
# Universidad de las Ciencias Informáticas

Facultad 3

Trabajo de Diploma para optar por el título de Ingeniería Informática.

**Titulo:** "Sistema de descarga y procesamiento automatizado de patentes en Bases de Datos de Internet. Rol de Programador, Modulo Parseo".

## AUTOR

Yoandry Castellanos López

## TUTOR

Ing. Yalice Gámez Batista

## CONSULTANTE

Ing. Elvio Ramón Chávez González

Ciudad de La Habana, Cuba
Junio, 2007.

## DECLARACIÓN DE AUTORÍA

Por este medio declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas para que hagan el uso que estimen pertinente con este trabajo.

Para que así conste firmamos la presente a los x días del mes de x del 2007.

---------- ----------

Firma del Autor Firma Tutor

**AGRADECIMIENTOS**

*A todas las personas que de una forma u otra han tenido que ver con el desarrollo de este trabajo.*

*A mi compañero Dusniel, Disnier, Dietmar por compartir en estos 5 años de carrera todos los conocimientos y por brindarme su amistad.*

*A mi consultante Elvio que siempre ha estado preocupado por el estado de mi tesis.*

*A Luís Ernesto por inculcarme ese habito de superación y de conocimiento.*

*A mis compañeros de proyecto por soportarme todos estos años.*

*A mi querida novia por siempre estar ahí apoyándome en lo que me hiciera falta.*

*A todos mis amigos del baloncesto, y a todos los que me preguntaron por el estado de mi tesis.*

*A la Revolución por permitirme hacer realidad mis sueños.*

**DEDICATORIA**

*A mis queridos padres por ser guías y ejemplos.*

*A mi querida novia por darme tanto amor y cariño en estos años.*

*A mi abuela Mirtha que siempre se ha preocupado por mí y que siempre me ha apoyado.*

*A todos mis amigos.*

*A mi Tío Renecito que siempre me apoyo y me dio fuerzas para seguir adelante.*

*A mi Tía Angelita que siempre me ha dado todo su apoyo cariño y comprensión.*

*La familia es el espejo de la sociedad. Víctor Hugo.*

## RESUMEN

En un entorno como el de la Consultoría del Ministerio de la Informática y las Comunicaciones, se realizan análisis de grandes volúmenes de información para hacer ejercicios de inteligencia de mercado, perfiles estratégicos y análisis de tendencias, sólo por mencionar algunos de los más relevantes. Para realizar los trabajos antes mencionados se requiere del análisis de grandes volúmenes de información acerca de los datos bibliográficos de las patentes, descargada desde Internet, entiéndase páginas HTML, o información que debe ser procesada antes de comenzar el estudio.

En la actualidad, este proceso se realiza de forma manual, lo cual trae incorporado demora y errores, lo que influye de forma negativa en la calidad y eficiencia del proceso. En este trabajo se hace un estudio detallado de la situación antes planteada, y se brinda una solución de implementación, a través del módulo Parseo, para el procesamiento de las patentes descargadas por el sistema; con el objetivo principal de obtener los campos válidos de las patentes definidos por el cliente, y aumentar con esto la calidad y la productividad de los trabajadores de Delfos.

## ÍNDICE DE CONTENIDO

## INDICE DE TABLAS

## INDICE DE FIGURAS

# INTRODUCCIÓN

En la actualidad las patentes[1] poseen una importancia extrema para configurar la estrategia de negocio de las compañías tecnológicas. Las patentes constituyen una forma de proteger la propiedad intelectual, lo que las convierte en una fuente importantísima de información sobre la competencia y las hace a su vez, una herramienta principal de la vigilancia tecnológica debido a que se hace necesario a las empresas conocer que patentes están vigentes en el mercado antes de ponerse a desarrollar nuevos productos, con una considerable inversión que podría verse bloqueada.

El uso de patentes como indicador de innovación ha sido exhaustivamente estudiado y ha alcanzado un gran nivel de madurez en la actualidad debido a que las patentes son documentos que contienen información muy valiosa desde el punto de vista legal, técnico y comercial. Las patentes son de gran importancia, no sólo para determinar indicadores de innovación de un país determinado, sino para establecer estrategias de negocios de las compañías tecnológicas, y políticas de desarrollo en países menos desarrollados. Se considera que el 80% de la información que aparece en las patentes no aparece publicado en otras fuentes (ESCORSA 2001), por lo que constituyen una fuente ideal para la vigilancia tecnológica y la inteligencia competitiva.

En Cuba existen empresas que se dedican exactamente a eso, a la vigilancia tecnológica. Tal es el caso de la Consultaría Delfos, que tiene como misión principal, actuar como centro coordinador del sistema de información del ministerio de la informática y las comunicaciones, y ejercer la vigilancia tecnológica como apoyo a la actividad gerencial. Esta consultoría brinda servicios como el estudio de tendencias, que no son más que los estudios basados en el análisis cualitativo y cuantitativo de la información de publicaciones científico-técnicas y patentes, entre otras, para determinar el estado actual y las principales tendencias en las investigaciones científicas-tecnológicas y/o comerciales en el sector de las tecnologías de la información y las comunicaciones (Tics).

[1] Una Patente es un título que reconoce el derecho de explotar en exclusiva la invención patentada, impidiendo a otros su fabricación, venta o utilización sin consentimiento del titular.

Actualmente el entorno en el que un trabajador de Delfos desarrolla su trabajo puede incluir tanto el acceso a Bases de datos de patentes (BDP), como el acceso a otras bases de datos (BD) bibliográficas, normalmente de tipo científico–tecnológico que se suelen complementar entre ellas.

