cálculo de la complejidad espacial.

Dado un algoritmo, nos interesa conocer su complejidad espacial, es decir, clasificar su coste espacial E(n) en una de las formas de crecimiento. Como ya se ha señalado, calcularemos siempre el coste para el caso peor, y trataremos de establecer una cota superior a E(n) usando una notación asintótica. [PÉREZ *et al.*, 2007]

Un programa puede requerir memoria para muchos conceptos distintos:

- Variables estáticas y locales (heap)
- Variables dinámicas (pila)
- Parámetros de funciones (pila)

Un programa durante su ejecución no consume una cantidad de memoria fija, sino que depende de las variables que vaya creando dinámicamente, de sucesión de llamadas a funciones, etc. lo cual puede

complicar considerablemente el análisis. Sin embargo, hay un caso sencillo: los algoritmos no recursivos que no utilizan memoria dinámica. En estos casos, para calcular la complejidad espacial de un algoritmo deben seguirse las siguientes reglas obvias:

- Una variable de tipo elemental ocupa una cantidad constante.
- Una variable de tipo tabla de n elementos ocupa n veces el espacio de cada elemento. [PÉREZ *et al.*, 2007]

1.12 Importancia del Proceso de Software Personal en el desarrollo de sistemas.

"Mayor calidad y reducir el costo"

La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente.
En la actualidad Ing. de experiencia introducen entre 7 y 10 defectos por línea de código, la mayoría de estos defectos son detectados en las pruebas.
Muchas organizaciones concentran sus esfuerzos en arreglar los errores en vez de prevenirlos.
Por lo tanto surge la necesidad de hacer planes ajustados, realizar detalles en el diseño, eliminación temprana de los defectos, inspecciones efectivas, y muy importante centrarse en la calidad todo el tiempo. [HUMPHREY, 2001]

Por tanto se necesita:

- Una infraestructura y ambiente que los soporte.
- Procesos (reglas y pasos) para ponerlos en práctica.
- Un sistema de métricas para gestionar y controlar los resultados.

Para esto tenemos el Proceso de Software Personal (PSP)
Que no es más que las habilidades individuales y la disciplina que debe tener un desarrollador de software, es exigirse uno mismo. Ser más crítico con su trabajo conocerse uno mismo. [HUMPHREY, 2001]

1.12.1 Paradigma del PSP.

- Se definen metas de procesos individuales
- Cada cual define los métodos que usa
- Miden su trabajo individualmente
- Analizan sus resultados
- Basados en estos resultados ajustan sus métodos alcanzar mejor sus metas personales [HUMPHREY, 2001]

1.12.2 Principios del PSP.

- La calidad de un sistema de software está regida por la calidad del proceso utilizado para desarrollarlo.
- La calidad de un sistema de software está regida por la calidad de sus peores componentes.
- La calidad de un componente de software está regida por el individuo que lo desarrolló.
- Esto está regido por el conocimiento, disciplina y compromiso del individuo. [HUMPHREY, 2001]
- Cómo un profesional del software, debe conocer su propia productividad.
- Se debe medir, registrar y analizar el trabajo.
- Se debe aprender de las variaciones de la productividad.
- Se debe incorporar estas lecciones en las prácticas personales. [HUMPHREY, 2001]

1.12.3 Gestión del Tiempo.

- Para hacer un plan realista, tienes que controlar la forma en que usas tu tiempo
- Debes estimar la exactitud de tus estimaciones.
- Determinar equivocaciones en los planes anteriores.
- Clasifica tus principales actividades.
- Registra el tiempo dedicado a cada una de las actividades principales.
- Registra el tiempo de forma normalizada.

- Guarda los datos en un lugar adecuado. [HUMPHREY, 2001]

1.12.4 Ideas para registrar el tiempo.

- Lleva siempre contigo el cuaderno de notas.
- Si olvidas registrar, haz una estimación en cuanto lo recuerdes.
- Utiliza un cronómetro para las interrupciones.
- Resume tu tiempo puntualmente. [HUMPHREY, 2001]

1.12.5 La Gestión de compromisos.

- ¿Qué se hará?
- Criterios para determinar lo que está hecho.
- ¿Quien lo hará?
- ¿Cuando se hará? [HUMPHREY, 2001]

1.12.6 Pasos para gestionar un compromiso.

- Hacer una lista de los compromisos actuales.
- Incluir Qué y para Cuando es el nuevo compromiso.
- Incluir estimación de cuanto trabajo te supondrá cada compromiso. [HUMPHREY, 2001]

1.12.7 Tips para la gestión de compromisos.

- Detectar si te estas retrasando
- Intentarlo más duramente no ayudará
- Define en que lugar del proyecto estas y que tiempo te queda
- No confíes en la suerte
- Estimaciones erróneas = estimaciones bajas
- Cambio, implica trabajo [HUMPHREY, 2001]

El PSP no es más que un motor pequeño que guía a cada uno con vista a realizar un trabajo más rápido, óptimo y con posibilidad de pocos errores, pero de manera personal, para lo que después viniese siendo el motor mayor el Proceso de Software en Equipo *(TSP)* con una mayor claridad y compenetración, lo que arrojara en un futuro mejores resultados.

1.13 Análisis de la plataforma Microsoft Visual Studio 2005.

Una vez expuestas algunas de las características que debe de tener un sistema para su calidad y eficiencia, y de esta manera proponer ya una posible solucionar a nuestro problema, creo que debemos mencionar el uso de las plataformas para desarrollar sistemas, ya que sin ellas tampoco seria posible la implementación de aplicaciones.
En la actualidad existen varias plataformas para el desarrollo de software las cuales poseen diferentes características y con diferentes potencialidades después de hacer un estudio de las mismas llegamos a la conclusión y es a la que me quiero referir, la plataforma utilizada en la solución de nuestro problema será Microsoft Visual Studio 2005 dentro de ella utilizaremos el lenguaje de programación Microsoft Visual C #.
Con Visual Studio 2005, Microsoft incorpora nuevas características y tecnologías que admiten el desarrollo de aplicaciones en todas las fases del ciclo de vida del software, desde su desarrollo hasta la fase de implantación, permitiendo una completa plataforma de desarrollo que satisfaga a los desarrolladores en todos los niveles. [IZQUIERDO, 2007]

También proporciona nuevas y avanzadas características para los desarrolladores de software. Con herramientas de calidad, desarrollo y diseño de aplicación altamente integradas, así como metodología de proceso de software personalizable. Ofrece a los equipos de desarrollo las herramientas y los procesos que les ayudarán a predecir que un proyecto será exitoso, a ganar en eficiencia organizacional y a reducir los costes totales de desarrollo. [IZQUIERDO, 2007]

Visual Studio es una plataforma multilenguaje, proporciona la (POO), es compatible con otras aplicaciones, proporciona una gran cantidad de librerías y funciones que no ahorra trabajo y por tanto cantidad de código, lo que hace el sistema más rápido y óptimo, algunos lo catalogan como la

integración de lo mejor de otros lenguajes. Aunque debemos de reconocer que se necesita una máquina potente para que pueda correr sin problemas y el trabajo no se haga lento y tedioso. Por tanto trabajaremos sobre esta plataforma utilizando el Microsoft Visual C# como lenguaje de programación debido a las potencialidades que este nos brinda.

1.14 Conclusiones.

Se hace referencia de esta manera a las características principales de la programación hoy en día, la importancia que esta jugando para el desarrollo de aplicaciones la programación orientada a objetos POO por sus grandes ventajas, el papel que juega la buena selección de la plataforma de trabajo, el uso del PSP y sus tablas, la utilización de patrones correctamente, etc.

Si se tiene en cuenta todos estos detalles mencionados anteriormente, se creará una base más sólida y factible para lograr el éxito. Ahora bien para ello se debe contar con algo muy importante que hasta ahora no se ha visto y es la propuesta de solución al problema dado, se debe tener convicciones clara de lo que se quiere y como se quiere para llegar a realizar una buena propuesta que satisfaga y erradique el problema planteado.

Capítulo 2: Descripción y análisis de la solución propuesta.

2.1 Introducción.

En este capítulo se dará una panorámica de cómo se dio respuesta al problema, la solución propuesta, para ello se abordará temas, sin los cuales no hubiese sido posible la realización del sistema, se hablará de la arquitectura, sus estilos, se hará una valoración critica del diseño propuesto por al analista paso fundamental para el cumplimiento de los requisitos de forma correcta.

También tratará sobre las estrategias de integración que conforman nuestro sistema y la importancia de las mismas para mayor organización y rapidez de la aplicación. Se Menciona además los patrones que tuvimos en cuenta a la hora de programar el sistema.

Como detalle importante se debe hablar de las Capas que fueron definidas por el arquitecto dentro de estas, capa de Presentación la cual brinda interfaces que le permiten al usuario intercambiar con el sistema, y la capa de Negocio que contiene clases controladoras encargadas de manejar toda la lógica del negocio de la aplicación. Se dará a conocer también algunos de los algoritmos y estructuras de datos que fueron utilizados en nuestro sistema, así como una descripción de las clases u operaciones que nos fueron útiles para dar cumplimiento a nuestros objetivos, y como aspecto fundamental abordará sobre el Proceso de Software Personal PSP y sus tablas.

2.2 Estilo Arquitectónico.

Existen varios lugares donde la arquitectura esta presente, aquí se verá en el desarrollo de aplicaciones informáticas, sin lugar a duda lo que la convierte en una herramienta de puntería para la organización del trabajo, partiendo de esto podemos decir que hay diferentes caminos por lo cual guiar un trabajo o un proyecto etc. Para el desarrollo de aplicaciones hay creados diferentes estilos arquitectónicos los cuales solucionan un problema, en dependencia de las características del mismo, ejemplo de esto tenemos el

modelo vista-controlador, arquitectura en capas entre otros. Por tanto es de vital importancia una selección óptima del mismo. [CAMEJO, 2007]

La Figura 1 muestra la vista lógica de la arquitectura en capas definido por el arquitecto. [CAMEJO, 2007]

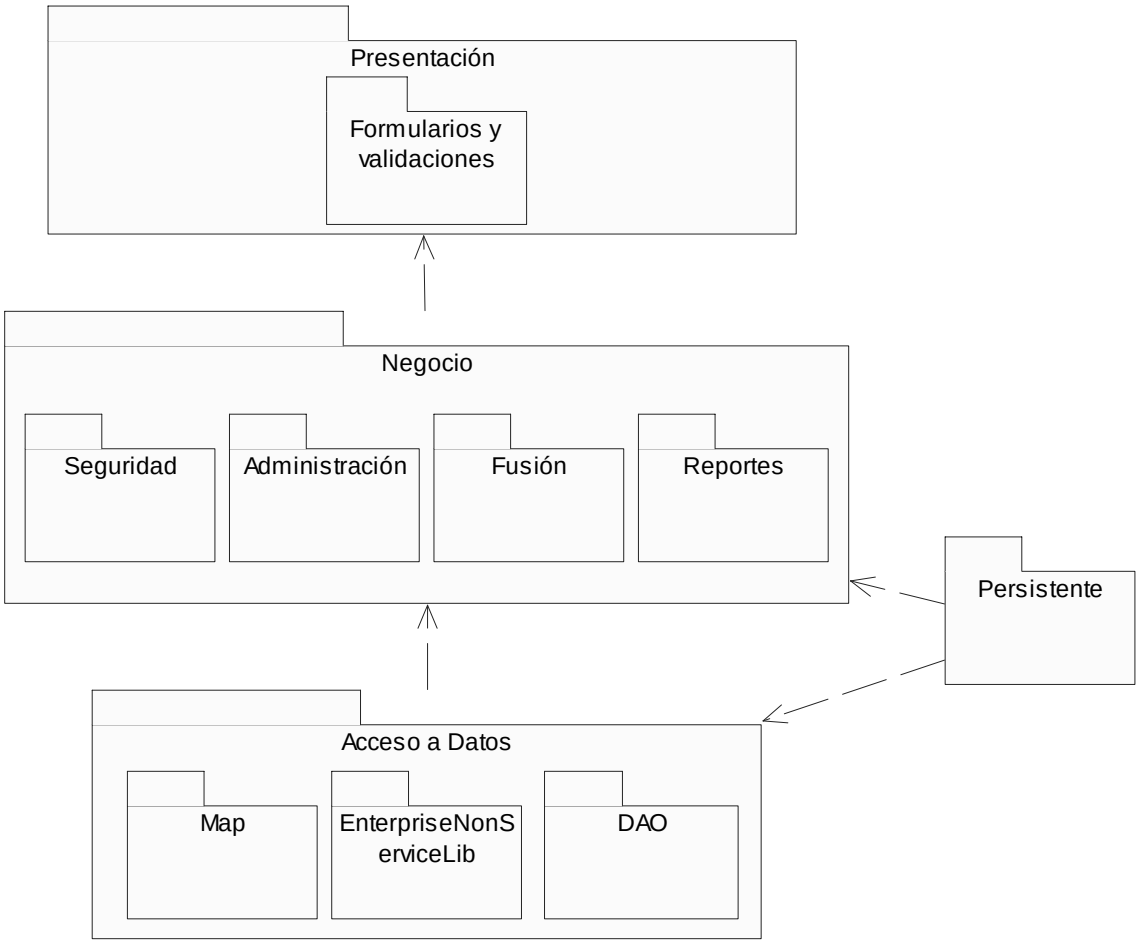


Figura 1: Vista lógica de la arquitectura

Como se observa es una arquitectura por capa, la cual nos brinda una mayor organización y eficiencia del trabajo. Dentro de la capa de Presentación encontramos las interfaces y clases que nos permiten insertar

y validar los datos introducidos por los usuarios, la cual hace llamadas a la capa de Negocio que es la que controla con sus clases y métodos todo el negocio del sistema y esta a través de la capa de Acceso a datos adquieren o salvan la información necesaria en la base de datos, para dar respuestas a las peticiones de los usuarios cuando interactúan con el sistema, según sus respectivos privilegios.

2.3 Valoración crítica del diseño propuesto por el diseñador.

Tanto como el arquitecto, el diseñador y su propuesta de diseño son iguales de importantes, pues esto da una idea de la cantidad de clases que existen, las relaciones entre ellas y los métodos que darán respuestas a cada uno de los casos de usos que contempla el sistema.

Se pretende hacer un análisis del diseño planteado por el diseñador, debemos decir que el sistema es sencillo, contiene un solo módulo ***IPC***, el cual esta dividido en varios subsistemas.

La Figura 2 muestra los subsistemas que conforman el módulo ***IPC***. [LUIS, 2007]

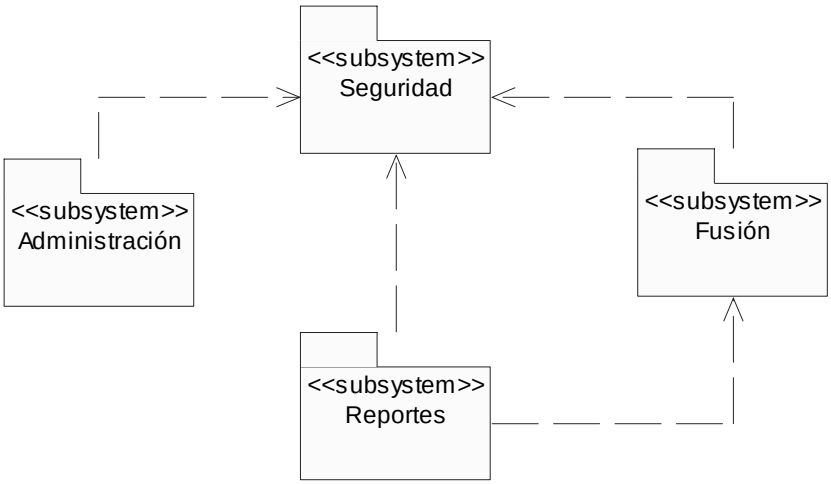


Figura 2: Diagrama de subsistemas del módulo *IPC*

La cual nos da idea de que el sistema cuenta con seguridad y que los demás subsistemas necesitan de este para poder realizar su función. Lo cual es factible desde el punto de vista lógico.

Para realizar cualquiera operación ya sea para emitir algún reporte o de administración (Gestionar canasta de productos), se necesita estar previamente autenticado y con los permisos concedidos, por tanto la existencia de jerarquías en los subsistemas. Así como la dependencia del subsistema Reportes con el subsistema Fusión es obvio que no se puede llegar a ningún reporte sin antes el sistema haber realizado al menos una fusión.

Ahora dentro de cada subsistema existen un conjunto de operaciones definidas por el diseñador en sus diagramas de clases, que dan solución a los casos de usos que plantea el analista en su modelo de negocio dando especificaciones de cada uno de ellos, estos son los llamados requisitos funcionales.

- Autenticarse
- Gestionar Usuario
- Fusionar Datos
- Emitir Fusión
- Emitir Tabla de Precios
- Calcular Índices de Precios
- Gestionar Canasta de Productos [RÍOS, 2007]

De estos casos de usos existen algunos que son más significativos como son el caso de Autenticarse, sin este no se podría acceder al sistema, Fusionar Datos sin el cual no seria posible procesar ningún tipo de información, ni realizar ningún calculo de índices, y por último tenemos Calcular Índices de Precios que es en fin el objetivo de nuestra aplicación. Para el correcto funcionamiento de estos casos de usos el analista ha propuesto un número de clases y métodos los cuales permiten realizar el trabajo y me gustaría detenerme en el cual puede ser más importante Calcular Índices de Precios, para lograr este importante requisito se tiene en cuenta una encuesta que se realizó en el año 1999, esta encuesta contiene por tipo de mercado el precio de los productos en aquel entonces, con estos valores se calculan índices de variaciones con respecto a los precios de ahora y me gustaría señalar que en el diseño no se tiene en cuenta una posible variación de esta encuesta que si bien hasta ahora no ha cambiado, podría llegar a hacerlo, y al sistema no contar con este funcionamiento llegaría a perder la funcionalidad de calcular las variaciones de nuevos índices con respecto a los precios de una posible nueva encuesta (año base), solo lo hará con respecto al mes anterior, se debió tener en cuenta este pequeño inconveniente, aunque

debemos reconocer que el sistema da solución a los requisitos funcionales. Por tanto para nuevas iteraciones se tendrá muy en cuenta el caso de uso Modificar Encuesta.

2.4 Estrategias de integración.

Como describimos previamente nuestra aplicación esta definida por capas, para ser exacto, tres capas conforman nuestro sistema, capa de Presentación, Negocio y Acceso a Datos, ahora bien planteamos también que mediante esta arquitectura el trabajo es más seguro, rápido y eficiente pues antes se programaba de una manera engorrosa todo detrás del botón lo que hace una aplicación mas lenta e insegura, ahora, ¿cómo trabajan estas tres capas?, la integración de las mismas es a través de instancias que se le hacen a las clases dentro de su misma capa y en la capa inmediata inferior, cumpliendo de esta manera con el paradigma de la arquitectura en capas, creando objetos que son en fin los que se comunican entre las capas, llevando la petición que le hace una capa a la otra en su orden jerárquico, mediante la capa de Presentación se hacen solicitudes ingresadas por los usuarios, y mediante de la manipulación de objetos, se lleva el pedido a la capa de Negocio, esta última con sus operaciones y clases procesa el pedido y mediante los mismos se comunica con la capa de acceso a dato siendo esta la que maneja la información de la base de datos, y devuelve una respuesta por el mismo camino de manera que la capa de Presentación no se comunicara directamente con la capa de Acceso a Dato sin antes pasar por la de Negocio que con sus clases controladoras es la encargada de manejar toda la lógica del negocio de forma correcta, evitando así fallas en el sistema. Garantizando así una total integración de las capas en el sistema.