Estas BD pueden estar situadas tanto en un puesto de red local (por ejemplo en una base de datos propia o en una base de datos comercial en CD), como en un sitio Web. Para realizar estudios de tendencias o de mercado, el investigador se conecta a las BDP de patentes que están publicadas en Internet, y tiene que descargar grandes volúmenes de información correspondientes a las páginas Web de las patentes sobre la cual está realizando su investigación, y si este proceso no está automatizado, es casi imposible o se necesitarían muchas horas y recursos humanos para realizar esta tarea de forma manual, y esto es solamente el proceso de descarga, después de descargar las páginas Web de las patentes ellos necesitan obtener los campos válidos de las mismas, para ello el trabajador o el investigador que esté realizando dicha tarea necesita pasarla por varias macros con el objetivo de filtrar la información correspondiente a las patentes. Este proceso trae consigo demoras y en ocasiones se producen incumplimientos en la realización de dichas tareas. Por este motivo los directores de Delfos se vieron en la necesidad de buscar un software que agilizara el trabajo con el fin de mejorar los servicios que ellos prestaban, y es entonces que surge en el 2005 Predictor “Sistema de descarga y procesamiento automatizado de patentes en Bases de Datos de Internet”. Esta primera versión del software incorpora los módulos de análisis y descarga de varias bases de datos libres en Internet como son espacenet.com de España. Esta primera versión fue desarrollada en la Plataforma.NET, específicamente en VisualStudio.Net 2005. Más tarde en la segunda versión se incorporan los mismos módulos pero para otras Bases de Datos, especialmente la base de datos de Estados Unidos, con la presencia de una arquitectura mucho más robusta, y más adaptable a cambios pedidos por el cliente basada en tres capas, donde la capa intermedia es la encargada de modelar y llevar paso a paso todo el proceso de negocio.

Para que el proyecto funcionara correctamente se aplica la metodología RUP y lo que esta propone para el desarrollo por roles dentro de un proyecto.

El Rol de desarrollador o programador tiene un papel importante en el desarrollo de un software. Un desarrollador o programador debe ser una persona capaz, responsable, que le guste estudiar. Debe poseer un buen conocimiento y dominio de la herramienta en la cual se va a desarrollar el sistema, debe

conocer y saber aplicar estándares de codificación para lograr la uniformidad entre los demás desarrolladores, así como los principales patrones de diseño y arquitectura, con el fin de mejorar la calidad del software a desarrollar, y lograr que su sistema sea lo más eficiente posible, y se pueda adaptar a cambios pedidos por el cliente, sin afectar la estructura del mismo.

Por todo lo planteado anteriormente, en este trabajo se abordará el rol de programador o desarrollador, como uno de los roles importantes en el desarrollo de un software.

**Problema**: ¿Cómo puede el desarrollador implementar el Módulo de Parseo de forma satisfactoria, para que el software tenga calidad y cumpla con los requisitos planteados por los clientes?

**Hipótesis**: Si no se aplican de forma correcta los estándares de codificación y los patrones que garantizan que la aplicación sea robusta, fácil de mantener, y que sea adaptable a los cambios pedidos por el cliente, ni se lleva a cabo una implementación correcta de los casos de uso; entonces el software no tendrá la calidad deseada, ni cumplirá con los requerimientos especificados por los clientes.

**Objeto de Estudio**: Procesos de Desarrollo del Procesamiento de Información.

**Campo de Acción:** Procesos de Desarrollo del Procesamiento de Información en páginas web.

**Objetivo General:** Implementar el módulo de Parseo del Sistema Predictor usando métodos sofisticados de programación que permitan el buen funcionamiento del sistema.

**Tareas**

- Aplicar un estándar de codificación correcto.
- Desarrollo de algoritmos para el procesamiento de información.
- Desarrollo de Casos de Prueba que certifiquen la veracidad de los algoritmos empleados.

**Diseño Metodológico de la Investigación**

**Métodos teóricos**:

- **Análisis Histórico-Lógico**: Este método permite conocer las tendencias actuales de desarrollo de software de procesamiento y análisis de patentes para darle solución al problema planteado.

- **Modelación**: Se usa para representar el trabajo realizado en la oficina y de esta manera tener una visión objetiva del proceso.

# CAPITULO 1 Fundamentación teórica.

## Introducción

En este capitulo se expone un análisis general de las distintas técnicas de programación, y se hace un estudio minucioso de los diferentes lenguajes de programación, tecnologías, frameworks y tendencias de uso para el desarrollo del modulo de Parseo del Sistema Predictor.

### 1.1. Paradigmas de programación.

A lo largo de este epígrafe se hará un estudio breve sobre los diferentes paradigmas[2] de programación, haciendo mayor énfasis en el paradigma que posee el lenguaje escogido para desarrollar el sistema.

#### 1.1.1 ¿Que es un paradigma de programación?

Se define como un conjunto de reglas a seguir que facilitan la tarea de programar. Son consejos de expertos programadores, que tras años de trabajo, han acumulado una gran experiencia. Pero estas técnicas, son obviamente, independientes del lenguaje en que se desarrolle. Una técnica de programación no es, obviamente, un lenguaje, pero puede aplicarse a cualquier lenguaje (Eidos 2000).

Otra definición es: colección de modelos conceptuales que juntos modelan el proceso de diseño y determinan, al final, la estructura de un programa. (MENÉNDEZ 2002).

En la actualidad existen múltiples paradigmas de programación, ninguno es mejor que otro, simplemente que cada uno tiene sus ventajas y desventajas. A continuación se dará una breve descripción de los

[2] Los paradigmas son un conjunto de conocimientos y creencias que forman una visión del mundo (cosmovisión), en torno a una teoría hegemónica en determinado periodo histórico.

paradigmas que se consideran más importantes, dedicando todo un epígrafe al paradigma aplicado al lenguaje escogido para implementar el sistema, debido a la gran importancia del mismo.

### 1.1.2 Tipos de Paradigmas.

En este epígrafe se explica de forma breve los diferentes paradigmas o técnicas de programación más importantes, haciendo énfasis en la técnica escogida que no es más que la técnica de programación orientada a objeto (POO).

- Programación imperativa.
- Programación Funcional.
- Programación Lógica.
- Programación Estructurada.
- Programación dirigida por eventos
- Programación Orientada a Aspectos.
- Programación Orientada a Objetos.
- Programación con restricciones.
- Programación por capas.
- Programación a nivel funcional.
- Programación a nivel de valores.

#### 1.1.2.1 Programación Imperativa.

La programación imperativa describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que

indican al computador cómo realizar una tarea. Ejemplos de lenguajes imperativos: Basic, C, C++, C#, Java, Perl.

#### 1.1.2.2 Programación Funcional.

La programación funcional tiene como objetivo conseguir lenguajes expresivos y matemáticamente elegantes, en los que no sea necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa, y evitando el concepto de estado del cómputo. La secuencia de computaciones llevadas a cabo por el programa se regiría única y exclusivamente por la reescritura de definiciones más amplias a otras cada vez más concretas y definidas, usando lo que se denominan "definiciones dirigidas".

Ejemplos de lenguajes funcionales: Haskell , Miranda,  Lisp, Scheme, Ocaml y Standard ML.

#### 1.1.2.3 Programación por Capas.

La programación por capas es un estilo de programación en la que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño, un ejemplo básico de esto es separar la capa de datos de la capa de presentación al usuario.

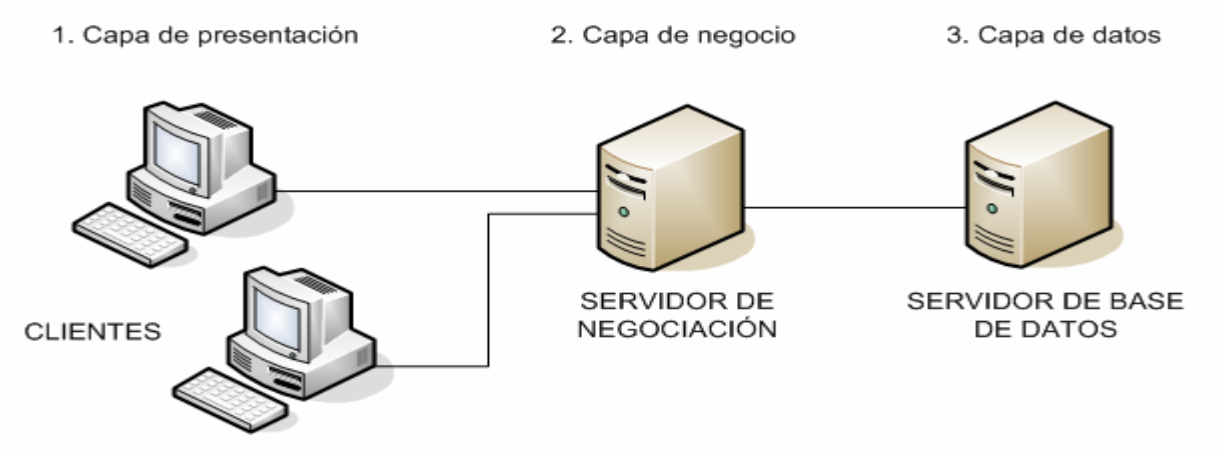


**Figura 1-1 Técnica de Programación 3 Capas.**

La ventaja principal de este estilo, es que el desarrollo se puede llevar a cabo en varios niveles y en caso de algún cambio sólo se ataca al nivel requerido sin tener que revisar entre código mezclado. Un buen ejemplo de este método de programación seria: Modelo de interconexión de sistemas abiertos

Además permite distribuir el trabajo de creación de una aplicación por niveles, de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, simplemente es necesario conocer la API que existe entre niveles.

En el diseño de sistemas informáticos actual se suele usar las arquitecturas multinivel o programación por capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

#### 1.1.2.4 Programación estructurada

La **programación estructurada** es una forma de escribir programación de ordenador de forma clara, para ello utiliza únicamente tres estructuras: secuencial, selectiva e iterativa; siendo innecesario y no permitiéndose el uso de la instrucción o instrucciones de transferencia incondicional.

**Ventajas de la Programación estructurada**

Con la programación estructurada, elaborar programas de computador sigue siendo una labor que demanda esfuerzo, creatividad, habilidad y cuidado. Sin embargo, con este estilo se obtienen las siguientes ventajas:

- Los programas son más fáciles de entender, ya que pueden ser leído de forma secuencial, sin necesidad de hacer seguimiento a saltos de línea (GOTO) dentro bloques de código para entender la lógica.

- La estructura del programa es más clara puesto que las instrucciones están más ligadas o relacionadas entre sí.

- Reducción del esfuerzo en las pruebas. El seguimiento de las fallas ("debugging") se facilita debido a la lógica más visible, por lo que los errores se pueden detectar y corregir más fácilmente.

- Reducción de los costos de mantenimiento.
- Programas más sencillos y más rápidos.
- Los bloques de código son auto explicativos, lo que apoya a la documentación.

**Desventajas de la programación estructurada.**

El principal inconveniente de este método de programación, es que se obtiene un único bloque de programa, que cuando se hace demasiado grande puede resultar problemático su manejo, esto se resuelve empleando la programación modular, definiendo módulos interdependientes programados y compilados por separado. Un método un poco más sofisticado es la programación por capas, en la que los módulos tienen una estructura jerárquica muy definida y se denominan capas.

#### 1.1.2.5 Programación dirigida por eventos.

La programación dirigida por eventos es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema o que ellos mismos provoquen.

El creador de un programa dirigido por eventos debe definir los eventos que manejarán su programa y las acciones que se realizarán al producirse cada uno de ellos, lo que se conoce como el manejador de evento. Los eventos soportados estarán determinados por el lenguaje de programación utilizado, por el sistema operativo e incluso por eventos creados por el mismo programador.

En la programación dirigida por eventos, al comenzar la ejecución del programa se llevarán a cabo las inicializaciones y demás código inicial y a continuación el programa quedará bloqueado hasta que se produzca algún evento. Cuando alguno de los eventos esperados por el programa tenga lugar, el programa pasará a ejecutar el código del correspondiente manejador de evento. Por ejemplo, si el evento consiste en que el usuario ha hecho click en el botón de play de un reproductor de películas, se ejecutará el código del manejador de evento, que será el que haga que la película se muestre por pantalla.

#### 1.1.2.6 Programación Orientada a Aspectos.

La Programación Orientada a Aspectos (POA) es un paradigma de programación relativamente reciente cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos. Gracias a la POA se pueden capturar los diferentes conceptos que componen una aplicación en entidades bien definidas, de manera apropiada en cada uno de los casos y eliminando las dependencias inherentes entre cada uno de los módulos. De esta forma se consigue razonar mejor sobre los conceptos, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables. Varias tecnologías con nombres diferentes se encaminan a la consecución de los mismos objetivos y así, el término POA es usado para referirse a varias tecnologías relacionadas como los métodos adaptivos, los filtros de composición, la programación orientada a sujetos o la separación multidimensional de competencias.