- La Figura 3 muestra de manera sencilla un ejemplo de lo expuesto anteriormente, como damos solución a la petición de ver la fusión realizada antes de ser almacenada mostrándose en un componente llamado TreeView, haciendo uso de las operaciones que están en la capa de negocio a través de sus objetos, como es el caso del objeto llamado fusionadora que es del tipo CFusion, y el uso de la interfaz CConversionNomencladores para obtener los nombres dado los códigos respectivos, cosa que permita al usuario ver la lista de provincias con sus respectivos mercados, productos y precios. Instancias creadas en la capa de Presentación.

```
        private void LLenarVistaPrevia()
        {
            FusionPreview.Nodes.Clear();
            int cantidadProductos = 0;
            int cantidadMercados = 0;
            int cantidadProvincias = Fucionadora.ObtenerProvincias().Count;

            for (int i = 0; i < cantidadProvincias; i++)
            {
                cantidadMercados = Fucionadora.ObtenerProvincias()[i].ObtenerMercados().Count;
                TreeNode[] tempMercados = new TreeNode[cantidadMercados];

                for (int j = 0; j < cantidadMercados; j++)
                {
                    cantidadProductos = Fucionadora.ObtenerProvincias()[i].ObtenerMercados()[j].ObtenerProductos().C
                    TreeNode[] tempProductos = new TreeNode[cantidadProductos];
                    //llenar productos
                    for (int k = 0; k < cantidadProductos; k++)
                    {
                        string texto= CConversionNomencladores.NombreProductoCodigo(Fucionadora.Provincias[i].Mercad
                        tempProductos[k] = new TreeNode(texto);
                        tempProductos[k].Nodes.Add(new TreeNode(Fucionadora.ObtenerProvincias()[i].ObtenerMercados()
                    }
                    tempMercados[j] = new TreeNode(CConversionNomencladores.MercadoPorCodigo(Fucionadora.ObtenerProv
                    tempMercados[j].Nodes.AddRange(tempProductos);
                }
                TreeNode tempProvincia = new TreeNode( CConversionNomencladores.ConvertirProvincia(Fucionadora.Obten
                tempProvincia.Nodes.AddRange(tempMercados);
                FusionPreview.Nodes.Add(tempProvincia);
            }
        }
```

Figura 3: Método LlenarVistaPrevia

2.5 Patrones Utilizados.

De manera simple un patrón es un par problema/solución que se puede aplicar en nuevos contextos, sabiendo como aplicarlos, esto último es importante tenerlo en cuenta ya que un patrón aplicado indebidamente puede causar grandes trabas en la obtención de un producto de calidad. Para un módulo sencillo como el nuestro no es necesario aplicar una gran cantidad de patrones, por tanto se aplicaron solo algunos, patrones Grasp siendo estos más generales y dan una idea a la hora de comenzar el diseño de un sistema. Nuestra aplicación se basó en la poca dependencia de clases y en la creación de objetos de una clase siempre y cuando cumpla con el paradigma del patrón Creador. [LARMAN, 2004]

- **Bajo Acoplamiento.** Debe haber pocas dependencias entre las clases.

Creador. Se asigna la responsabilidad de que una clase B cree un Objeto de la clase A solamente cuando.

1. B contiene a A.
2. B es una agregación (o composición) de A
3. B almacena a A
4. B tiene los datos de inicialización de A (datos que requiere su constructor)
5. B usa a A.

La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización.

Otro grupo de patrones son los conocidos patrones GoF dentro de estos queremos hacer mención a dos de ellos, patrón Singlenton y Facade, en el caso del primero su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella, restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. [WELICKI, 2007]

- La Figura 4 muestra un ejemplo de cómo se utiliza el patrón Singlenton.

```
using System.Windows.Forms;
using ONE.Clases;


namespace ONE
{
    public partial class Form1 : Form
    {

        Fusion.CFicheroParse Parser = new ONE.Fusion.CFicheroParse();
        Clases.CFusion Fucionadora = new ONE.Clases.CFusion();
        FrmProgreso progreso = new FrmProgreso();

        public Form1()
        {
            InitializeComponent();
            InicializarComboFechas();

        }
```

Figura 4: Uso del patrón Singlenton

El objeto de tipo CFusión llamado fusionadora aquí tiene un punto de acceso global, al igual que el objeto del tipo CficheroParse llamado Parser.

Para el caso del patrón Facade (fachada) como su nombre lo indica, es una interfaz la cual actúa de mediación, entre dos capas, en nuestro sistema esta interfaz actúa entre la capa de Negocio y Acceso a Datos, logrando obtener los nombres de productos, mercados y provincias, de la base de datos según el código enviado de la capa de Presentación a la de Negocio, si algún día esto cambia pues no es necesario hacer cambio en ninguna de las capas solo en la interfaz, posibilidad que brinda dicho patrón.

2.6 Capa de Presentación.

Anteriormente se hizo referencia a la capa de Presentación, se dice que es la intermediaria entre el usuario y la aplicación, es el frente del sistema, es quien va a comunicarse directamente con el usuario,

por tanto debe presentar una buena calidad, entendimiento, debe de cumplir con ciertas normas estándar, esta capa es la encargada de darle la posibilidad al usuario de insertar o obtener información deseada, el usuario debe estar previamente autenticado y esto debe ser nuestro primer paso, la aplicación una vez iniciada deberá de pedirle al usuario registrarse, manteniendo todas las demás funcionalidades del sistema inhabilitadas, cosa que permitirá la seguridad del sistema así como la integridad de los datos, a través de componentes que nos brinda la plataforma Visual Studio 2005 en particular el lenguaje de programación Microsoft Visual C#, el usuario introducirá sus datos, de esta forma comprobar si el usuario existe y asignarle su privilegio en el sistema, sabemos de ante mano que el usuario con privilegio de operador no tendrá acceso a algunas funcionalidades tales como gestionar usuario que no es mas que la que permite incorporar nuevos usuarios, modificar o eliminar alguno ya existente, solo el usuario con privilegio de administrador tendrá acceso completo al sistema, a todas sus funcionalidades. Para lograr una interfaz con un entorno amigable, sencilla, que le permitiera al usuario una buena y rápida familiarización con la misma. Utilizamos un componente que no brinda la plataforma anteriormente mencionada, este componente es llamado TabControl el cual contiene una colección de TabPages fáciles de manejar, permitiendo que la aplicación trabaje con una sola forma, para no sacar de paso al usuario con tantas ventanas, todos los TabPages tiene la misma dimensión y estructura, lo que lógicamente da la medida de la estandarización de la misma, cada TabPages corresponde a una funcionalidad del sistema que en dependencia del rol será o no activada.

Se dará una breve descripción de las interfaces de usuario que contiene la capa de Presentación.

- La Figura 5 muestra la interfaz Autenticarse.

Figura 5: Interfaz Autenticarse

El usuario solo deberá introducir sus datos y hará clic en el botón entrar para acceder al sistema, como se explica anteriormente este es el primer paso una vez inicializado el sistema, se deberá tener en cuenta que la contraseña es algo privado y se tendrá en cuenta algoritmos de encriptación para la misma.
Ahora se planteó que en dependencia del rol del usuario que accede al sistema, tendrá o no acceso a determinadas funcionalidades.

- La Figura 6 muestra la interfaz para un usuario con rol de administrador.

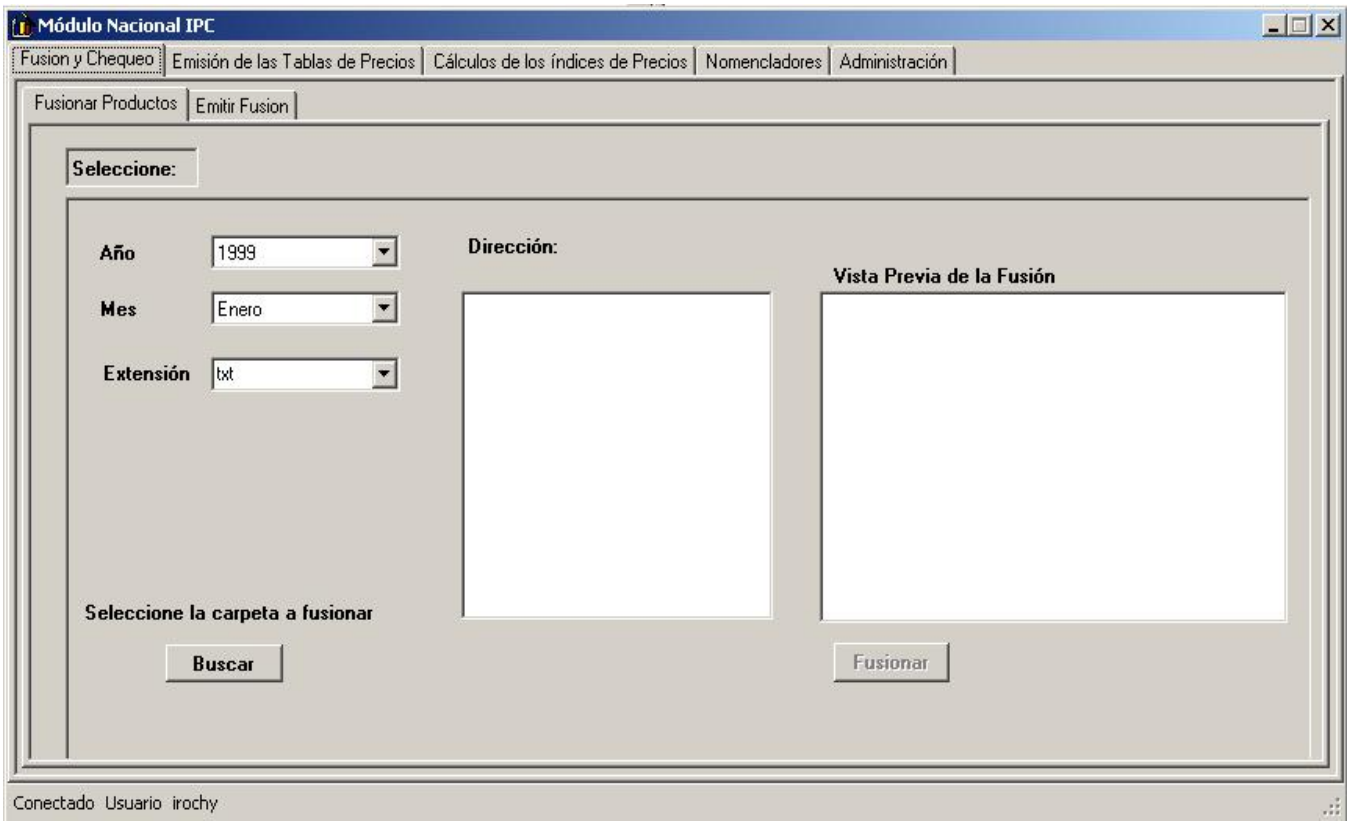


Figura 6: Interfaz para usuarios con rol de administrador

Se aprecia los casos de usos Nomencladores y Administración contenidos en los dos últimos TabPages respectivamente, dichos casos de usos son los encargados de adicionar, modificar o eliminar tanto usuarios como productos, por lo cual son funcionalidades que solo serán accedidos por administradores del sistema.

- La Figura 7 muestra la interfaz para un usuario con rol de operador.

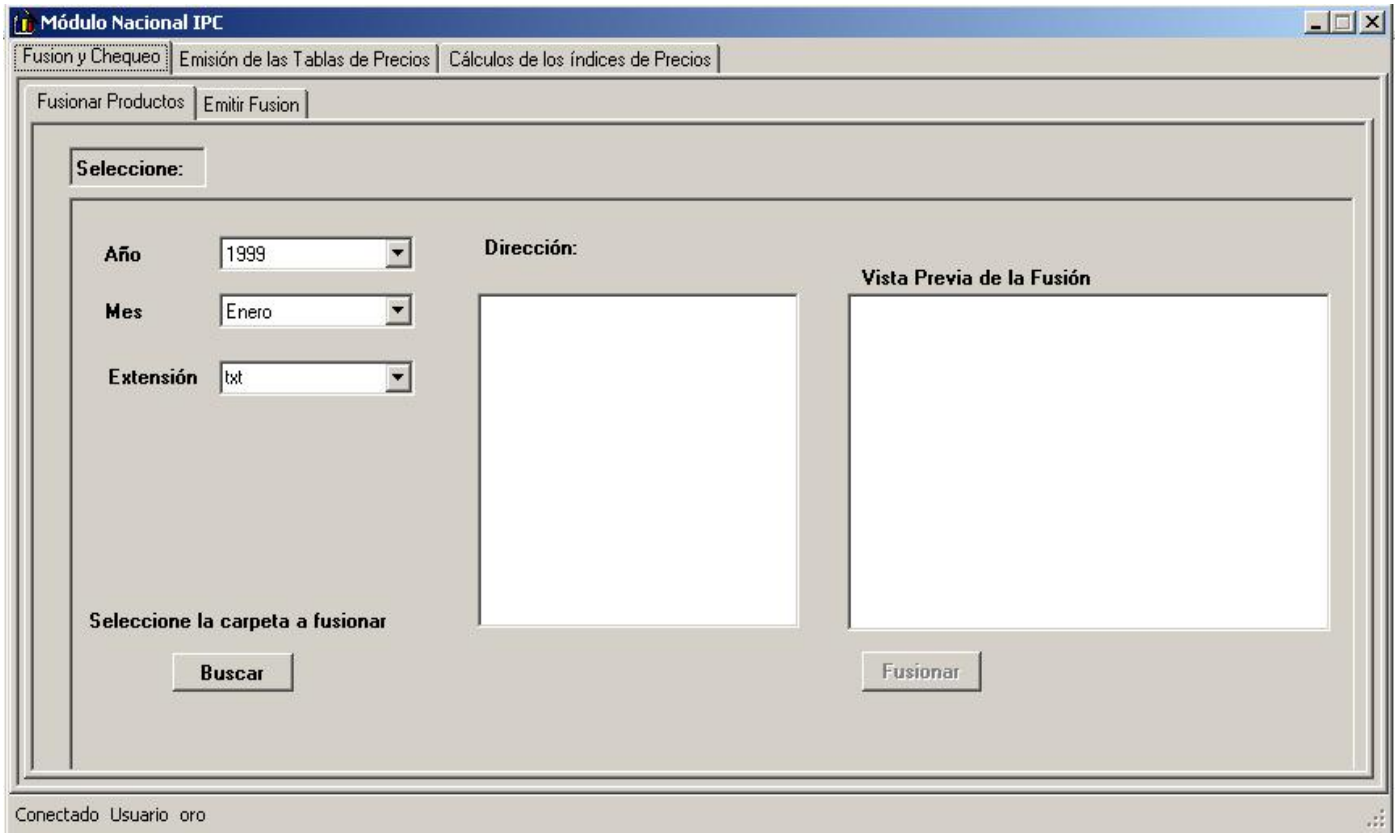


Figura 7: Interfaz para usuarios con rol de operador

Se aprecia que los casos de usos Nomencladores y Administración mostrados anteriormente, para este usuario con rol de operador no se encuentran disponibles. Garantizando la seguridad del sistema.

- La Figura 8 muestra la interfaz Gestionar Usuario para insertar un nuevo usuario al sistema.

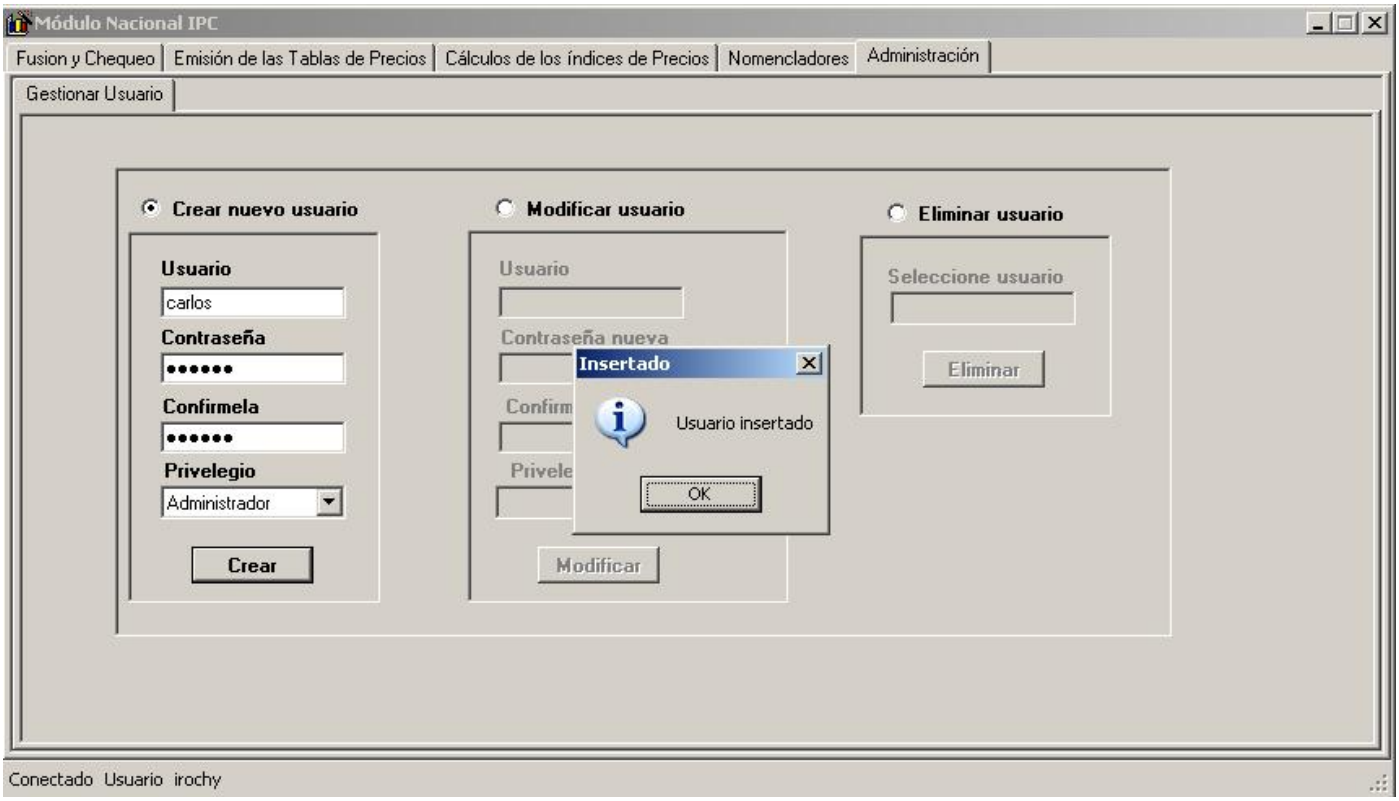


Figura 8: Interfaz para insertar nuevos usuarios

Un administrador del sistema tendrá la posibilidad de agregar nuevos usuarios al sistema, teniendo en cuenta el nombre de usuario, contraseña, y su privilegio, para esto utilizamos los componentes TexBox y ComboBox, debemos de tener en cuenta aquí que las contraseñas deben ser iguales en el caso de la contraseña y la confirmación de la misma, es una validación que se debe tener presente a la hora de almacenar la contraseña del nuevo usuario para evitar posibles errores, e igual para el nombre de usuario no podrá existir números. Para acceder a las otras funcionalidades se utilizó el componente RadioButton.

- La Figura 9 muestra la interfaz Gestionar Usuario para Modificar un usuario.