**Ventajas de la Programación Orientada a Aspectos.**

La programación orientada a aspectos, permite, de una manera comprensible y clara, definir aplicaciones considerando estos problemas. Por aspectos se entiende dichos problemas que afectan a la aplicación de manera horizontal y que este paradigma persigue el poder tenerlos de manera aislada de forma adecuada y comprensible, brindando la posibilidad de construir el sistema componiéndolos junto con el resto de los componentes.

#### 1.1.2.7 Programación con restricciones

La Programación con restricciones es un paradigma de la programación en informática, donde las relaciones entre las variables son expresadas en términos de restricciones (ecuaciones). Actualmente es usada como una tecnología de software para la descripción y resolución de problemas combinatorios particularmente difíciles, especialmente en las áreas de planificación y programación de tareas (calendarización).

Este paradigma representa uno de los desarrollos más fascinantes en los lenguajes de programación desde 1990 y no es sorprendente que recientemente haya sido identificada por la ACM (Asociación de Maquinaria Computacional) como una dirección estratégica en la investigación en computación.

Se trata de un paradigma de programación basado en la especificación de un conjunto de restricciones, las cuales deben ser satisfechas por cualquier solución del problema planteado, en lugar de especificar los pasos para obtener dicha solución.

La programación con restricciones se relaciona mucho con la programación lógica y con la investigación operativa. De hecho cualquier programa lógico puede ser traducido en un programa con restricciones y viceversa. Muchas veces los programas lógicos son traducidos a programas con restricciones debido a que la solución es más eficiente que su contraparte.

La diferencia entre ambos estriba principalmente en sus estilos y enfoques en el modelado del mundo. Para ciertos problemas es más natural (y por ende más simple) escribirlos como programas lógicos, mientras que en otros es más natural escribirlos como programas con restricciones.

El enfoque de la programación con restricciones se basa principalmente en buscar un estado en el cual una gran cantidad de restricciones sean satisfechas simultáneamente. Un problema se define típicamente como un estado de la realidad en el cual existe un número de variables con valor desconocido. Un programa basado en restricciones busca dichos valores para todas las variables.

Algunos dominios de aplicación de este paradigma son:

- Dominios boléanos, donde solo existen restricciones del tipo verdadero/falso.
- Dominios en variables enteras y racionales.
- Dominios lineales, donde solo se describen y analizan funciones lineales.
- Dominios finitos, donde las restricciones son definidas en conjuntos finitos.
- Dominios mixtos, los cuales involucran dos o más de los anteriores.

### 1.1.3 Programación Orientada a Objetos.

En este epígrafe se hará un estudio exhaustivo acerca de la [3]técnica de programación orientada a objetos, esta técnica se aplica al lenguaje seleccionado y está presente en la programación del sistema, pues se modeló el sistema lo mas orientado a objeto posible, por las ventajas que presenta dicha técnica.

**¿Qué es la Programación Orientada a Objetos?**

La OOP son un conjunto de técnicas que permiten incrementar enormemente el proceso de producción de software; aumentando drásticamente la productividad por un lado y brindando la posibilidad de abordar proyectos de mayor envergadura por otro. Usando estas técnicas, se asegura la re-usabilidad de código, es decir, “los objetos que hoy escribimos, si están bien escritos, nos servirán para siempre (Eidos 2000).

**Origen y Evolución de la Programación Orientada a Objetos.**

La programación orientada a objetos aparece a finales de los 60, pero es a principios de los 80 cuando con el lenguaje Smalltalk comienza un interés claro hacia este paradigma. La idea ocurrió para agrupar los diversos tipos de naves en diversas clases de objetos, siendo responsable cada clase de objetos de definir sus *propios* datos y comportamiento.

La programación orientada a objetos tomó posición como el estilo de programación dominante a mediados de los años ochenta, en gran parte debido a la influencia de C++, una extensión del lenguaje de programación C.

Las características de orientación a objetos fueron agregadas a muchos lenguajes existentes durante ese tiempo, incluyendo Ada, BASIC, Lisp, Pascal, y otros. Los lenguajes orientados a objetos "puros", por otra parte, carecían de las características de las cuales muchos programadores habían venido a depender.

Para saltar este obstáculo, se hicieron muchas tentativas para crear nuevos lenguajes basados en métodos orientados a objetos, pero permitiendo algunas características imperativas de maneras

[3] En algunas bibliografías se llama técnica de programación a los paradigmas de programación, y viceversa.

"seguras". El Eiffel de Bertrán Meyer fue un temprano y moderadamente acertado lenguaje con esos objetivos pero ahora ha sido esencialmente reemplazado por Java, en gran parte debido a la aparición de Internet, y a la implementación de la máquina virtual de Java en la mayoría de navegadores.

**Conceptos Básicos:**

En la programación orientada a objetos están presentes importantes conceptos para facilitar la representación en la solución de grandes problemas que se presenten a lo largo de la vida del software.

**Definición de Clase:**

Es simplemente una abstracción que se obtiene de la experiencia sensible. El ser humano tiende a agrupar seres o cosas -objetos- con características similares en grupos -clases-. Así, aun cuando existen por ejemplo multitud de vasos diferentes, se puede reconocer un vaso, incluso aun cuando ese modelo concreto de vaso no se haya visto nunca. El concepto de vaso es una abstracción de nuestra experiencia sensible.