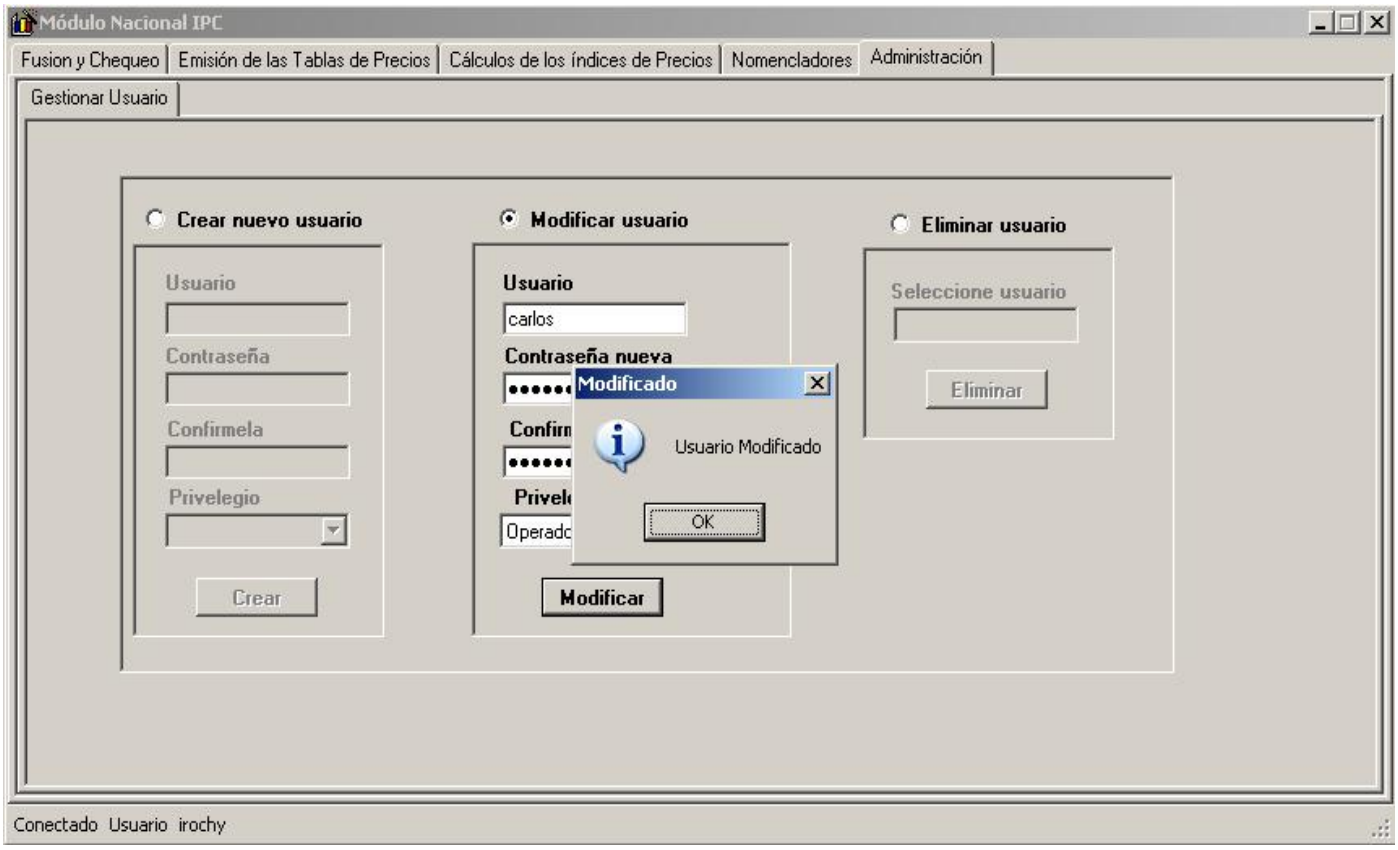


Figura 9: Interfaz para modificar usuarios

Esta funcionalidad permite dar la posibilidad a un usuario de modificar sus características, donde previamente se deberá introducir en el TextBox, su login, es decir nombre de usuario, luego modificar sus atributos teniendo en cuenta también que debe de coincidir la contraseña, comprobar además que el usuario introducido debe ya estar almacenado en la base de datos sino el sistema deberá lanzar un mensaje de error.

La Figura 10 muestra la interfaz Gestionar Usuario para Eliminar un usuario.

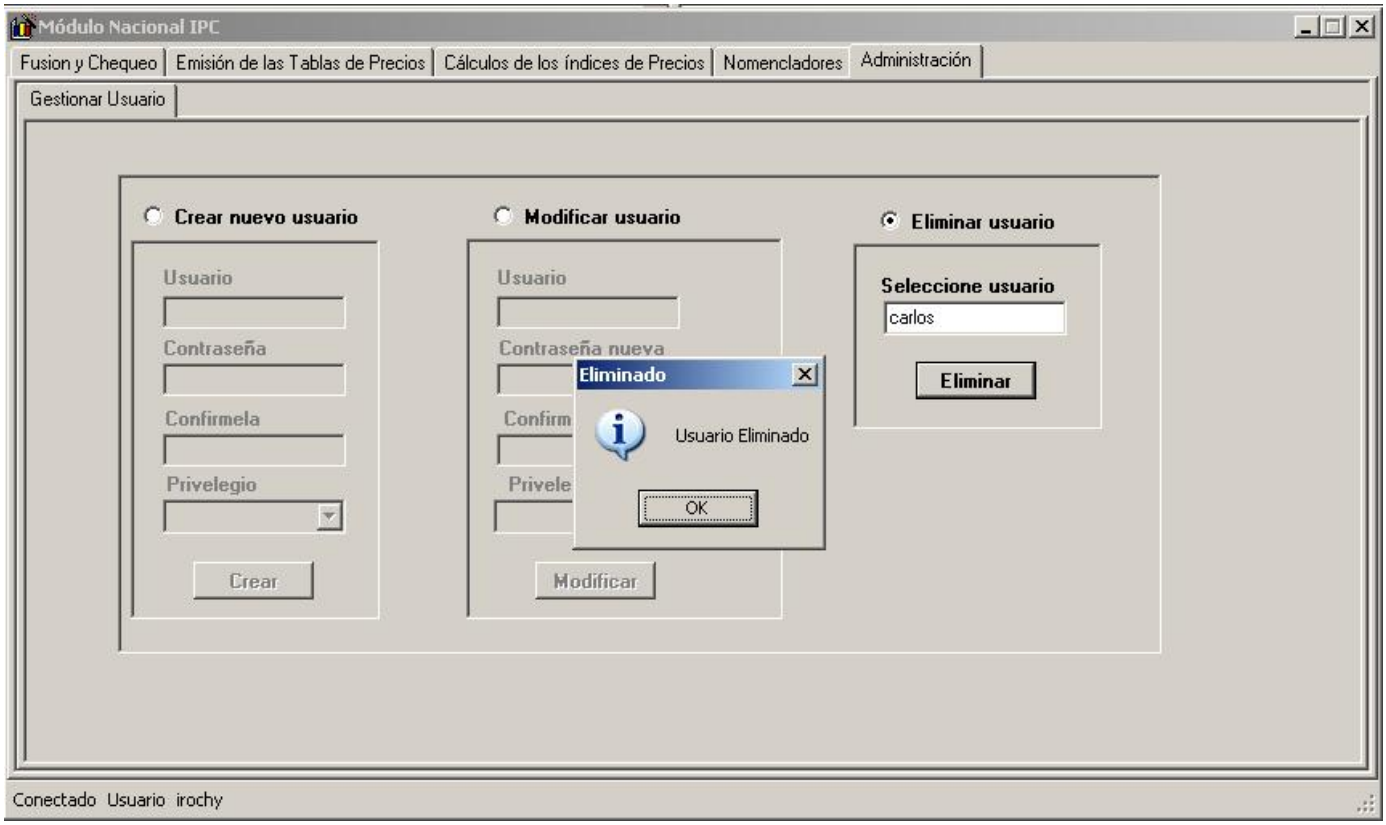


Figura 10: Interfaz para eliminar usuarios

Y como toda aplicación debe garantizar poder eliminar usuarios, en caso de ser necesario eliminar aquella persona que por distintas razones, ya no trabajará más con el sistema, aquí al igual que en la anterior comprobar que el usuario a eliminar exista en nuestra base de datos, sino el sistema lanzará un mensaje de error.

- La Figura 11 muestra la interfaz Fusionar datos.

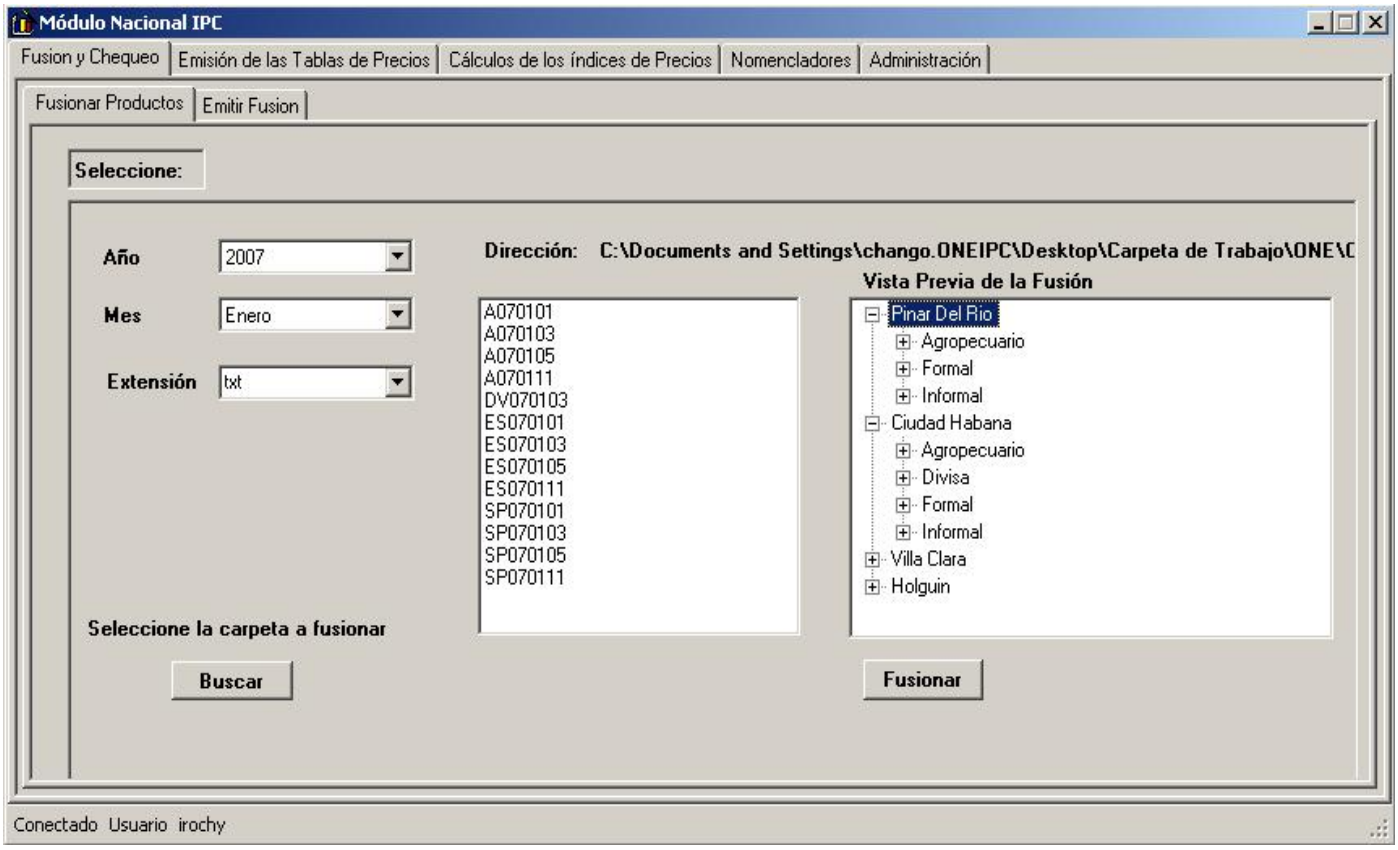


Figura 11: Interfaz para fusionar los datos

Esta es una de las funcionalidades significativas, en ella el usuario selecciona una fecha y busca los ficheros a fusionar para ello se utilizó el componente OpenFileDialog que permite examinar la PC, para en este caso seleccionar los ficheros que se van a fusionar, además permite hacer filtros para la extensión de los ficheros a buscar, ejemplo xls, txt, doc, etc. Selecciona la dirección URL, el nombre y la extensión de los ficheros. Además utilizamos el componente ComboBox para el caso de seleccionar la fecha. Una vez concluidas estas operaciones, mostrará en un ListBox automáticamente los ficheros que son válidos por la fecha y extensión definidas por el usuario, una vez el usuario comprueba que todo esta en orden que son realmente los ficheros que desea fusionar pues se le emitirá una vista previa de la fusión es decir se le mostrará por cada provincia sus respectivos mercados con sus productos y precios, para esto utilizamos el componente TreeView. El próximo paso a dar pues será dar clic al botón fusionar encargado de almacenar dicha fusión en la base de datos.

- La Figura 12 muestra la interfaz Emitir Fusión.

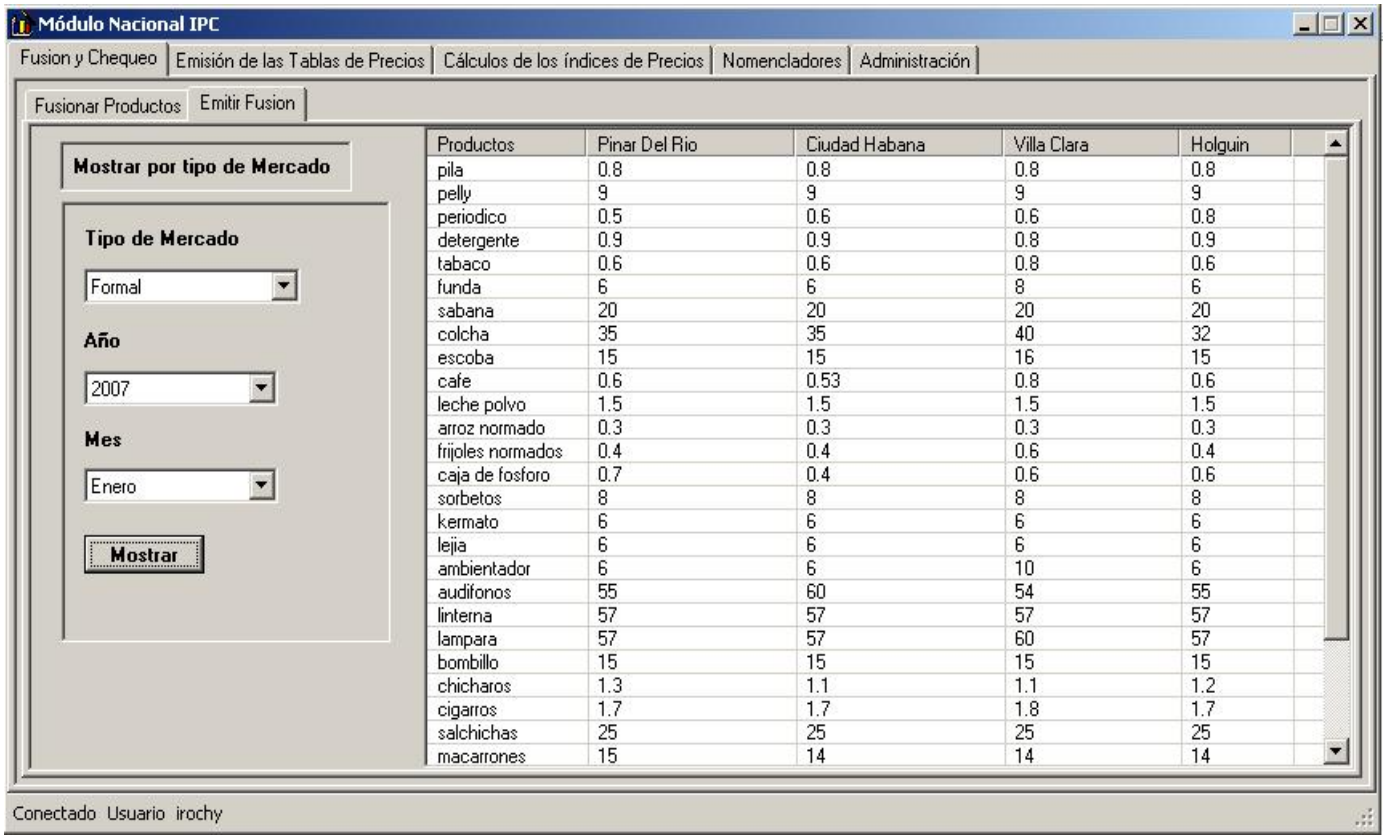


Figura 12: Interfaz para emitir fusiones

Esta funcionalidad permite al usuario hacer una revisión de la información fusionada, seleccionando la fecha en que desea hacerlo y el tipo de mercado, para este caso se utilizaron algunos componente antes mencionado, ejemplo el ComboBox, ya que el mismo permite una lista con las informaciones necesarias de la cual el usuario define la que desea realmente, y mostrando en el ListView toda la fusión, esta funcionalidad permite hacer una comparación de los precios de los productos en diferentes provincias y llegar a conclusiones.

- La Figura 13 la interfaz Emitir tablas de Precios.

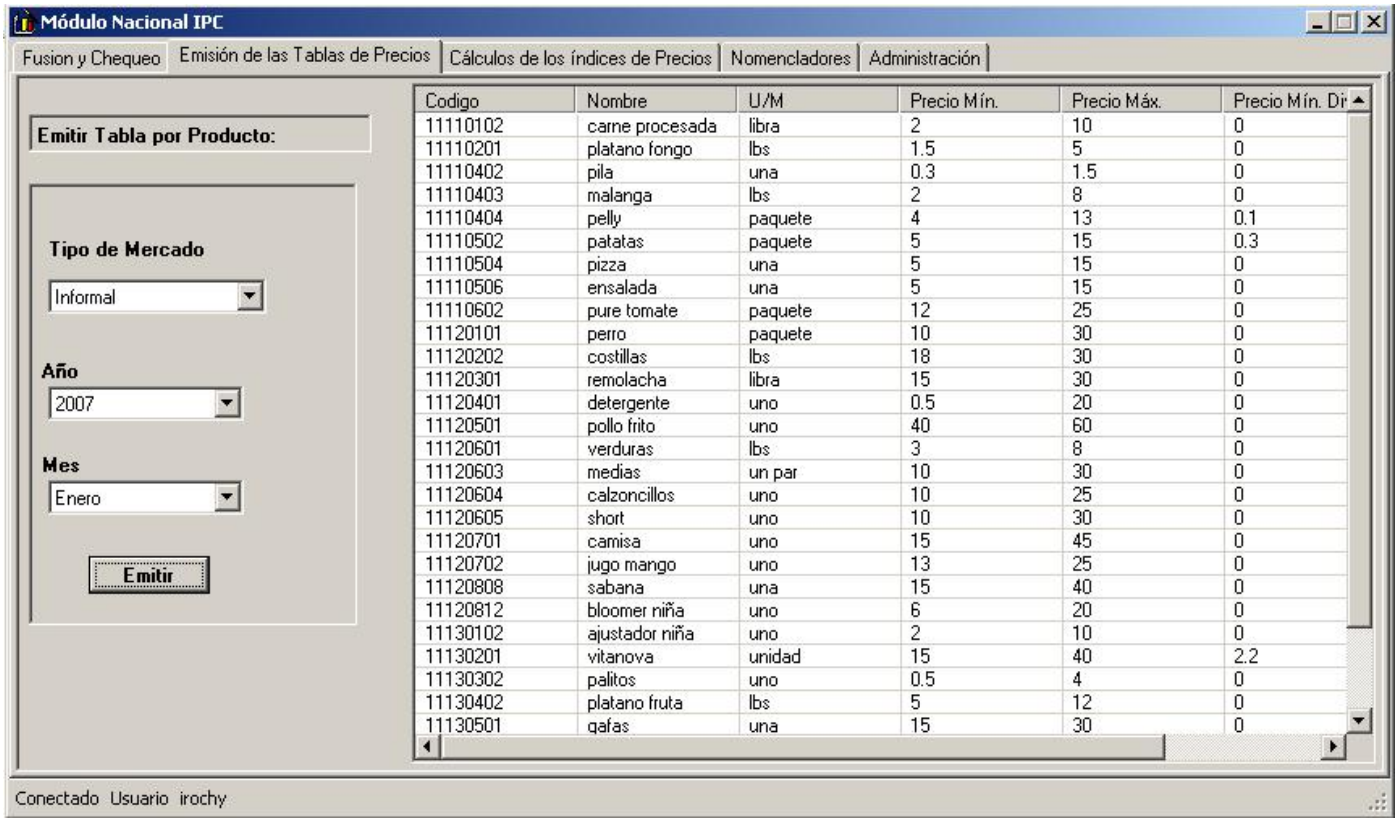


Figura 13: Interfaz para emitir las tablas de precios

Funcionalidad que permite tener una idea de como ondean los precios de los diferentes productos en los diferentes mercados del País en una fecha determinada, componentes utilizados ComboBox para el caso de la fecha, el mercado y el ListView para visualizar la información de utilidad para el usuario.

- La Figura 14 muestra la interfaz Calcular Índices de Precios

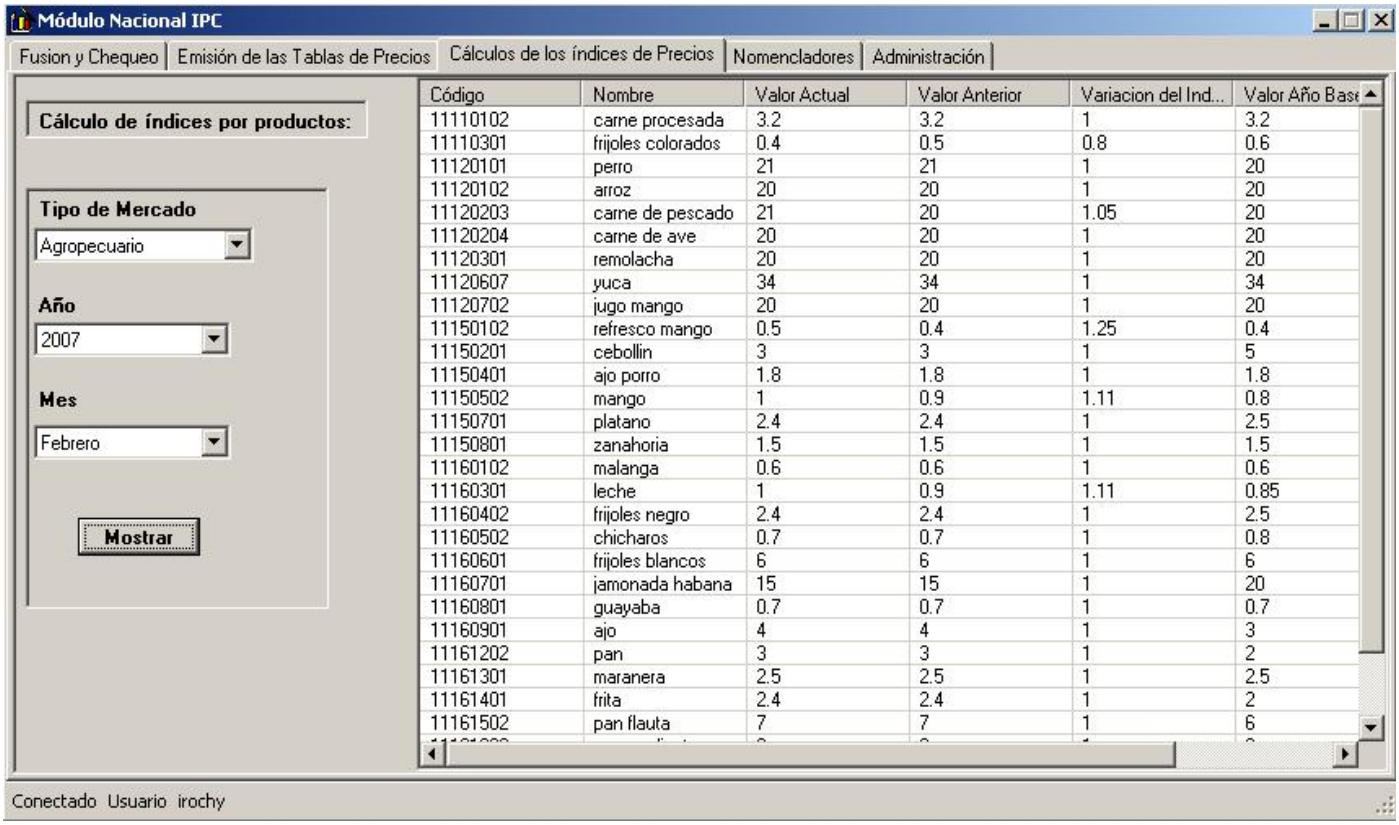
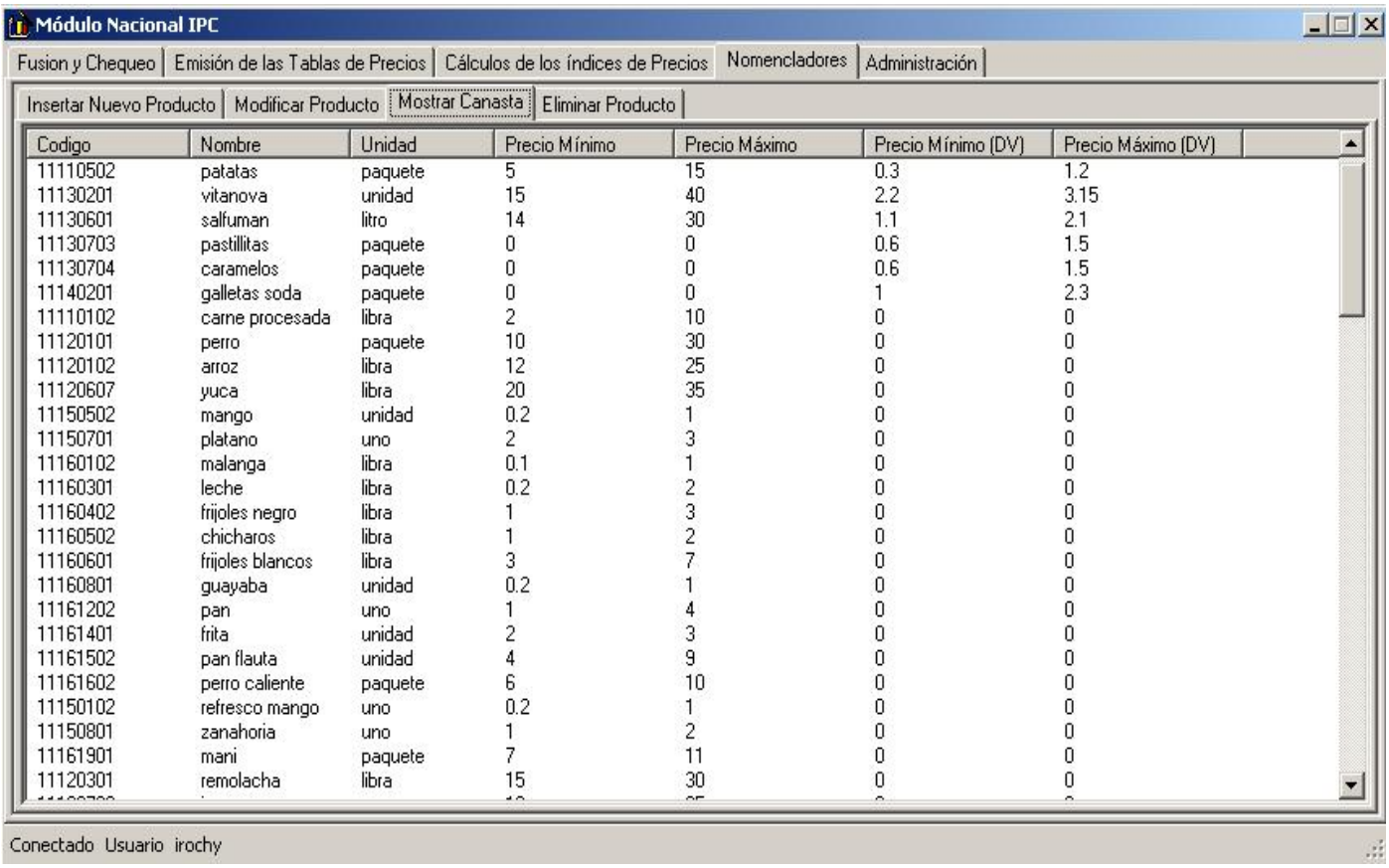


Figura 14: Interfaz para emitir y calcular los índices de precios

Esta es la funcionalidad más importante ya que es en si, es el objetivo de nuestra aplicación, que permite obtener la variación de los índices a nivel de productos con respecto a varias etapas una de ellas es el año base, dando la medida de cuanto han variado estos productos desde la base a la actualidad, y la otra es con respecto al mes anterior. Al igual que la funcionalidad Emisión de Tablas de Precios cuenta con los componentes ComboBox y ListView, seleccionando la fecha y el tipo de mercado.

- La Figura 15 muestra la interfaz Nomencladores, Mostrar Canasta.



Codigo	Nombre	Unidad	Precio Mínimo	Precio Máximo	Precio Mínimo (DV)	Precio Máximo (DV)
11110502	patatas	paquete	5	15	0.3	1.2
11130201	vitanova	unidad	15	40	2.2	3.15
11130601	salfuman	litro	14	30	1.1	2.1
11130703	pastillitas	paquete	0	0	0.6	1.5
11130704	caramelos	paquete	0	0	0.6	1.5
11140201	galletas soda	paquete	0	0	1	2.3
11110102	carne procesada	libra	2	10	0	0
11120101	perro	paquete	10	30	0	0
11120102	arroz	libra	12	25	0	0
11120607	yuca	libra	20	35	0	0
11150502	mango	unidad	0.2	1	0	0
11150701	platano	uno	2	3	0	0
11160102	malanga	libra	0.1	1	0	0
11160301	leche	libra	0.2	2	0	0
11160402	frijoles negro	libra	1	3	0	0
11160502	chicharos	libra	1	2	0	0
11160601	frijoles blancos	libra	3	7	0	0
11160801	guayaba	unidad	0.2	1	0	0
11161202	pan	uno	1	4	0	0
11161401	frita	unidad	2	3	0	0
11161502	pan flauta	unidad	4	9	0	0
11161602	perro caliente	paquete	6	10	0	0
11150102	refresco mango	uno	0.2	1	0	0
11150801	zanahoria	uno	1	2	0	0
11161901	mani	paquete	7	11	0	0
11120301	remolacha	libra	15	30	0	0

Figura 15: Interfaz para mostrar la canasta de productos

Funcionalidad que permite al usuario ver la canasta de productos, se pueden ver todos los productos que existen hasta el momento, así como todos sus atributos, nombres, precios, unidad de medida, etc. Para utilizar esta funcionalidad se debe tener rol de administrador del sistema, se utilizo el componente ListView para mostrar la información.

- La Figura 16 muestra la interfaz de usuario Nomencladores, Insertar Nuevo Producto.

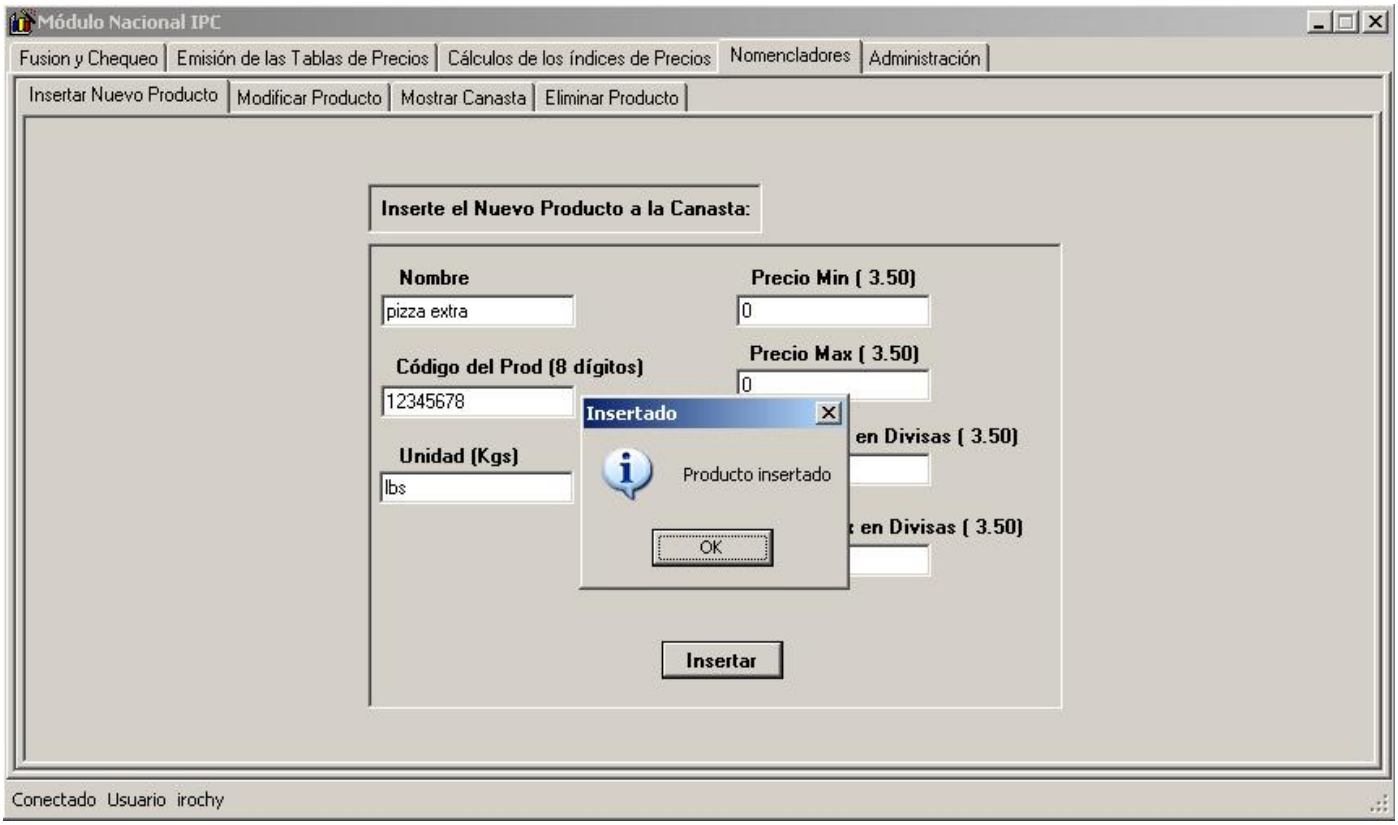


Figura 16: Interfaz para insertar nuevos productos

Esta es una funcionalidad importante ya que el sistema anterior es muy rígido respecto a esto, no inserta nuevos productos, aquí hay que tener en cuenta el código de los productos, deben de ser de 8 dígitos y a la hora de insertar un nuevo producto no puede ser con un código ya utilizado, al igual pasa con el nombre y por tanto darle al usuario una guía de como introducir los datos en dicha interfaz, y validar los datos, utilizamos los componentes, Label para especificarle al usuario como entrar los datos, TextBox para recibir los mismos.

- La Figura 17 muestra la interfaz Nomencladores, Modificar Producto.

Figura 17: Interfaz para modificar productos

Funcionalidad que permite modificar un producto, se escoge el código del producto a modificar y automáticamente se llena los campos vacíos, que previamente serán de solo lectura dando la obligación de escoger primero el producto a modificar, aquí el usuario tendrá la posibilidad de cambiar todos los campos del producto, se utilizó el componente TextBox y el ComboBox para el caso de seleccionar el código del producto ya que brindará una lista de todos los productos, rectificar aquí que en caso del código a entrar y el nombre no podrán ya estar en uso por otro producto.

- La Figura 18 muestra la interfaz Nomencladores, Eliminar Producto.

Figura 18: Interfaz para eliminar productos

Esta funcionalidad permite eliminar un producto, seleccionando el código del producto, una vez hecho esto, los demás campos del producto se llenarán para que el usuario se asegure de que ese producto es el que realmente desea eliminar, hay que tener en cuenta que los demás campos vacíos son de solo lectura, para guiar al usuario que en ellos no podrá modificar nada, propiedad que nos brinda el componente TextBox, Además también utilizamos el componente ComboBox, esta funcionalidad importante no la brinda el sistema existente.

2.7 Capa de Negocio.

Ya se comentó anteriormente la importancia de la capa de Negocio, es la encargada de realizar toda la lógica del sistema a través de sus clases controladoras u operaciones, por tanto se debe tener un

entendimiento claro del funcionamiento del sistema y del diseño de clases realizado por el diseñador, estas deben cumplir con las particularidades de la plataforma de trabajo y con la arquitectura previamente definida por el arquitecto, se debe desglosar el sistema en funcionalidades las cuales corresponden a los requisitos funcionales que darán satisfacción al cliente, para esto, analizamos el funcionamiento desde la etapa de inicio hasta la descripción de análisis, teniendo en cuenta además los patrones de diseño y un buen empleo de los mismos.

Para dar inicio a la lógica del Negocio debemos partir por la funcionalidad de autenticarse, pues ella permite asignar los distintos privilegios que obtendrá el usuario en dependencia de su rol, destacar aquí que el sistema actualmente utilizado en la ONE hoy en día no cuenta con esta operación y debido a esto se han cometido violaciones en los ficheros enviados por provincias, por ejemplo se encontraron ficheros con códigos repetidos siendo estos identificadores únicos de los productos, se han estado realizando cambios en los mismos sin previa autorización, claro que al sistema carecer de seguridad, no hay manera de evitar esto ya que cualquier persona puede manejar toda la información a su antojo, por tanto se nos dio la tarea de evitar estas situaciones, para esto se realizó una clase la cual permitirá determinar permisos dentro del sistema de manera que solo haga cambios aquel usuario destinado a hacerlo.

- Clase CUsuario (contiene login (nombre de usuario), password (contraseña) y rol (permiso dentro del sistema)).

- La Figura 19 muestra el diagrama de clase CUsuario implementado.