Quizás el ejemplo más claro para exponer esto se aprecie en las taxonomías; los biólogos han dividido a todo ser (vivo o inerte) sobre la tierra en distintas clases. Si se toma como ejemplo una pequeña porción del inmenso árbol taxonómico:

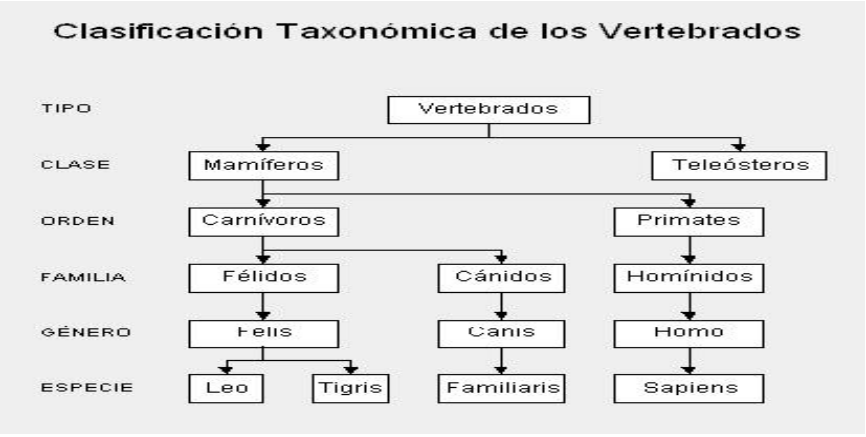


**Figura 1-1 Arbol Taxonomico de los Vertebrados.**

**Definición de Objeto:**

Según el Diccionario del Uso del Español de María Moliner (Ed. Gredos, 1983), en la tercera acepción del termino objeto se puede leer: "Con respecto a una acción, una operación mental, un sentimiento, etc., cosa de cualquier clase, material o espiritual, corpórea o incorpórea, real o imaginaria, abstracta o concreta, a la cual se dirigen o sobre la que se ejercen."

La definición de objeto, es mucho más fácil. En OOP, un objeto es un conjunto de datos y métodos (Eidos 2000).

Viendo desde otro punto de vista los datos son los llamados características o atributos, y los métodos son los comportamientos que pueden realizar.

En la figura siguiente se muestra un ejemplo de objeto, observando los datos y sus métodos:

| Datos | Métodos |
|---|---|
| Color | Desplazarse |
| Tamaño | Masticar |
| Peso | Digerir |
| Uñas retráctiles | Respirar |
| Colmillos | Secretar hormonas |
| Cuadrúpedo | Parpadear |
| etc. | etc. |

**Figura 1-2 Definición de Objeto con sus métodos.**

**Definición de Herencia:**

Los tipos se organizan de forma jerárquica (clases bases y derivadas, Superclases y subclases). La herencia proporciona un mecanismo simple mediante el cual se pueden definir unos tipos en función de otros, a los que añade sus características propias. Hay que distinguir entre herencia de interfaz y herencia de implementación (que aumenta el acoplamiento entre los módulos de un programa(MEYER).

Esta es la cualidad más importante de un sistema OOP, que proporciona mayor potencia y productividad, lo que permite ahorrar horas de codificación y de depuración de errores. Es por ello que no se puede considerar que un lenguaje sea OOP, si no incluye herencia.

En este ejemplo se muestra una jerarquía de clases generada por la herencia.

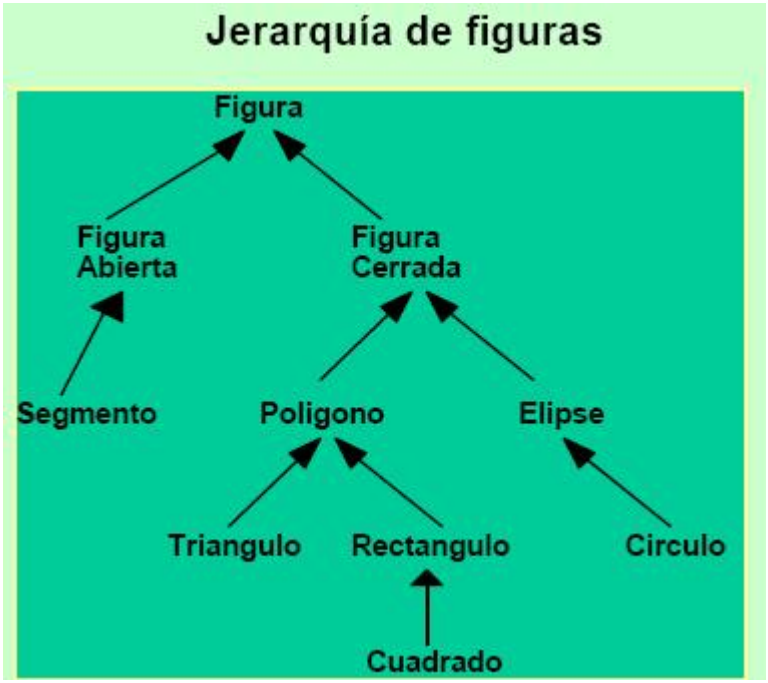


**Figura 1-3 Ejemplo de Jerarquía de Figuras.**

**Definición de Polimorfismo:**

Capacidad de usar un objeto sin saber su tipo exacto. Formalmente, el polimorfismo es la capacidad de que un elemento de código pueda denotar, en tiempo de ejecución, objetos de dos o más tipos distintos. (MEYER).

Puede también denotar objeto de muchas clases diferentes relacionados mediante alguna superclase común, de este modo todo objeto denotado por este nombre es capaz de responder a algún conjunto común de operaciones de diferentes maneras. Cualquier objeto denotado por este nombre puede responder a algún conjunto común de operaciones, lo contrario del polimorfismo es el monomorfismo, el cual se encuentra en todos los lenguajes que están a la vez fuerte y estáticamente ligados como Ada. (BOOCH 1996).

La sobrecarga puede ser considerada como un tipo especial de polimorfismo que casi todos los lenguajes de OOP incluyen.

Varios métodos (incluidos los "constructores", de los que se hablará más adelante) pueden tener el mismo nombre siempre y cuando el tipo de parámetros que recibe o el número de ellos sea diferente. De este modo, por ejemplo la clase File puede tener tantos métodos Write () como tipos de datos queramos escribir.

| | |
|---|---|
| File::Write( int i ); | Escribe un integer |
| File::Write( long l ); | Escribe un long |
| File::Write( float f ); | Escribe un flota |
| File::Write( string s ); | Escribe una cadena |
| File::Write( string s, boolean b ); | Escribe una cadena pasándola a mayúsculas |

**Figura 1-4 Ejemplo de Sobrecarga.**

**Definición de Encapsulación:**

Solo pueden accederse a los valores del tipo que define, o modificarlos mediante las operaciones abstractas definidas sobre ellos. La encapsulación hace referencia al ocultamiento de la información y a la capacidad de expresar la unidad formada por los valores y operaciones.(BOOCH 1996)

**Definición de Abstracción:**

Se reparan las propiedades lógicas de los datos de su representación o implementación.(BOOCH 1996).

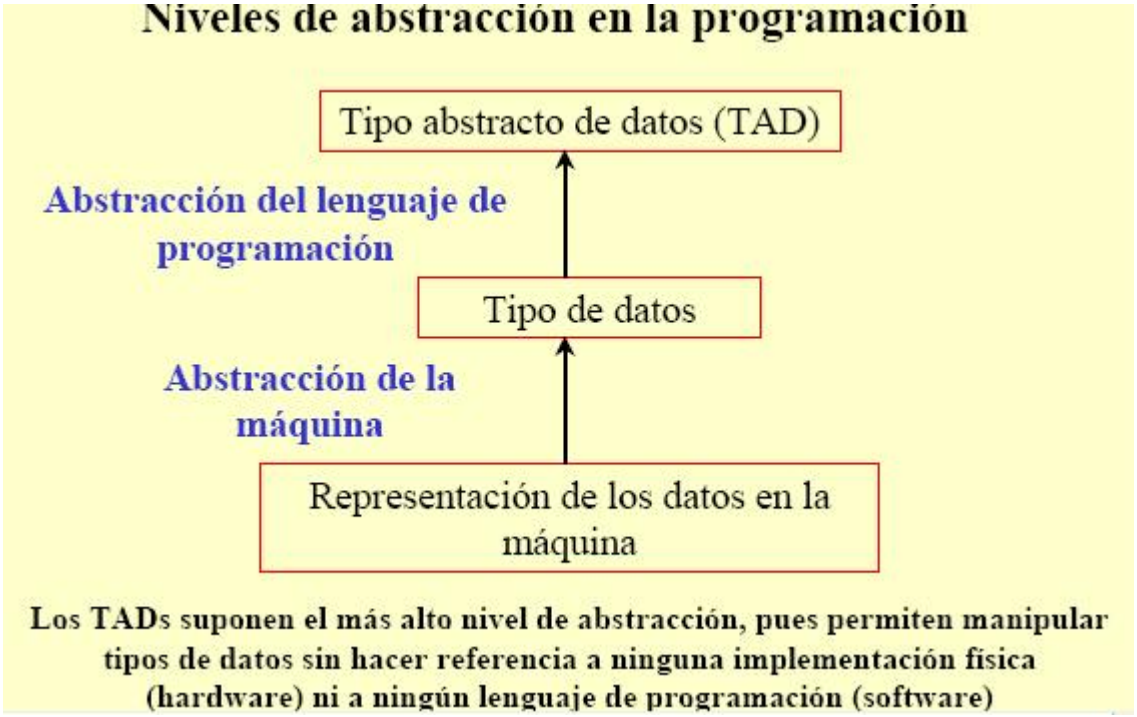


**Figura 1-5 Niveles de Abstracción**

### 1.1.3.1 Ventajas de la Programación Orientada a Objetos:

1. Es intuitiva, describe un problema en términos similares a los que utiliza la mente humana.
2. Permite construir soluciones más seguras y con un mantenimiento más sencillo.
3. Fomenta la reutilización y el trabajo en equipo.

4. Escalabilidad de las aplicaciones.

## 1.2 Lenguajes de Programación.

En esta sección del epígrafe se da una breve panorámica, acerca de los lenguajes de programación, su clasificación, tendencias de los mismos, así como se hace un análisis profundo del lenguaje usado para desarrollar el sistema.

Toda la historia de los lenguajes de programación se ha desarrollado en base a una sola idea conductora: hacer que la tarea de realizar programas para ordenadores sea cada vez lo más simple, flexible y portable posible.

En muchas de las ocasiones cuando se habla en términos de software, sistemas, aplicaciones, o simplemente de informática, se piensa rápidamente en una máquina operada por un especialista, y en el mejor de los casos, en un programador capaz de realizar lo inimaginable; olvidando por completo que todo lo que se es capaz de realizar en una computadora esta sustentado gracias al conocimiento, habilidades y dominio que tenga dicho programador sobre uno o varios lenguajes de programación los cuales son el pilar fundamental para el flujo de instrucciones entre el programador y su PC.

**¿Qué es un lenguaje de programación?**

Un lenguaje de programación, es algo de lo cual todos tienen una idea: un conjunto de instrucciones entendibles directamente o traducibles al lenguaje del ordenador con el que se trabaje; combinando estas instrucciones realizamos programas (Eidos 2000).

Estos lenguajes están formados por un **Léxico** que no es mas que un conjunto de símbolos permitidos y una **Sintaxis** o reglas que indican cómo realizar las construcciones del lenguaje además de otras reglas que permiten determinar el significado de cualquier construcción del lenguaje denominadas **Semántica**, las cuales en su unión forman el lenguaje como tal.

### 1.2.1 Clasificación de los lenguajes de programación.

Hoy día debido al alto crecimiento y al gran avance de la informática y las comunicaciones, se han desarrollado gran cantidad de lenguajes con diferentes características y para diversos usos, por lo que se pueden clasificar en:

**Según el nivel de abstracción:**

- **Programación de bajo nivel:**

  Es el tipo de lenguaje que cualquier computadora es capaz de entender. Se dice que los programas escritos en forma de ceros y unos están en lenguaje de máquina, porque esa es la versión del programa que la computadora realmente lee y sigue.

- **Programación de medio nivel:**

Son lenguajes de programación que se asemejan a las lenguas humanas usando palabras y frases fáciles de entender. En un lenguaje de bajo nivel cada instrucción corresponde a una acción ejecutable por el ordenador, mientras que en los lenguajes de alto nivel una instrucción suele corresponder a varias acciones.