Figura 19: Diagrama de clase CUsuario

Particularmente nuestro sistema calcula índices de precios a nivel de productos, estos productos son enviados a través de ficheros desde las provincias los cuales contienen los códigos de los productos con sus respectivos precios, de un determinado mercado, en una fecha dada, (año y mes), por tanto la información quedaría de la siguiente manera, para un año y un mes tendríamos una lista de provincias, con un lista de sus respectivos mercados y en cada mercado sus productos y precios. Por tanto la solución a esta problemática quedo de la forma siguiente. Se creó una clase CProducto con los atributos, que son enviados de provincias (código y precio). Una clase CMercado que contendría además del tipo de mercado, una lista de productos del tipo CProducto. Y lo mismo ocurre con las provincias, se creó una clase CProvincia la cual contiene además del ID de la provincia una lista de mercados y sus productos respectivos. Ahora bien toda esta información es procesada todos los meses, lo que dio lugar a la creación de una nueva clase llamada CFusion la cual contiene en una fecha (año y mes) un listado de todas las provincias que enviaron información de sus productos, entonces de manera jerárquica quedo de la siguiente manera :

- Clase CFusion (contiene fecha, y un listado de provincias).

- Clase CProvincia (contiene el ID, y un listado de mercados).
- Clase CMercado (contiene el tipo, y un listado de productos).
- Clase CProducto (contiene el código y el precio del producto).

- La Figura 20 muestra el diagrama de clases ONE implementado.

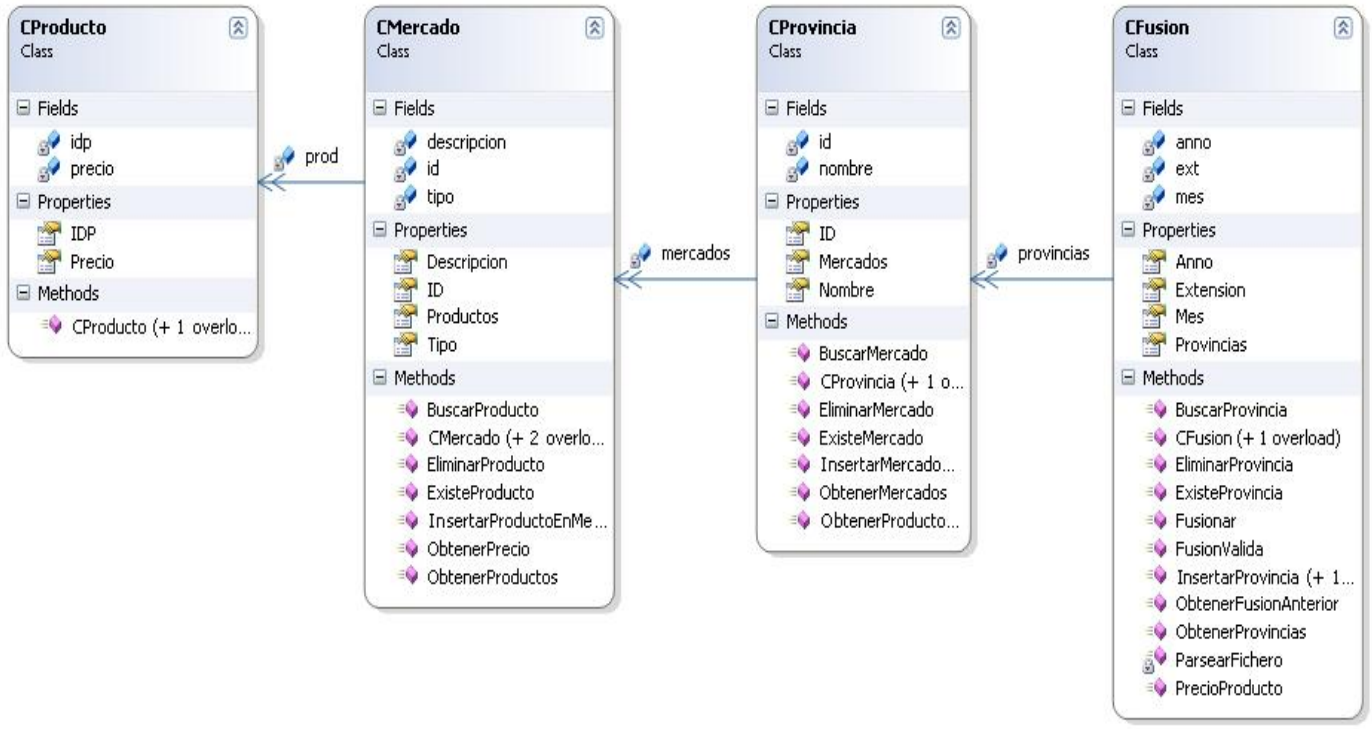


Figura 20: Diagrama de clases ONE

Ahora bien se planteó que parte de esta información proviene de ficheros que son enviados de provincias donde el mismo nombre del fichero en si, nos brinda una gran parte de información, se nos hizo necesario obtener información de los ficheros y por tanto definir nuevas clases las cuales me permitan esta operación, los ficheros son guardados en una carpeta cuando son recibidos de provincias, lo que se convierte en un nuevo problema, porque a la hora de fusionar la información solo debemos seleccionar de toda esa carpeta, solo aquellos ficheros que sean validos por la fecha , y por el formato de nombre que

ellos traen, garantizando no fusionar aquella información que sea irrelevante para el usuario en ese momento, se debe también tener en cuenta la extensión de los ficheros a fusionar siendo flexible en este sentido, ya que en estos momentos los ficheros vienen con una extensión especifica, pero el día de mañana esto puede variar y se tendrá en cuenta darle al usuario la posibilidad de seleccionar el tipo de extensión con la cual el va trabajar en caso de que los ficheros cambien en algún momento, siendo una aplicación extensible, contamos con las siguientes clases:

- Clase CFichero (contiene la dirección URL, extensión y nombre del fichero).
- Clase CFicheroParse (contiene el año, mes, extensión, y la carpeta donde se guardaran todos los ficheros).

- La Figura 21 muestra el diagrama de clases Fusión implementado

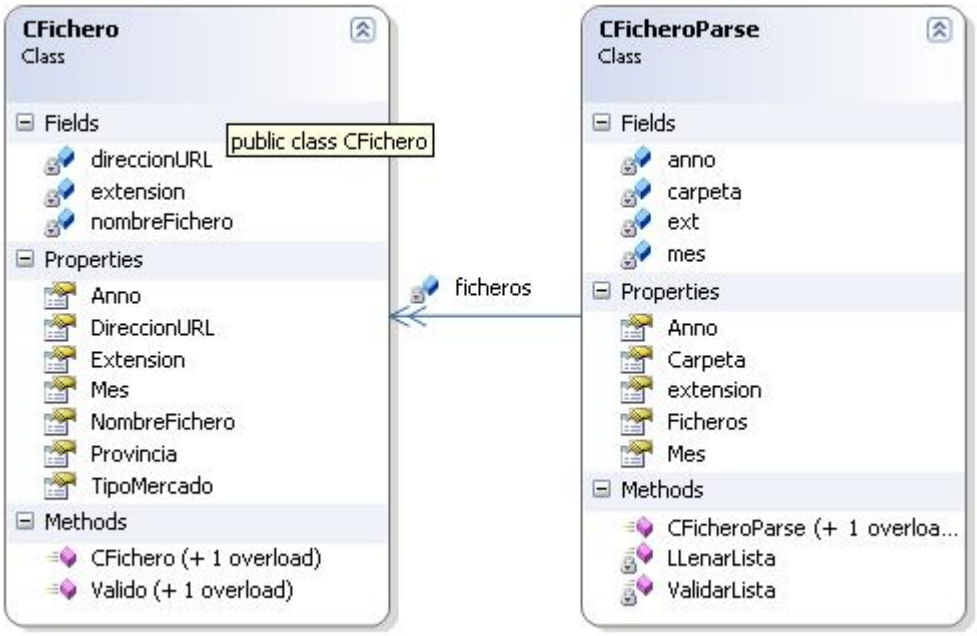


Figura 21: Diagrama de clases Fusión

No solo con la posibilidad que se brinda con la extensión de los ficheros, logramos un producto totalmente extensible y flexible, se conoce que el sistema con el cual trabaja la empresa ONE en la actualidad, es poco flexible, pues no cuenta con la funcionalidad de insertar nuevos productos, ya que a medida que

transcurre el tiempo hay productos que van desapareciendo y otros que se van incorporando, con esto el sistema debe ser capaz de insertar nuevos productos y eliminar algunos ya existentes, siendo ventajoso con respecto al que existe actualmente, ahora bien como funciona dicha operación, para insertar un nuevo producto se debe hacer en la canasta de productos que cuenta la ONE en ella están todos los productos y sus respectivos atributos. Por tanto en las provincias no podrá hacerse esta operación, solo a nivel nacional se decidirá cuando insertar un nuevo producto así como sus atributos, nombre, código, precios mínimo y máximo etc. Por tanto analizando esta situación se decide realizar nuevas clases, ya que la clase CProducto contiene código y precio, pero no es el mismo precio al que se refieren en la canasta de productos, ellas solo coinciden en el código del producto, siendo los demás atributos de ambas clases diferentes, descartando por ahí una posible herencia. De la clase CProducto, dando lugar a otras clases para la solución del problema. Ellas son:

- Clase CProductoCanasta (contiene código, nombre, precio mínimo, precio máximo, precio mínimo en divisa, precio máximo en divisa, unidad).
- Clase CCanastaGeneral (contiene la canasta de productos).

- La Figura 22 muestra el diagrama de clases Canasta implementado.

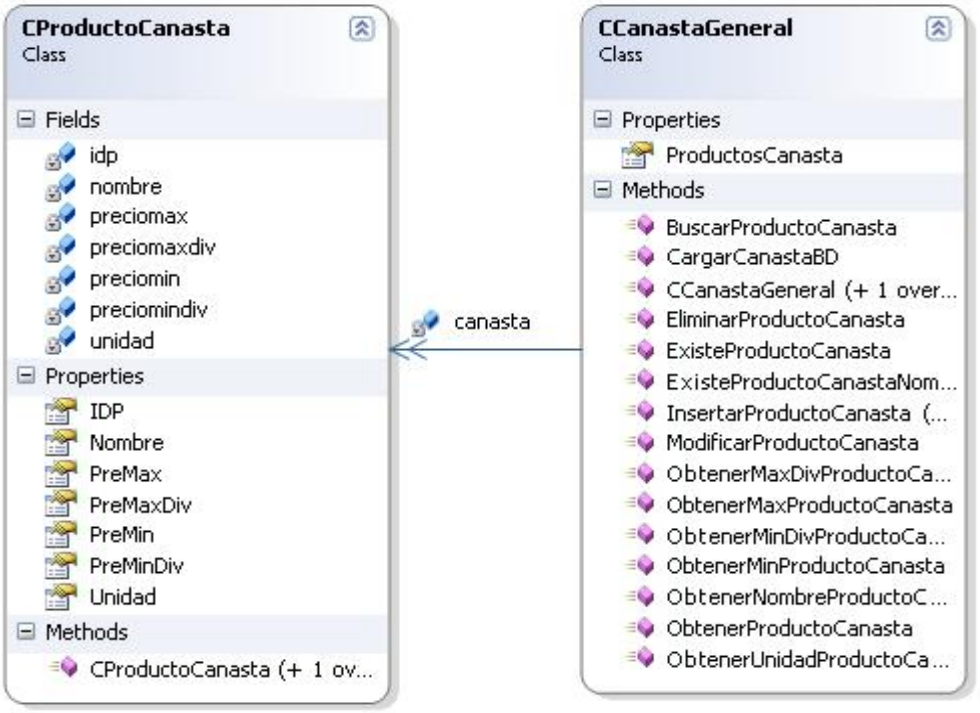


Figura 22: Diagrama de clases Canasta

El cálculo de índices de precios es el principal objetivo, para ello se necesita del precio del producto actual, el precio del mes anterior y el precio del periodo base, para comparar la variación que van teniendo los productos, con respecto al mes anterior, y del periodo base, para eso se requiere de la encuesta realizada en el periodo base, que en ella almacena los precios de los productos por mercados en ese año base, esta también esta contemplada en la ONE, y con características independientes pues contiene los códigos de los productos, los valores de precios de los mismos en todos los tipos de mercados en ese año. Por tanto tampoco utilizamos la herencia, y creamos nuevas clases ellas son:

- Clase CProductoEncuesta (contiene código, precio del mercado Agropecuario, precio del mercado Formal, precio del mercado Informal, precio del mercado en Divisa).
- Clase CEncuestaGeneral (contiene la encuesta).

- La Figura 23 muestra el diagrama de clases Encuesta implementado.

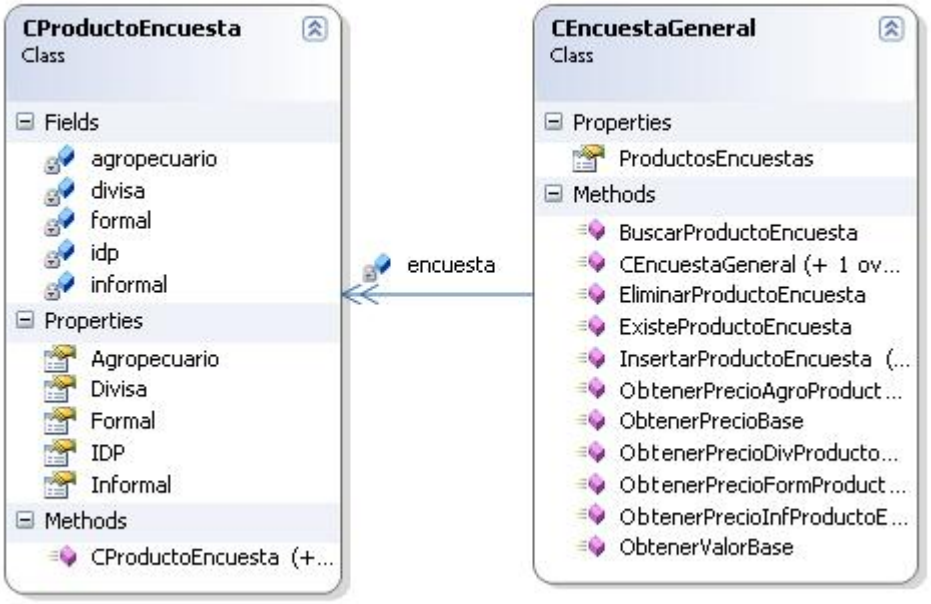


Figura 23: Diagrama de clases Encuesta

Después de analizar que en la aplicación existen códigos para clasificar los productos, mercados y provincias, se vio la posibilidad de crear una clase controladora la cual permita obtener los nombres de dichas entidades a través de su código, creando de esta manera la posibilidad de que si algún día existiera algún cambio de códigos o nombres solo se deberá realizar cambios en dicha interfaz sin necesidad de cambiar otras clases lo cual da la medida de la no dependencia de clases, logrando una aplicación extensible, y reutilizable ya que la misma puede servir para clasificar otras entidades definiendo solo el código y el nombre respectivo. Dicha clase se llama CConversionNomencladores la cual contiene las operaciones necesarias para lograr la conversión.

La Figura 24 muestra el diagrama de clases Nomenclador implementado.

Figura 24: Diagrama de clase Nomenclador

2.8 Algoritmos y Estructuras de datos utilizados.

Para una eficiente solución de las funcionalidades en el sistema, es muy importante tener en cuenta factores como, los algoritmos y estructuras de datos utilizados. Pues son capaces de almacenar y brindar la información necesaria que espera el cliente, además de ahorrar tiempo y líneas de código a implementar, lo que hace un sistema más eficiente y rápido.

Las listas son muy usadas en la aplicación pues estas poseen un gran número de operaciones ya implementadas que fueron de mucha utilidad como es el caso de adicionar, eliminar en una posición, saber longitud de la lista, además de que son colecciones genéricas pueden guardar cualquier tipo de datos, hasta los mismos que son creados por los programadores, en nuestro caso tenemos listas de productos, mercados, provincias etc. Estas se obtiene a través del lenguaje Microsoft Visual C#, utilizando la librería System.Collections.Generic. Para resolver el problema que habiamos plantedo en la capa de Negocio con respecto a la información que se obtiene del nombre del fichero, hicimos uso de la sub cadena (substring), que tambien son operaciones que brindan las listas para asi poder dividir el nombre del fichero en partes y se obtuvo las informaciones que se querian.

Para la problemática de la seguridad, se utilizó el algoritmo Md5 ya implmentado en el framework de .Net, algoritmo ampliamente usado, el cual permite poder almacenar la contraseña ya encriptada, y brinda de esta manera seguridad al sistema.

2.9 Descripción de las nuevas clases u operaciones necesarias.

Para lograr una aplicación eficiente hay que tener en cuenta las clases u operaciones, pues en ellas están encerradas todas las funcionalidades que conformara el sistema y de esta manera darle cumplimento a los requisitos planteados por el cliente, por tanto se debe ser muy cuidadoso en este sentido, a continuación presentaremos las clases u operaciones utilizadas en el sistema.

Tabla 1: Descripción de la clase CUsuario

NOMBRE:	CUSUARIO
Tipo de clase	Entidad
Atributo	**Tipo**
login rol password	string int string
Para cada responsabilidad	
Nombre:	**Descripción:**
CUsuario()	Constructor vacío, para la creación de objetos del tipo CUsuario
public CUsuario(string plogin, int prol, string pass)	Constructor por parámetros, para la creación de objetos con parámetros del tipo CUsuario
Login	Propiedades para la asignación y obtención del atributo login.
Rol	Propiedades para la asignación y obtención del atributo rol.
Password	Propiedades para la asignación y obtención del atributo password.
InsertarBD(CUsuario user)	Operación para insertar un usuario la base de datos, pasando el usuario por parámetros.
ActualizarBD(CUsuario user)	Operación para modificar un usuario la base de datos, pasando el usuario por parámetros.
EliminarBD(CUsuario user)	Operación para eliminar un usuario la base de datos, pasando el usuario por parámetros.

Tabla 2: Descripción de la clase CConversionNomencladores

NOMBRE:	CCONVERSIONNOMENCLADORES
Tipo de clase	Controladora
Atributo	**Tipo**
Para cada responsabilidad	
Nombre:	**Descripción:**
ConvertirProvincia(int COD)	Operación para obtener el nombre de una provincia dado el código.
ObtenerMercados(out string[] ListaCodigosMercados,outstring[] ListaNombreMercados)	Operación para dado una lista de los tipos de mercados devuelva una lista de los nombres de los mercados.
MercadoPorCodigo(string codmercado)	Operación para retornar el nombre de un mercado dado su código.
NombreProductoCodigo(int Cod)	Operación para obtener el nombre de un producto dado el código del mismo.
ObtenerCodigoMercado(string TipoMercado)	Operación para obtener el código de un mercado dado el tipo del mismo.
ConvertirAnno(string anno)	Operación para convertir el año del fichero a cuatro cifras pasándole por parámetros las dos últimas.

Tabla 3: Descripción de la clase CProducto

NOMBRE:	CPRODUCTO
Tipo de clase	Entidad
Atributo	**Tipo**
idp precio	int float
Para cada responsabilidad	
Nombre:	**Descripción:**
CProducto()	Constructor vacío, para la creación de objetos del tipo CProducto
CProducto(int p, float pre)	Constructor por parámetros, para la creación de objetos con parámetros del tipo CProducto
IDP	Propiedades para la asignación y obtención del atributo idp.
Precio	Propiedades para la asignación y obtención del

	atributo precio.