- **Programación de alto nivel:**

**Según la forma de ejecución:**

- **Lenguajes compilados**
- **Lenguajes interpretados**

### 1.2.2 Algunos Lenguajes de Programación

- ABAP
- ABC
- Ada
- ActionScript
- Afnix
- ALGOL
- APL
- ASP
- ASP.NET
- AWK
- B
- BASIC
- BCPL
- Befunge
- Boo
- C
- C++
- C#
- Caml
- Clipper
- CLIPS
- CLU
- COBOL
- CORAL
- D
- Delphi
- DIV
- Dylan
- Eiffel
- Erlang
- Ensamblador
- Extended ML
- Euphoria
- Fénix
- Flow-Matic
- Forth
- FORTRAN
- Gambas
- GML
- GRAFCET
- FP
- Haskell
- Icon
- Inform
- INTERCAL
- ISWIM
- J
- Java
- JavaScript
- Joy
- KWC
- LADDER
- Lexico
- Lingo
- Lisp
- Lua
- MAGIC
- Mainsail
- Mesa
- Miranda
- ML
- Modula
- Modula-2
- Modula-3
- Natural
- NetREXX
- Oberon
- Object REXX
- Objective-C
- Ocaml
- Occam
- Oz
- Pascal
- Parlog
- Perl
- PHP
- PL/1
- Plankalkül
- PostScript
- PowerBuilder
- Prolog
- Python
- Rapid
- REXX
- RPN
- RPG
- Ruby
- Sail
- Sather
- Scheme
- Scriptol
- Seed7
- Self
- Sh
- Simula
- Smalltalk
- Snobol
- SPARK
- Squeak
- SR
- Standard ML
- TI-Basic
- TCL
- VBA
- Visual Basic
- Visual C++
- Visual DialogScript
- Visual Foxpro
- Yurix
- ZPL

**Figura 1-6 Lenguajes De Programación Existentes**

### 1.2.3 Selección del Lenguaje y el Entorno de Desarrollo.

En esta sección de este epígrafe se trata de justificar modestamente el lenguaje y la plataforma usada para la implementación del sistema.

Para la implementación del sistema se escogió la Plataforma .NET, ¿Por que la Plataforma .NET?, se eligió esta plataforma puesto que para el desarrollo de la aplicación se necesitaba de una plataforma moderna que cumpliera las siguientes condiciones.

- Tenga soporte para la Programación Orientada a Objetos.
- Sea popular, pues de esta forma se pueden encontrar y soluciones ya implementadas.

- Sea posible utilizar las últimas técnicas de desarrollo ágil y existan herramientas implementadas para hacerlo.
- Sea factible la implementación de los patrones de diseño.
- Sea Multiplataforma.

Hay que destacar que el cliente no hizo un requerimiento específico sobre la definición de una plataforma, las dos grandes opciones en competencia, que cumplían estas condiciones, eran sin lugar a dudas Java2ee y .NET Framework, se escogió la segunda opción debido a varios factores, que se mencionan a continuación.

- La velocidad de crecimiento y maduración de.NET, es realmente exponencial con respecto a la de Java.
- La facilidad de aprendizaje y acceso a tutoriales es mucho mayor a la de Java.
- La estandarización de C# y el CLI (common lenguaje infrastructure) permitió que Proyectos como Sharp Develop, desarrollaran un IDE muy completo, similar al Visual Studio .NET pero de código abierto.
- Relacionado con el CLI se desarrollo el proyecto MONO que permite ejecutar aplicaciones de .NET en distintas plataformas (Linux, Mac, Windows, etc.) sin necesidad de recompilar, directamente se pasan los mismos ejecutables de Windows al runtime de Mono y listo (se encuentra en un estado productivo y por la gran adhesión e interés comercial llegara a un estado de madurez muy pronto.
- El conocimiento sobre dicha tecnología o plataforma que se escogiera, para ahorrar tiempo de estudio y capacitación.

Por todas estas razones se selecciono la Plataforma. NET (mas información ver epígrafe 1.3.1) y lenguaje de programación C# (información mas detallada en epígrafe 1.2.3.2), para el desarrollo del sistema cumpliendo con los factores mencionados anteriormente, y específicamente el lenguaje C#,

dentro de la plataforma por el conocimiento del mismo, por poseer las estructuras de programación necesarias para poder implementar lo planteado en la fase de análisis y diseño de este sistema, como son las interfaces de implementación, los eventos y delegados, listas genéricas entre otros.

#### 1.2.3.1 El lenguaje C#.

En esta sección de este epígrafe se trata de justificar el lenguaje y el entorno de desarrollo que se utiliza para la implementación de este sistema. Se eligió este lenguaje ya que posee en primer lugar las estructuras de programación necesarias para poder implementar lo planteado en la fase de análisis y diseño de este sistema como son: un lenguaje que fuera totalmente orientado a objetos, con interfaces de implementación, que diera soporte al manejo de eventos y delegados de la manera mas orientada a objeto posible, y que fuera un lenguaje del dominio de los desarrolladores, en el siguiente epígrafe se darán a conocer características importantes de este lenguaje.

#### 1.2.3.2 Características de C#.

- -Sencillez: C# elimina muchos elementos que otros lenguajes incluyen y que son innecesarios en .NET. Por ejemplo: el código escrito en C# es auto contenido (no necesita de ficheros adicionales al propio fuente tales como ficheros de cabecera) y el tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile (no como en C++), lo que facilita la portabilidad del código.

- -Modernidad: C# incorpora en el propio lenguaje elementos que a lo largo de los años ha ido demostrándose son muy útiles para el desarrollo de aplicaciones: tipo básico decimal (128 bits), instrucción foreach, tipo básico string, etc.

- -Orientación a Objetos: C# no admite ni funciones ni variables globales, sino que todo el código y datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código. Soporta todas las características propias del paradigma de programación Orientado a objetos: polimorfismo, la encapsulación y la herencia.

- Orientación a componentes: La sintaxis de C# permite definir cómodamente propiedades (similares a campos de acceso controlado), eventos (asociación controlada de funciones de respuesta a notificaciones) o atributos (información sobre un tipo o sus miembros).

- Gestión automática de memoria. Todo lenguaje de .NET tiene a su disposición el recolector de basura del CLR.

- Seguridad de tipos. # incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente.

- No se pueden usar variables no inicializadas: Se comprueba que los accesos a los elementos de una tabla se realicen con índices que se encuentren dentro del rango de la misma.

- C# incluye delegados, que son similares a los punteros a funciones de C++ pero siguen un enfoque orientado a objetos: pueden almacenar referencias a varios métodos simultáneamente, y se comprueba que los métodos a los que apunten tengan parámetros y valor de retorno del tipo indicado al definirlos.