Tabla 4: Descripción de la clase CMercado

NOMBRE:	CMERCADO
Tipo de clase	Entidad
Atributo	**Tipo**
id tipo productos	string string List<CProducto>
Para cada responsabilidad	
Nombre:	**Descripción:**
CMercado()	Constructor vacío, para la creación de objetos del tipo CMercado.
CMercado(string a, string tip, List<CProducto> aproductos)	Constructor por parámetros, para la creación de objetos con parámetros del tipo CMercado.
ID	Propiedades para la asignación y obtención del atributo id.
Tipo	Propiedades para la asignación y obtención del atributo tipo.
InsertarProductoEnMercado(int cod, float pre)	Operación para insertar un producto en un mercado x, pasando por parámetros los atributos del producto.
InsertarProductoEnMercado(CProducto temp)	Operación para insertar un producto en un mercado x, pasando por parámetros un ítem de tipo producto.
ExisteProducto(int cod)	Operación para saber si determinado producto se encuentra en un mercado x, pasando por parámetro el código del producto.
BuscarProducto(int cod)	Operación para determinar la posición de un producto dentro de la lista de productos que contiene un mercado x, pasando por parámetros el código del producto.
EliminarProducto(int cod)	Operación para eliminar un producto de la lista de productos que contiene un mercado x, pasando por parámetros el código del producto a eliminar.
ObtenerPrecio(int cod)	Operación para obtener el precio de un producto de un mercado x, pasando por parámetros el

	código del producto.
List<CProducto> ObtenerProductos()	Operación para obtener la lista de productos de un mercado x.

Tabla 5: Descripción de la clase CProvincia

NOMBRE:	CPROVINCIA
Tipo de clase	Entidad
Atributo	**Tipo**
id nombre mercados	string string List<CMercado>
Para cada responsabilidad	
Nombre:	**Descripción:**
CProvincia()	Constructor vacío, para la creación de objetos del tipo CProvincia.
CProvincia(string pid, string pnombre, List<CMercado> amercados)	Constructor por parámetros, para la creación de objetos con parámetros del tipo CProvincia.
ID	Propiedades para la asignación y obtención del atributo id.
Nombre	Propiedades para la asignación y obtención del atributo nombre.
InsertarMercado(string id, string tip)	Operación para insertar un mercado en una provincia x, pasando por parámetros los atributos id y tipo del mercado.
InsertarMercado(CMercado temp)	Operación para insertar un mercado en una provincia x, pasando por parámetros un ítem de tipo CMercado.
ExisteMercado(string cod)	Operación para saber si determinado mercado se encuentra en una provincia x, pasando por parámetro el código del mercado.
BuscarMercado(int cod)	Operación para determinar la posición de un mercado dentro de la lista de mercados que contiene una provincia x, pasando por parámetros el código del mercado.
EliminarMercado(int cod)	Operación para eliminar un mercado de la lista de mercados que contiene una provincia x, pasando por parámetros el código del mercado a eliminar.

List<CMercado> ObtenerMercados()	Operación para obtener la lista de mercados de una provincia x.
List<CProducto> ObtenerProductosDadoMercado(string cod)	Operación para obtener la lista de productos de un mercado x entre la lista de mercados que contiene una provincia x, pasando por parámetros el código del mercado.

Tabla 6: Descripción de la clase CFusión

NOMBRE:	CFUSION
Tipo de clase	Controladora
Atributo	**Tipo**
año mes provincias ext	int int List<CProvincia> string
Para cada responsabilidad	
Nombre:	**Descripción:**
CFusion()	Constructor vacío, para la creación de objetos del tipo CFusion.
CFusion(int a, string m, List<CProvincia> aprovincias)	Constructor por parámetros, para la creación de objetos con parámetros del tipo CFusion.
Año	Propiedades para la asignación y obtención del atributo año.
Mes	Propiedades para la asignación y obtención del atributo mes.
InsertarProvincia(string id, string nom)	Operación para insertar una provincia en una fusión de fecha x, pasando por parámetros los atributos id y nombre de la provincia.
InsertarProvincia(CProvincia temp)	Operación para insertar una provincia en una fecha x, pasando por parámetros un ítem de tipo CProvincia.
ExisteProvincia(string cod)	Operación para saber si determinada provincia se encuentra dentro de una fusión de fecha x, pasando por parámetro el código de la provincia.
BuscarProvincia(string cod)	Operación para determinar la posición de una provincia dentro de la lista de provincias que contiene una fusión de fecha x, pasando por parámetros el código de la provincia.

EliminarProvincia(int cod)	Operación para eliminar una provincia de la lista de provincias que contiene una fusión x, pasando por parámetros el código de la provincia a eliminar.
List<CProvincia> ObtenerProvincias()	Operación para obtener la lista de provincias de una fusión x.
FusionValida(int mes, int anno)	Operación para saber si una fusiones valida o no, sabiendo su mes y año.
PrecioProducto(int codProv, string codMerc, int codProd)	Operación para obtener el precio de un producto en una provincia determinada, pasándole por parámetros la provincia, el mercado, y el producto sus respectivos códigos.
CMercado ObtenerFusionAnterior(string tipoMercado, int mes, int anno)	Operación para obtener la fusión de un mes anterior pasándole el año y el mes actual, de un mercado determinado.
List<CProducto> ParsearFichero(string URL)	Operación para leer del fichero la información enviada de las provincias, conociendo su URL
Fusionar(Fusion.CFicheroParse ListaFicheros)	Operación para Fusionar la información pasándole por parámetros todos los ficheros válidos a fusionar. La cual organiza los ficheros por provincias, mercados, productos y precios.

Tabla 7: Descripción de la clase CProductoEncuesta

NOMBRE:	CPRODUCTOENCUESTA
Tipo de clase	Entidad
Atributo	**Tipo**
idp	int
agropecuario	float
formal	float
informal	float
divisa	float
Para cada responsabilidad	
Nombre:	**Descripción:**
CProductoEncuesta ()	Constructor vacío, para la creación de objetos del tipo CProductoEncuesta.
CProductoEncuesta(int cod, float a, float f, float i, float d)	Constructor por parámetros, para la creación de objetos con parámetros del tipo CProductoEncuesta.
IDP	Propiedades para la asignación y obtención del atributo idp.

Agropecuario	Propiedades para la asignación y obtención del atributo agropecuario.
Formal	Propiedades para la asignación y obtención del atributo formal.
Informal	Propiedades para la asignación y obtención del atributo informal.
Divisa	Propiedades para la asignación y obtención del atributo divisa.

Tabla 8: Descripción de la clase CEncuestaGeneral

NOMBRE:	CENCUESTAGENERAL
Tipo de clase	Controladora
Atributo	**Tipo**
encuesta	List<CProductoEncuesta>
Para cada responsabilidad	
Nombre:	**Descripción:**
CEncuestaGeneral ()	Constructor vacío, para la creación de objetos del tipo CEncuestaGeneral.
CEncuestaGeneral (List<CProductoEncuesta> encuesta)	Constructor por parámetros, para la creación de objetos con parámetros del tipo CEncuestaGeneral.
InsertarProductoEncuesta(int cod, float agro, float form, float inf, float div)	Operación para insertar un producto en la encuesta, pasando por parámetros los atributos código del producto, valores de precio de los productos en los diferentes mercados, agropecuario, formal, informal, y divisa.
InsertarProductoEncuesta(CProductoEncuesta temp)	Operación para insertar un producto en la encuesta, pasando por parámetros un item del tipo CProductoEncuesta.
ExisteProductoEncuesta(int cod)	Operación para saber si determinado producto se encuentra en la encuesta pasando por parámetros el código del producto.
BuscarProductoEncuesta(int cod)	Operación para buscar la posición de un producto x dentro de la lista de productos en la encuesta, pasando por parámetros el código del producto.
EliminarProductoEncuesta(int cod)	Operación para eliminar un producto x de la lista de productos de la encuesta, pasando por parámetros el código del producto.

ObtenerPrecioAgroProductoEncuesta(int cod)	Operación para obtener el precio de un producto x del la lista de productos del mercado Agropecuario, pasando por parámetros el código del producto.
ObtenerPrecioFormProductoEncuesta(int cod)	Operación para obtener el precio de un producto x del la lista de productos del mercado Formal, pasando por parámetros el código del producto.
ObtenerPrecioInfProductoEncuesta(int cod)	Operación para obtener el precio de un producto x del la lista de productos del mercado Informal, pasando por parámetros el código del producto.
ObtenerPrecioDivProductoEncuesta(int cod)	Operación para obtener el precio de un producto x del la lista de productos del mercado Divisa, pasando por parámetros el código del producto.
ObtenerValorBase(int cod, string mercado)	Operación para obtener el valor del precio del año base pasándole por parámetros el código del producto, y el mercado donde va a comprobar su precio.
List<CProductoEncuesta> ProductosEncuestas	Operación para devolver un objeto de tipo encuesta.

Tabla 9: Descripción de la clase CProductoCanasta

NOMBRE:	CPRODUCTOCANASTA
Tipo de clase	Entidad
Atributo	**Tipo**
idp	int
nombre	string
preciomin	float
preciomax	float
preciomindiv	float
preciomaxdiv	float
unidad	string
Para cada responsabilidad	
Nombre:	**Descripción:**
CProductoCanasta()	Constructor vacío, para la creación de objetos del tipo CProductoCanasta.
CProductoCanasta(int cod, float a, float f, float i, float d)	Constructor por parámetros, para la creación de objetos con parámetros del tipo CProductoCanasta.
IDP	Propiedades para la asignación y obtención del atributo idp.

Nombre	Propiedades para la asignación y obtención del atributo nombre.
PreMin	Propiedades para la asignación y obtención del atributo preciomin.
PreMax	Propiedades para la asignación y obtención del atributo preciomax.
PreMinDiv	Propiedades para la asignación y obtención del atributo preciomindiv.
PreMaxDiv	Propiedades para la asignación y obtención del atributo preciomaxdiv.
Unidad	Propiedades para la asignación y obtención del atributo unidad.

Tabla 10: Descripción de la clase CCanastaGeneral

NOMBRE:	CCANASTAGENERAL
Tipo de clase	Controladora
Atributo	**Tipo**
canasta	List<CProductoCanasta>
Para cada responsabilidad	
Nombre:	**Descripción:**
CCanastaGeneral ()	Constructor vacío, para la creación de objetos del tipo CCanastaGeneral.
CCanastaGeneral (List<CProductoCanasta> canasta)	Constructor por parámetros, para la creación de objetos con parámetros del tipo CCanastaGeneral.
InsertarProductoCanasta(int cod, string nom, float min, float max, float mindiv, float maxdiv,string uni)	Operación para insertar un producto en la canasta, pasando por parámetros los atributos código del producto, nombre, precio mínimo, precio máximo, precio mínimo en divisa, precio máximo en divisa y unidad.
InsertarProductoCanasta(CProductoCanasta temp)	Operación para insertar un producto en la canasta, pasando por parámetros un ítem del tipo CProductoEncuesta.
ExisteProductoCanasta(int cod)	Operación para saber si determinado producto se encuentra en la canasta pasando por parámetros el código del producto.
BuscarProductoCanasta(int cod)	Operación para buscar la posición de un producto x dentro de la lista de productos de la canasta, pasando por parámetros el código del producto.

ExisteProductoCanastaNombre(string nom)	Operación para saber si determinado producto se encuentra en la canasta pasando por parámetros el nombre del producto.
EliminarProductoCanasta(int cod)	Operación para eliminar un producto x de la lista de productos de la canasta, pasando por parámetros el código del producto.
ModificarProductoCanasta(int cod, int ncod, string nnom, float nmin, float nmax, float nmindiv, float nmaxdiv, string nuni)	Operación para modificar un producto x de la lista de productos de la canasta, pasándole por parámetros el código del producto y los nuevos atributos del producto.
ObtenerNombreProductoCanasta(int cod)	Operación para obtener el nombre de un producto x del la lista de productos de la canasta, pasándole por parámetros el código del producto.
ObtenerMinProductoCanasta(int cod)	Operación para obtener el precio mínimo de un producto x del la lista de productos de la canasta, pasándole por parámetros el código del producto.
ObtenerMaxProductoCanasta(int cod)	Operación para obtener el precio máximo de un producto x del la lista de productos de la canasta, pasándole por parámetros el código del producto.
ObtenerMinDivProductoCanasta(int cod)	Operación para obtener el precio mínimo en divisa de un producto x del la lista de productos de la canasta, pasándole por parámetros el código del producto.
ObtenerMaxDivProductoCanasta(int cod)	Operación para obtener el precio máximo en divisa de un producto x del la lista de productos de la canasta, pasándole por parámetros el código del producto.
ObtenerUnidadProductoCanasta(int cod)	Operación para obtener la unidad de un producto x del la lista de productos de la canasta, pasándole por parámetros el código del producto.
CproductoCanasta ObtenerProductoCanasta(int codProducto)	Operación para obtener un objeto producto pasándole por parámetros el código del producto.
List<CProductoCanasta> ProductosCanasta	Operación para devolver la lista de productos de la canasta.
bool CargarCanastaBD()	Operación para saber si se logró cargar la canasta automáticamente de la base de datos

Tabla 11: Descripción de la clase CFichero

NOMBRE:	CFICHERO
Tipo de clase	Entidad

Atributo	Tipo
nombreFichero direcciónURL extension	string string string
Para cada responsabilidad	
Nombre:	**Descripción:**
CFichero()	Constructor vacío, para la creación de objetos del tipo CFichero.
CFichero(string dirreccion, string extencion, string nombre)	Constructor por parámetros, para la creación de objetos con parámetros del tipo CFichero.
NombreFichero	Propiedades para la asignación y obtención del atributo nombre.
DireccionURL	Propiedades para la asignación y obtención del atributo direccionURL.
Extension	Propiedades para la asignación y obtención del atributo extensión.
TipoMercado	Propiedades para la obtención del tipo de mercado información que se obtiene del nombre del fichero.
Anno	Propiedades para la obtención del año, información que se obtiene del nombre del fichero.
Mes	Propiedades para la obtención del mes, información que se obtiene del nombre del fichero.
Provincia	Propiedades para la obtención de la provincia, información que se obtiene del nombre del fichero.
Valido()	Saber si un fichero es válido por su nombre.
Valido(int mes, int anno, string ext)	Saber si un fichero es válido por su fecha y extensión.

Tabla 12: Descripción de la clase CFicheroParse

NOMBRE:	CFICHEROPARSE
Tipo de clase	Controladora
Atributo	**Tipo**
carpeta mes anno	string int int

ficheros ext	List <CFichero> string
Para cada responsabilidad	
Nombre:	**Descripción:**
CFicheroParse()	Constructor vacío, para la creación de objetos del tipo CFicheroParse.
CFicheroParse (carpeta)	Constructor por parámetros, para la creación de objetos con parámetros del tipo CFicheroParse.
Anno	Propiedades para la asignación y obtención del año a realizar la operación de parser.
extension	Propiedades para la asignación y obtención de la extensión a realizar la operación de parser.
Mes	Propiedades para la asignación y obtención del mes a realizar la operación de parser.
Carpeta	Propiedades para la asignación y obtención del atributo carpeta.
LLenarLista()	Llenar la lista de fichero a fusionar
ValidarLista()	Operación para validar la lista, para solo quedarnos con aquellos ficheros válidos a fusionar por la fecha entrada por el usuario.
List<CFichero> Ficheros	Operación que retorna la lista de ficheros.

2.10 Estrategia para la captura de errores.

Como todo sistema, para garantizar su correcto funcionamiento se debe tener en cuenta un tratamiento de errores, hacer captura de las excepciones que son lanzadas por el sistema o la base de datos para advertir al usuario de que la cosa va mal y debe ser cambiada, el sistema captura todas aquellas excepciones que son lanzadas y se le da un tratamiento para que el sistema no colapse.

Se sabe de antemano que a la hora de registrar un nuevo usuario por poner un ejemplo, la contraseña debe de ser confirmada y por tanto deben coincidir ambas contraseñas, nuestro sistema debe ser capaz de advertir al usuario en caso contrario, esta excepción es lanzada por el sistema.

- La Figura 25 muestra un ejemplo para el caso de crear un nuevo usuario, error de contraseña.

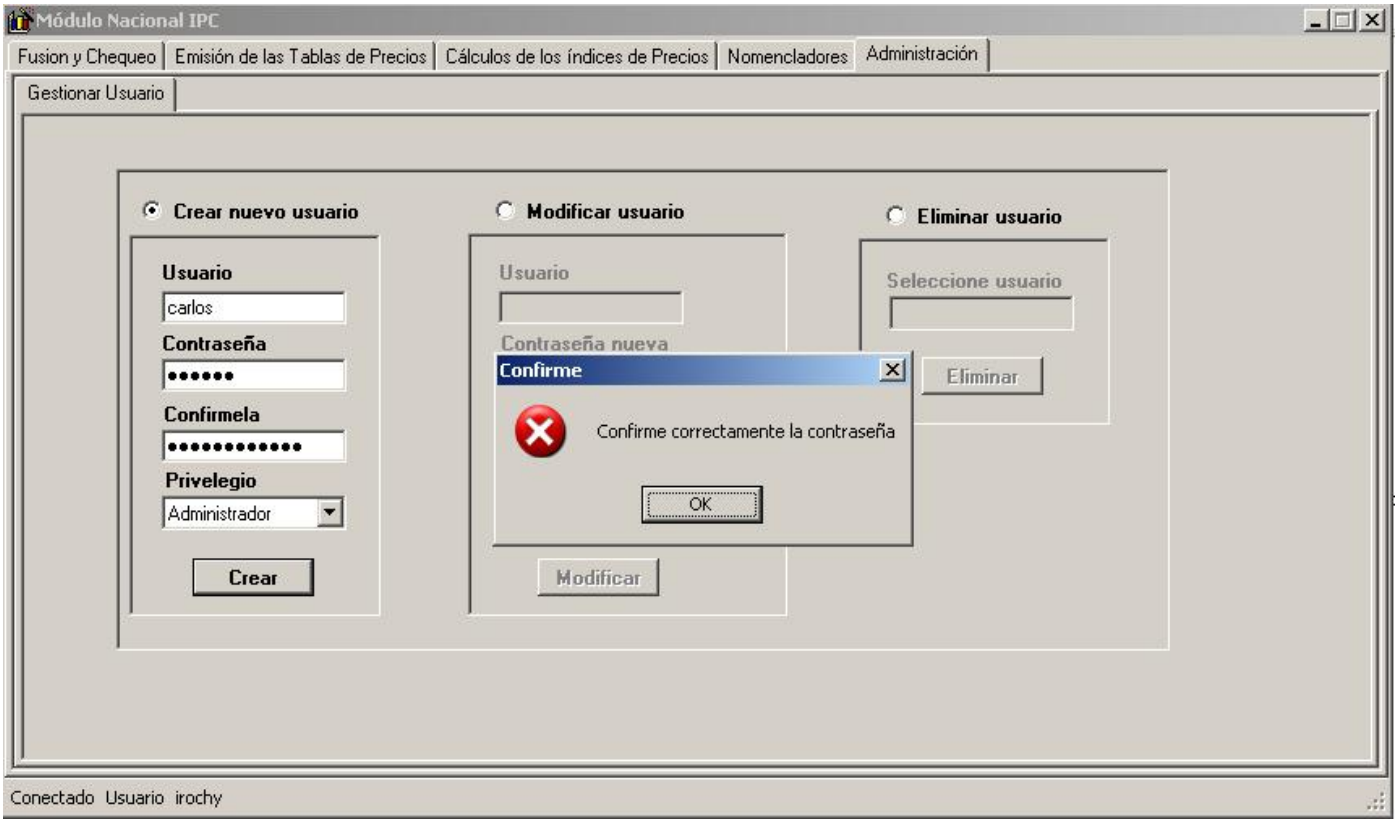


Figura 25: Tratamiento para la captura de excepciones del sistema

Ahora se vera un ejemplo de una excepción lanzada por la base de datos, para este caso comprobaremos de no dar margen a que existan dos productos con el mismo código, defecto encontrado en los ficheros enviados de provincia a la ONE, bajo ningún concepto esto puede ser posible.

- La Figura 26 muestra un ejemplo para el caso de tratar de insertar un producto con el mismo código.

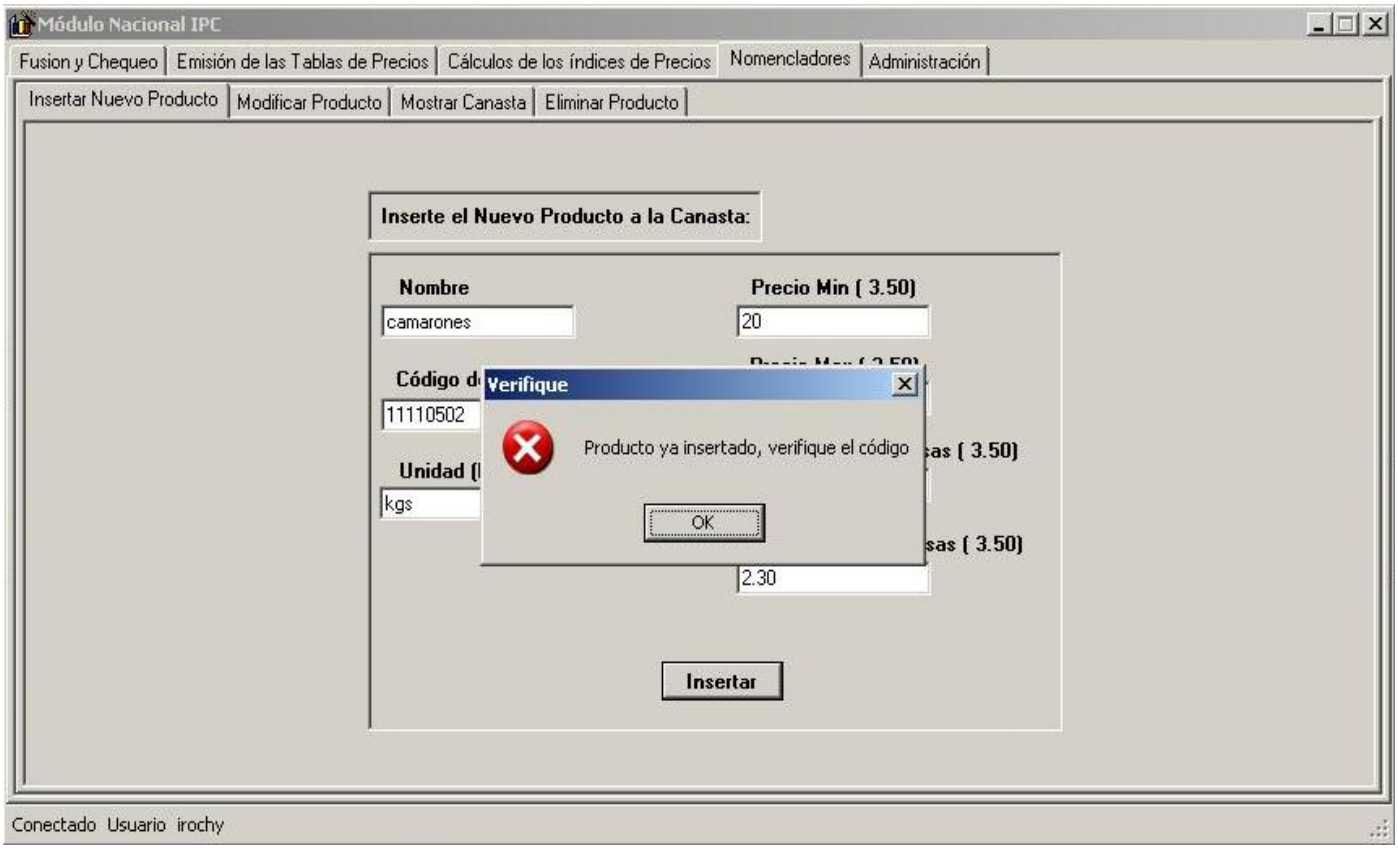


Figura 26: Tratamiento para la captura de excepciones sql

De esta manera nosotros damos solución a las problemáticas de los lanzamientos de excepciones que se dan en el sistema según las peticiones del usuario, utilizando los try - catch, funciones que posibilitan hacer capturas de errores, capturando los tipos de errores lanzadazos y mostrando de manera visible al usuario mensajes de confirmación, el cual le dará una idea de que anda mal y como solucionarlo.

- En la Figura 27 muestra como usamos los try - catch dentro del lenguaje Microsoft Visual C# y como se muestra los mensajes, especificando el tipo de mensaje (de tipo Error, de tipo Información).

```
private void button2_Click(object sender, EventArgs e)
{
    try
    {

        CProductoCanasta prod = new CProductoCanasta();
        prod.IDP = int.Parse(textBox3.Text);
        prod.Nombre = textBox1.Text;
        prod.PreMax = double.Parse(textBox10.Text);
        prod.PreMaxDiv = double.Parse(textBox8.Text);
        prod.PreMin = double.Parse(textBox2.Text);
        prod.PreMinDiv = double.Parse(textBox9.Text);
        prod.Unidad = textBox5.Text;
        DAO.DAO_Productos temp = new DAO.DAO_Productos();
        temp.Salvar(prod);
        MessageBox.Show("Producto insertado", "Insertado", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (FormatException)
    {
      MessageBox.Show("Una de las entradas no está en un formato válido", "Error de Formato", MessageBoxButtons.OK,
    }
    catch (Exception)
    {
        MessageBox.Show("Producto ya insertado, verifique el código", "Error", MessageBoxButtons.OK, MessageBoxIcon
    }

}
```

Figura 27: Utilización del los try - catch para la captura de excepciones

2.11 Proceso de Software Personal (PSP).

Un factor a tener en cuenta a la hora de desarrollar aplicaciones de calidad son los paradigmas del PSP, para hacer sistemas de software eficientes y seguros, sin que exista la posibilidad de que se creen errores aún después de culminada la aplicación, ya que después la misma, será usada por el cliente y este debe quedar satisfecho y poder entonces cumplir con sus necesidades, pues el PSP crea la base del trabajo individual, hace énfasis en la disciplina que se debe tener en cuenta en estos aspectos para desarrollo de aplicaciones, se basa en la gestión del tiempo, gestión de compromisos, y en la calidad, trabajando en la eliminación de errores desde edades tempranas del desarrollo.

Permitiendo un aprovechamiento del tiempo de manera optima y segura, de esta manera asegura que los programadores dediquen el tiempo necesario para la realización de las aplicaciones de manera que cumplan con las planificaciones definidas, haciendo uso de la tabla de tiempo en el cual se registran las actividades con el tiempo empleado y la cantidad de LOC (acrónimo de Lines of Code) por minuto en cada tarea de manera especifica. Trabaja además con el registro semanal de actividades el cual garantiza tener un estricto cumplimiento de las tareas a realizar así como el tiempo empleado en las mismas, esto permite una mayor calidad y aprovechamiento.

La tabla de registro de defectos permite agrupar la mayor cantidad de defectos, así como en que estado se encuentran los mismos, siendo esto algo fundamental en el desarrollo del sistema ya que permite corregirlos y de esta forma acerca cada vez más a un producto de calidad que cumpla con los requisitos establecidos.

- La Figura 28 muestra las tablas de resúmenes semanales, semana (1, 2).

Resumen de actividades sem 1				
Tarea	**Investigar**	**Codificar**	**Pruebas**	**Total**
Fecha				
D (6-12)				
Domingo	0	0	0	0
Lunes	30	0	0	30
Martes	40	0	0	40
Miercoles	0	0	0	0
Jueves	35	45	10	90
Viernes	35	60	15	110
Sabado	0	112	15	127
Total	140	217	40	**397**

Resumen de actividades sem 2				
Tarea	**Investigar**	**Codificar**	**Pruebas U**	**Total**
Fecha				
D (13-19)				
Domingo	0	0	0	0
Lunes	0	30	9	39
Martes	0	45	10	55
Miercoles	20	60	20	100
Jueves	0	25	10	35
Viernes	0	36	15	51
Sabado	30	45	20	95
Total	50	241	84	**375**

Resumen de actividades de la semana anterior				
Total	0	0	0	0
Media	0	0	0	0
Max	0	0	0	0
Min	0	0	0	0

Resumen de actividades de la semana anterior				
Total	140	217	40	397
Media	70	108.5	20	198.5
Max	140	217	40	397
Min	140	217	40	397

Resumen incluyendo la ultima semana				
Total	140	217	40	397
Media	70	108.5	20	198.5
Max	140	217	40	397
Min	140	217	40	397

Resumen incluyendo la ultima semana				
Total	190	458	124	772
Media	60	174.75	52	286.75
Max	140	241	84	465
Min	50	217	40	307

Figura 28: Resumen semanal de actividades (1,2)

- La Figura 29 muestra las tablas de resúmenes semanales, semana (3,4).

Resumen de actividades sem 3				
Tarea	Investigar	Codificar	Pruebas	Total
Fecha				
D (20-26)				
Domingo	0	0	0	0
Lunes	20	45	10	75
Martes	30	56	15	101
Miercoles	0	30	15	45
Jueves	0	20	10	30
Viernes	20	32	10	62
Sabado	0	36	15	51
Total	70	219	75	**364**

Resumen de actividades sem 4				
Tarea	Investigar	Codificar	Pruebas	Total
Fecha				
D (27-2)				
Domingo	0	0	0	0
Lunes	30	60	30	120
Martes	0	180	20	200
Miercoles	0	129	35	164
Jueves	20	128	20	168
Viernes	30	215	15	260
Sabado	10	200	10	220
Total	90	912	130	**1132**

Resumen de actividades de la semana anterior				
Total	190	458	124	772
Media	60	174.75	52	286.75
Max	140	241	84	465
Min	50	217	40	307

Resumen de actividades de la semana anterior				
Total	260	458	124	842
Media	65	174.75	52	291.75
Max	140	241	84	465
Min	50	217	40	307

Resumen incluyendo la ultima semana				
Total	260	677	199	1136
Media	65	653.375	718.375	1436.75
Max	140	241	84	465
Min	50	217	40	307

Resumen incluyendo la ultima semana				
Total	350	1370	1720	3440
Media	77.5	543.375	620.875	1241.75
Max	140	912	1052	2104
Min	50	217	40	307

Figura 29: Resumen semanal de actividades (3,4)

Se aprecia el tiempo dedicado por semana a la investigación, codificación, y la realización de pruebas, da medida de como se emplea el tiempo y sirve de planificación para semanas venideras. Pero no solo esto es lo único que el PSP brinda, también se usan tablas de desarrollo la cual permite hacer estimaciones de Minutos/LOC.

La Figura 30 muestra el tiempo dedicado a cada actividad en específico, así como el aprovechamiento del tiempo de trabajo y la cantidad de LOC que se realizan por minutos.

Tiempos de desarrollo en la Implementacion					
No.	Implementacion	Tiempo	LOC	Min/LOC	Funciones
	Capa de Presentación				
1	FrmPrincipal class	320	629	0.508744	Bucle For y Switch complejo
2	FrmProgreso class	10	18	0.5555556	Instruccion simple
3	Diseño e implementación de las IU_Usuarios	463			
	Capa de Negocio				
4	CProducto class	10	41	0.2439024	Instruccion simple
5	CMercado class	25	132	0.1893939	Instruccion simple
6	CProvincia class	45	121	0.3719008	Instruccion simple
7	CFusion class	93	225	0.4133333	Bucle For y Try complejo
8	CUsuario class	35	60	0.5833333	Instruccion simple
9	CProductoEncuesta class	25	69	0.3623188	Instruccion simple
10	CEncuestaGeneral class	53	88	0.6022727	Bucle For y Try sencillo
11	CProductoCanasta class	45	88	0.5113636	Instruccion simple
12	CCanastaGeneral class	70	191	0.3664921	Bucle For y Try complejo
13	CConversionNomencladores class	37	67	0.5522388	Bucle For sencillo
14	CFichero class	79	175	0.4514286	Try complejo
15	CFicheroParse class	60	133	0.4511278	Bucle For sencillo
Total		1370	2037	0.6725577	
Media		91.333333	145.5	0.6277205	

Figura 30: Registro del tiempo de implementación

Como detalle importante se verá ahora el cuaderno de defectos el cual proporciona una lista de defectos que han sido encontrados, brinda el tipo de defecto y la etapa del proyecto en que fueron introducidos. Permitiendo hacer una corrección temprana de los mismos y a la vez codificar de manera eficiente, lo que quiere decir que el uso de PSP en general es de vital importancia para obtener un producto de calidad.

- La Figura 31 muestra el cuaderno de defectos.

Cuaderno de registro de defecto						
Estudiante :	Irochy Oro Hurtado					
Profesora :	Merlyn Aviles Espinosa					
Fecha	**Número**	**Tipo**	**Introducido**	**Eliminado**	**Tiempo de corrección (min)**	**Defecto corregido**
23/05/2007	1	40	Implementación	23/05/2007	2	X
Descripción	Error en la declaración de la variable. Faltaba el tipo.					
Fecha	**Número**	**Tipo**	**Introducido**	**Eliminado**	**Tiempo de corrección (min)**	**Defecto corregido**
25/05/2007	2	80	Implementación	26/05/2007	15	X
Descripción	Error en el bucle para recorrer una lista, sin condicion de parada.					
Fecha	**Número**	**Tipo**	**Introducido**	**Eliminado**	**Tiempo de corrección (min)**	**Defecto corregido**
28/05/2007	3	80	Implementación	28/05/2007	10	X
Descripción	Error en bucle a la hora de rocorrer una lista e insertar un en una posición determinada.					
Fecha	**Número**	**Tipo**	**Introducido**	**Eliminado**	**Tiempo de corrección (min)**	**Defecto corregido**
02/06/2007	4	40	Implementación	02/06/2007	1	X
Descripción	Error en la comparacion de dos variables, utilizacion del operador de asignación					

Figura 31: Cuaderno de registro de defectos

2.12 Conclusiones.

De manera general todos estos detalles vistos en este capítulo dan sin lugar a duda la descripción de la solución propuesta, para lograr la implementación de las capas de Presentación y Negocio, todos los puntos que se han expuesto anteriormente, aplicados eficientemente, garantizan en su totalidad lo que es llamado un sistema de calidad, fiable, seguro etc.

Una vez concluido la aplicación, no todo esta ganado, a pesar de seguir bien los pasos de la solución propuesta, se debe tener en cuenta que antes de entregar el producto debemos de realizar ciertas pruebas al sistema para chequear su correcto funcionamiento.

Capítulo 3: Validación de la solución propuesta.

3.1 Introducción.

En este capítulo se realiza la validación a la solución propuesta en el capítulo anterior, de manera que se puede comprobar la eficiencia de las clases u operaciones utilizadas para dar respuesta a los distintos requisitos planteados por el cliente, para lo que se presentan las descripciones de las clases de prueba que verifican la validez de las funcionalidades del módulo nacional ***IPC***.

3.2 Herramienta utilizada para la validación.

A la hora de implementar sistemas informáticos, juega un papel fundamental el uso de las técnicas de evaluación dinámica o pruebas. Para esto se debe llevar a cabo ciertas estrategias para evaluar dinámicamente el software, se debe comenzar por los componentes más simples y más pequeños e ir avanzando progresivamente hasta probar el software en su totalidad. Para esto se utilizan las pruebas Unitarias. Pues es una forma de probar el correcto funcionamiento de un módulo de código. Dando la seguridad, que cada uno de los módulos funcione correctamente por separado. Las pruebas Unitarias tienen como objetivo conformar casos de prueba individuales. Un caso de prueba responde a una única pregunta sobre el código que está probando. Lo que permite aislar, cada parte del programa y mostrar que las partes individuales son correctas. Dando medida de que si todas las partes están bien pues el sistema también lo estará.

Para dar solución a los distintos casos de prueba, dentro del módulo nacional ***IPC***, se utilizó el Nunit Framework, el cual permite que sean ejecutados cada uno de los casos de prueba devolviendo visualmente al desarrollador los resultados que este espera. Y en caso de que no se encuentren estos valores devuelven los errores por lo que no se pudieron obtener los resultados

esperados, el cual indica que se debe hacer una revisión del método al cual se le esta aplicando la prueba, en caso de que los parámetros pasados sean correctos.

- La Figura 32 muestra un ejemplo del Nunit para un caso de prueba aceptado.

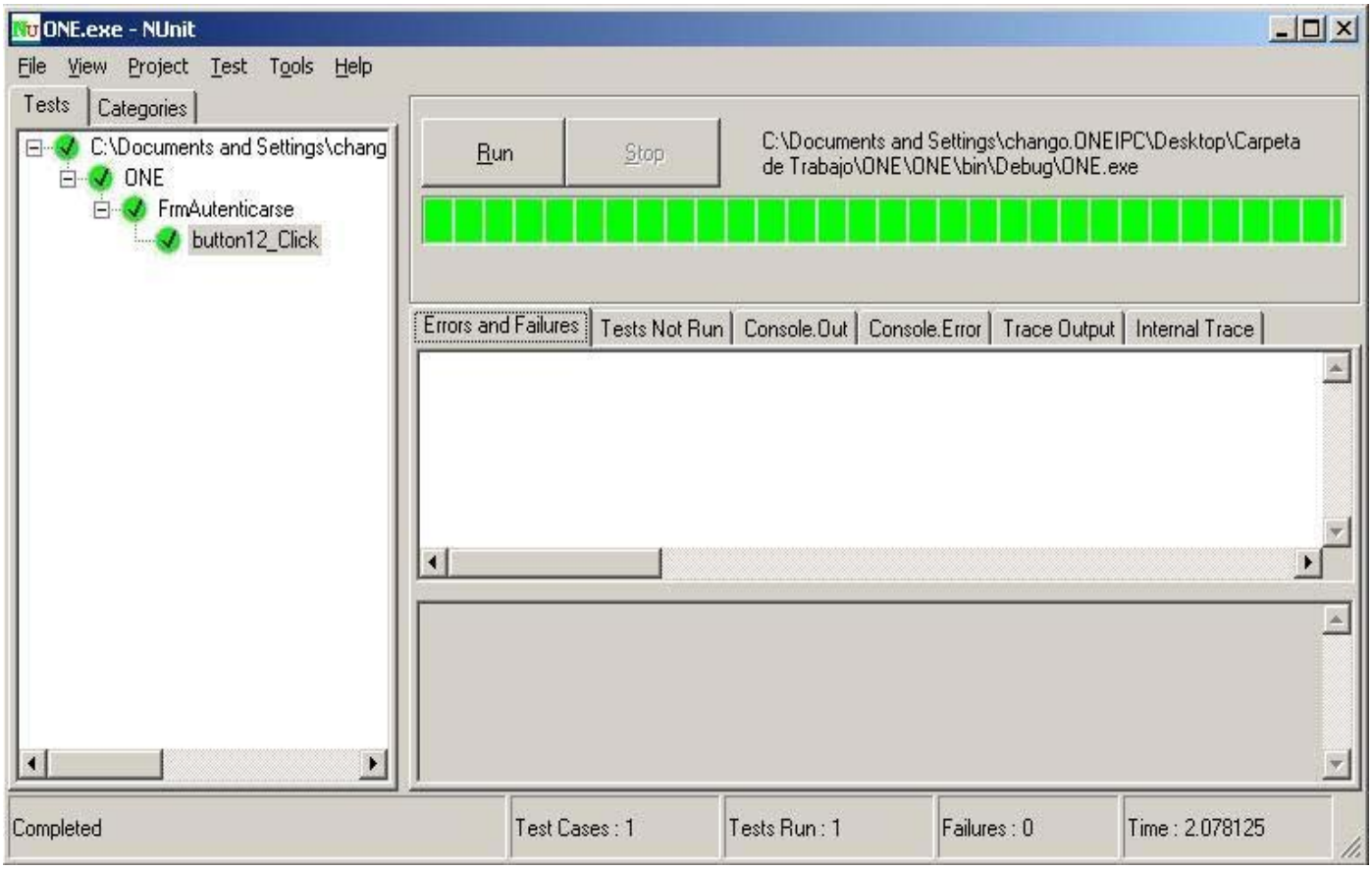


Figura 32: Caso de prueba aceptado para el caso de uso Autenticar Usuario

- La Figura 33 muestra un ejemplo del Nunit para un caso de prueba fallido.

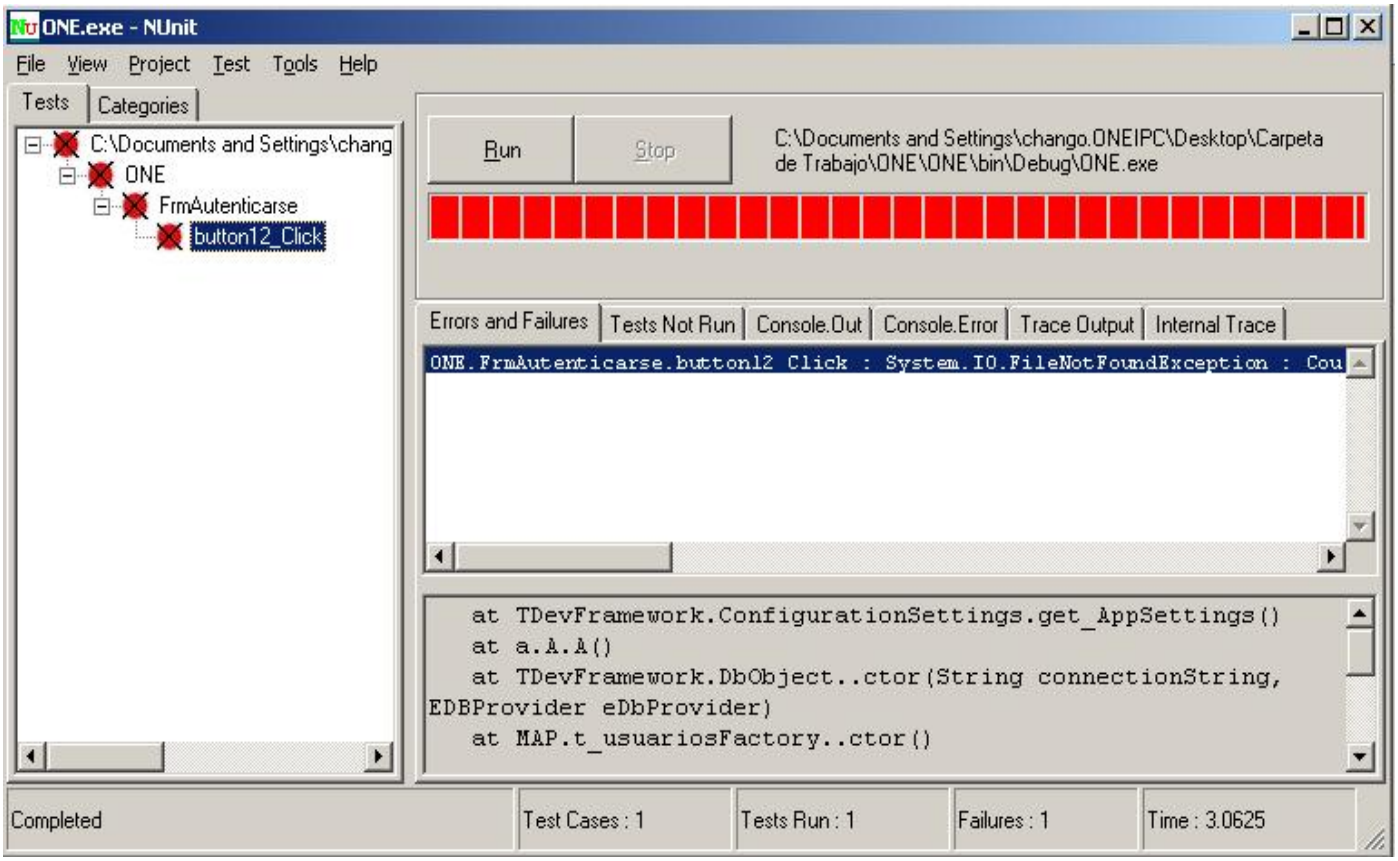


Figura 33: Caso de prueba fallido para el caso de uso Autenticar Usuario

3.3 Descripción de las pruebas realizadas.

A continuación se muestran los casos de pruebas del módulo nacional ***IPC*** las cuales pretenden mostrar las distintas condiciones que se deben cumplir, así como los resultados de las mismas.

Tabla 13: Descripción del caso de prueba para Autenticar Usuario

CASO DE USO	AUTENTICAR USUARIO
Caso de prueba	testVerificarUsuario

Condiciones	Para que esta prueba se cumpla satisfactoriamente el sistema debe de tener usuarios y contraseñas guardadas en la base de datos.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
user= ”irochy” pass= “irochy”	user = null pass = null user= “molina” pass= “molina”	El usuario se encuentra registrado en la base de datos, y su password coincide, el sistema permite el acceso. Para el caso contrario de que no exista el usuario o el password no coincida, el sistema debe negar el acceso.	Clases Válidas: El usuario esta registrado y coincide su pass. El sistema permite acceso. Clases no válidas: El sistema no permitió acceso.	En caso de que algún campo sea null. El sistema no permite acceso. El resultado de la prueba fue satisfactorio.

Tabla 14: Descripción del caso de prueba Asignar Privilegio

CASO DE USO	ASIGNAR PRIVILEGIO			
Caso de prueba	testAsignarPrivilegio			
Condiciones	Para que esta prueba se cumpla satisfactoriamente el sistema debe de tener algún usuario y este estar asignado a algún rol en la base de datos			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**

obj = ”irochy”	obj = ”null” obj = ”molina”	En caso de existir algún usuario y este debe tener asignado a algún rol en la base de datos, el sistema debe ser capaz de conceder a ese usuario sus privilegios según su rol. En caso de que no exista el usuario en la base de datos, no tendrá ningún rol respectivo y el sistema debe de no asignar privilegios.	Clases válidas: Los privilegios fueron asignados correctamente según el rol respectivo. Clases no válidas: El usuario no se encontraba, por tanto no tenia rol asignado y el sistema no concedió ningún tipo de privilegio.	En caso de que el usuario sea null, el sistema tampoco asigna ningún tipo de privilegios. El resultado de la prueba se obtuvo satisfactoriamente.

Tabla 15: Descripción del caso de prueba Gestionar Usuario

CASO DE USO	GESTIONAR USUARIO			
Caso de prueba	testInsertarUsuario			
Condiciones	Para que esta prueba se cumpla satisfactoriamente el sistema no debe tener el usuario en la base de datos.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones.**

obj= ”molina”	obj= ”null” obj= ”irochy”	En caso de que el usuario no se encuentre registrado en la base de datos y la confirmación de contraseña coincida, este debe ser insertado. En caso de que esté registrado en la base de datos no se podrá insertar.	Clases válidas: El usuario no estaba registrado y la confirmación de contraseña coincidió, por tanto fue insertado satisfactoriamente. Clases no válidas: El usuario se encontraba en la base de datos y no pudo ser insertado.	En caso de que el usuario sea null el sistema tampoco realizará la función de insertar. El resultado de la prueba fue satisfactorio.

Tabla 16: Descripción del caso de prueba Modificar Usuario

CASO DE USO	GESTIONAR USUARIO			
Caso de prueba	testModificarUsuario			
Condiciones	Para que esta prueba se cumpla satisfactoriamente el sistema debe tener al menos un usuario en la base de datos.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones.**

obj= ”irochy”	obj= ”null” obj= ”jose”	En caso de que el usuario se encuentre registrado en la base de datos el mismo podrá, ser modificado. En caso de que no esté registrado en la base de datos no se podrá modificar.	Clases válidas: El usuario estaba registrado en la base de datos y pudo ser modificado satisfactoriamente. Clases no válidas: El usuario no se encontraba en la base de datos y no pudo ser modificado.	En caso de que el usuario sea null el sistema tampoco realizará la función de modificar. El resultado de la prueba fue satisfactorio.

Tabla 17: Descripción del caso de prueba Eliminar Usuario

CASO DE USO	GESTIONAR USUARIO			
Caso de prueba	testEliminarUsuario			
Condiciones	Para que esta prueba se cumpla satisfactoriamente el sistema debe de tener al menos un usuario en la base de datos			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones.**
obj= ”irochy”	obj= ”null” obj= ”jose”	En caso de que el usuario se encuentre registrado en la base de datos	Clases válidas: El usuario estaba registrado y pudo ser eliminado con éxito.	En caso de que el usuario sea null el sistema tampoco podrá realizar la función de eliminar

		este debe ser eliminado. En caso de que este no este registrado en la base de datos no se podrá eliminar.	Clases no válidas: El usuario no se encontraba en la base de datos y no pudo se eliminado.	puesto que no se sabe el usuario a eliminar. El resultado de la prueba fue satisfactorio.

Tabla 18: Descripción del caso de prueba Insertar Producto

CASO DE USO	GESTIONAR PRODUCTOS			
Caso de prueba	testInsertarProducto			
Condiciones	Para que esta prueba se cumpla satisfactoriamente el sistema no debe tener el producto en la base de datos.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones.**
obj= ”pizza”	obj= ”null” obj= ”pelly”	En caso de que el producto no se encuentre registrado en la base de datos, este debe ser insertado. En caso de que esté registrado en la base de datos	Clases válidas: El producto no estaba registrado, por tanto fue insertado satisfactoriamente. Clases no válidas: El producto se	En caso de que el producto sea null el sistema tampoco realizará la función de insertar. El resultado de la prueba fue satisfactorio.

		no se podrá insertar.	encontraba en la base de datos y no pudo ser insertado.	

Tabla 19: Descripción del caso de prueba Modificar Producto

CASO DE USO	GESTIONAR PRODUCTO			
Caso de prueba	testModificarProducto			
Condiciones	Para que esta prueba se cumpla satisfactoriamente el sistema debe tener al menos un producto en la base de datos.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones.**
obj= ”pizza”	obj= ”espaguetis”	En caso de que el producto se encuentre registrado en la base de datos el mismo podrá, ser modificado. En caso de que no esté registrado en la base de datos no se podrá modificar.	Clases válidas: El producto estaba registrado en la base de datos y pudo ser modificado satisfactoriamente. Clases no válidas: El producto no se encontraba en la base de datos y no pudo ser modificado.	El resultado de la prueba fue satisfactorio.

Tabla 20: Descripción del caso de prueba Eliminar Producto

CASO DE USO	GESTIONAR PRODUCTO			
Caso de prueba	testEliminarProducto			
Condiciones	Para que esta prueba se cumpla satisfactoriamente el sistema debe tener al menos un producto en la base de datos.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones.**
obj= ”pizza”	obj= ”espaguetis”	En caso de que el producto se encuentre registrado en la base de datos el mismo podrá, ser eliminado. En caso de que no esté registrado en la base de datos no se podrá eliminar.	Clases válidas: El producto estaba registrado en la base de datos y pudo ser eliminado satisfactoriamente. Clases no válidas: El producto no se encontraba en la base de datos y no pudo ser modificado.	El resultado de la prueba fue satisfactorio.

Tabla 21: Descripción del caso de prueba Emitir Fusión

CASO DE USO	EMITIR FUSIÓN
Caso de prueba	testEmitirFusión

Condiciones	Para que esta prueba se cumpla satisfactoriamente el sistema debe tener al menos una fusión almacenada en la base de datos.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones.**
obj= ”Enero,2007,Formal”	obj= ”null” obj= ”Noviembre,2007,Formal”	En caso de que la fusión se encuentre registrada en la base de datos la misma podrá emitirse. En caso de que no esté registrada en la base de datos no se podrá emitir.	Clases válidas: La fusión estaba registrada en la base de datos y pudo ser emitida satisfactoriamente. Clases no válidas: La fusión no se encontraba en la base de datos y no pudo ser emitida.	En caso de que la fusión sea null, tampoco el sistema podrá emitirla. El resultado de la prueba fue satisfactorio.

Tabla 22: Descripción del caso de prueba Emitir Tablas de Precios

CASO DE USO	EMITIR TABLAS DE PRECIOS			
Caso de prueba	testEmitirTablasPrecios			
Condiciones	Para que esta prueba se cumpla satisfactoriamente el sistema debe tener al menos una fusión almacenada en la base de datos.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones.**

obj= ”Enero,2007,Formal”	obj= ”null” obj= ”Noviembre,2007,Formal”	En caso de que la fusión se encuentre registrada en la base de datos la misma podrá emitir la tabla de precios. En caso de que no esté registrada en la base de datos no se podrá emitir la tabla de precios.	Clases válidas: La fusión estaba registrada en la base de datos y pudo ser emitida la tabla de precios satisfactoriamente. Clases no válidas: La fusión no se encontraba en la base de datos y no pudo ser emitida la tabla de precios.	En caso de que la fusión sea null, tampoco el sistema podrá emitir la tabla de precios. El resultado de la prueba fue satisfactorio.

Tabla 23: Descripción del caso de prueba Cálculo de Índices de Precios

CASO DE USO	CÁLCULOS DE ÍNDICES DE PRECIOS			
Caso de prueba	testCálculosÍndicesPrecios			
Condiciones	Para que esta prueba se cumpla satisfactoriamente el sistema debe tener al menos una fusión almacenada en la base de datos.			
Clases Válidas	**Clases Inválidas**	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones.**

obj= ”Marzo,2007,Formal”	obj= ”null” obj= ”Noviembre,2007,Formal”	En caso de que la fusión se encuentre registrada en la base de datos la misma podrá emitir la tabla con los índices calculados. En caso de que no esté registrada en la base de datos no se podrá emitir la tabla de índices.	Clases válidas: La fusión estaba registrada en la base de datos y pudo ser emitida la tabla de índices satisfactoriamente. Clases no válidas: La fusión no se encontraba en la base de datos y no pudo ser emitida la tabla de índices.	En caso de que la fusión sea null, tampoco el sistema podrá emitir la tabla de índices. El resultado de la prueba fue satisfactorio.

3.4 Resultados obtenidos.

Después que se obtuvo los resultados de las pruebas unitarias, se puede hablar de los resultados obtenidos: todas las pruebas dieron resultados satisfactorios, de esta manera se demuestra que el sistema cumple con cada uno de los casos de uso que lo conforman. Siendo posible que sea una aplicación eficiente en su totalidad y cubre todas las necesidades del cliente.

3.5 Conclusiones.

El empleo de estas pruebas permitió verificar la solidez de la implementación de la lógica del negocio en el sistema, utilizadas posteriormente por la capa inmediata superior o capa de Presentación para completar la total integración entre las capas que conforman la aplicación.

En este capítulo se realizó una evaluación de la solución planteada en el capítulo anterior, para lograr esto, se le aplicaron pruebas a la implementación de los casos de usos de la capa de Negocio del módulo nacional ***IPC***.

Conclusiones Generales.

Se puede afirmar que se le da cumplimiento a las tareas de la investigación propuestas en este trabajo de diploma pues se logró implementar un sistema más flexible y seguro para la realización de cálculos del ***IPC***, que el existente y que da respuesta a todas las necesidades planteadas por el cliente de una forma correcta.

Se realizó un estudio del ***IPC***, lo que logró una mejor familiarización y entendimiento de esta importante estadística, así como las principales características de la POO, lo que permitió hacer uso de algunas de ellas en la confección del nuevo sistema, se implementaron todas las clases u operaciones que dieron lugar a eficientes resultados, la estrategia de integración entre capas y el proceso de capturas de errores hizo que el sistema funcionara de forma sólida y segura evitando que el mismo colapse, además el uso de patrones dio lugar a una mayor cohesión, se aplicaron también los principios del PSP en aras de una mejor planificación y aprovechamiento del tiempo de trabajo. Se crearon los roles específicos para garantizar la seguridad del sistema.

Se expone de manera visible las pruebas Unitarias realizadas a las funcionalidades que conforman el sistema, así como la herramienta utilizada para lograr las mismas, mostrando dando la medida que la aplicación cumple sus objetivos.

Recomendaciones.

Partiendo de la idea que el proceso de desarrollo de software es iterativo e incremental, se pretende seguir trabajando en este sistema, por lo que para el futuro se recomienda:

- Realizar una aplicación web que permita realizar cálculos de ***IPC*** desde las distintas provincias del país.
- Implementar nuevas funcionalidades las cuales permitan cálculos de ***IPC*** por tipo de monedas y cálculo ampliado.
- Emigrar la solución a plataformas de software libre, se recomienda Mono para poder hacer uso de la mayor cantidad de código.

Bibliografía.

ALFARO, F. M. *Procesamiento y Difusión del IPC*, 2001. [Disponible en: http://www.inei.gob.pe/biblioineipub/bancopub/Est/Lib0344/procesa.HTM

ALVAREZ, M. A. *Qué es la Programación Orientada a Objetos*, 2007. [Disponible en: http://www.desarrolloweb.com/articulos/499.php

CAMEJO, D. *Propuesta de la arquitectura para el Sistema Integral de Cálculo de Índices de Precios al Consumidor*, UCI, 2007.

FERNÁNDEZ, L. A. C. *Fundamentos de la POO*, 2007. [Disponible en: http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_2578.asp

HUMPHREY, W. S. *Introducción al Proceso Software Personal*, 2001.

IZQUIERDO, J. *Microsoft presenta el precio de Visual Studio 2005*, 2007. [Disponible en: http://www.desarrolloweb.com/articulos/s1940.php

LARMAN, C. *UML y Patrones. Introducción al análisis y diseño orientado a objetos. Volumen 1 y 2*. La Habana, Editorial Félix Varela, 2004. p. 84-205-3438-2

LUIS, R. M. *Diseño del Sistema Integral para el Cálculo de Índices de Precios al Consumidor*, UCI, 2007.

MARE, D. A. D. *Reutilización de Contenedores Parametrizables con Lenguajes de Semántica Limitada* 1999. p.

ONE Índice de Precios al Consumidor, 2000.

PÉREZ, R. R.; A. V. GARCÍA, *et al. Eficiencia en los algoritmos*, 2007.

POL, H. A. *El modelo de objetos*, 1996-1997. [Disponible en: http://fpsalmon.usc.es/genp/doc/cursos/poo/modelo.html

Programación Orientada a Objeto (POO). 2007. [Disponible en: http://mipagina.cantv.net/dreamlocos/dreamlocos/dream/mirc/programacion_objetos.htm

RÍOS, E. *Sistema de Cálculo de Índices de Precios al Consumidor*, UCI, 2007.

TESO, L. D. *El paradigma orientado a objetos*, 2007. [Disponible en: http://www.monografias.com/trabajos14/paradigma/paradigma.shtml

TORRES, F. *Revisiones de código y estándares de codificación.*, 2006. [Disponible en: http://www.dosmilmastres.com/blog.php?anotacionid=4

WELICKI, L. *El Patrón Singleton*, 2007. [Disponible en: http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_4081.asp