- Instrucciones seguras. En C# se han impuesto una serie de restricciones en el uso de las instrucciones de control más comunes (toda condición está controlada por una expresión condicional, todo caso de un switch ha de terminar en un break o goto, etc.).

- Extensibilidad de operadores: C# permite redefinir el significado de la mayoría de los operadores incluidos los de conversión, tanto para conversiones implícitas como explícitas cuando se apliquen a diferentes tipos de objetos.

- Extensibilidad de modificadores: C# ofrece, a través del concepto de atributos, la posibilidad de añadir, a los meta datos del módulo resultante de la compilación de cualquier fuente, información adicional a la generada por el compilador que luego podrá ser consultada en tiempo ejecución a través de la biblioteca de Reflexión de .NET. Esto, que más bien es una característica propia de la Plataforma .NET y no de C#, puede usarse como un mecanismo para definir nuevos modificadores.

- Versionable: C# incluye una política de versionado que permite crear nuevas versiones de tipos sin temor a que la introducción de nuevos miembros provoque errores difíciles de detectar en tipos hijos previamente desarrollados y ya extendidos con miembros de igual nombre a los recién introducidos.

### 1.2.4 Tendencias de los lenguajes de programación

Las generaciones de los lenguajes de programación, ha seguido dos tendencias fundamentales:

- El desplazamiento del centro de atención de la programación de pequeños sistemas hechos por una persona a los medianos y grandes sistemas, hechos por equipos de desarrollo de software
- La evolución de los lenguajes, un desplazamiento desde los lenguajes que dicen al computador qué hacer, o lenguajes imperativos, hacia lenguajes que describen las abstracciones clave en el dominio del problema (lenguajes declarativos).

Lenguajes de primera generación (1954-1958)(BOOCH):

- FORTRAN 1 Expresiones matemáticas.
- ALGOL 58 Expresiones matemáticas.
- Flowmatic Expresiones matemáticas.
- IPLV Expresiones matemáticas.

Los lenguajes de la primera generación se utilizaron principalmente para aplicaciones científicas y de ingeniería, y su vocabulario fue matemático casi por completo. Así, los lenguajes como el FORTRAN 1 se desarrollaron para que el programador pudiera escribir fórmulas matemáticas, liberándole de esta forma de algunas de las complicaciones del lenguaje ensamblador o del lenguaje máquina. Esta primera generación de lenguajes de alto nivel representó por lo tanto un paso de acercamiento al espacio del problema, y un paso de alejamiento de la máquina que había debajo.

Lenguajes de segunda generación (1959-1961)(BOOCH):

- FORTRAN 2 Subrutinas, compilación separada.
- ALGOL 60 Estructura en bloques, tipos de datos.

- COBOL Descripción de datos, manejo de ficheros.
- Lisp Procesamiento de listas**,** punteros, recolección de basura.

Los lenguajes de segunda generación, marcaron las abstracciones algorítmicas. Por esta época, las máquinas eran cada vez más y más potentes, y el abaratamiento en la industria de los computadores significó que podía automatizarse una mayor variedad de problemas, especialmente para aplicaciones comerciales. En este momento, lo principal era decirle a la máquina lo que debía hacer: lee primero estas fichas personales, ordénalas después, y a continuación imprime este informe. Una vez más, esta nueva generación de lenguajes de alto nivel acercaba a los desarrolladores aun más hacia el espacio del problema y los alejaba de la máquina subyacente.

Lenguajes de tercera generación (1962-1970)(BOOCH):

- PL/ 1 FORTRAN + ALGOL + COBOL.
- ALGOL 68 Sucesor riguroso del ALGOL 60.
- Pascal Sucesor sencillo del ALGOL 60.
- Simula Clases, abstracción de datos.

En los lenguajes de la tercera generación a finales de los sesenta se observo un gran avance, especialmente con la llegada de los transistores y la tecnología de circuitos integrados, el coste del hardware de los computadores había caído de forma dramática, pero su capacidad de procesamiento había crecido casi exponencialmente. Ahora podían resolverse problemas mayores, pero eso exigía la manipulación de más tipos de datos. Así, lenguajes como el ALGOL 60 y posteriormente el Pascal evolucionaron soportando abstracción de datos. El programador podía describir el significado de clases de datos relacionadas entre sí (su tipo) y permitir que el lenguaje de programación apoyase estas decisiones de diseño. Esta generación de lenguajes acercó de nuevo el software un paso hacia el dominio del problema, y lo alejó otro paso de la máquina.

El Hueco Generacional (1970- 1980):

Los setenta ofrecieron un frenesí de actividad investigadora en materia de lenguajes de programación, con el resultado de la creación de diferentes lenguajes cada uno con sus propios dialectos.

“Se inventaron muchos lenguajes diferentes, pero pocos perduraron” (PETER 1981). En gran medida, la tendencia a escribir programas más y más grandes puso de manifiesto las deficiencias de los lenguajes más antiguos; así, se desarrollaron muchos nuevos mecanismos lingüísticos para superar estas limitaciones. Sólo algunos de estos lenguajes han sobrevivido, mientras que otros como Fred, Chaos o Tranquil, se quedaron en el camino, y encontrar literatura de ellos se hace bastante difícil. Sin embargo, muchos de los conceptos que introdujeron encontraron su camino en otros sucesores. Así, hoy en día existen Smalltalk (un sucesor revolucionario de Simula), Ada (un sucesor del ALGOL 68 y Pascal, con contribuciones de Simula, Alphard y CLU), CLOS (que surgió del Lisp**,** LOOPS y Flavors), C++ (derivado de C y Simula), y Eiffel (derivado de Simula y Ada).

## 1.3 Plataforma, Herramientas de Desarrollo y Frameworks.

En epígrafes anteriores se explico de forma breve la selección del lenguaje de programación y la plataforma para la implementación del sistema, y se dieron a conocer algunas de sus características esenciales, en esta sección de este epígrafe se explica de forma detallada la plataforma, las herramientas de desarrollo, y el framework usado por la herramienta de desarrollo.

### 1.3.1 Plataforma .NET

Es un conjunto de tecnologías dispersas, que en muchos casos ya existían, que Microsoft ha integrado en una plataforma común con el objetivo de facilitar el desarrollo de este nuevo tipo de servicios de tercera generación.(HORALLO).

Estos son los pilares de esta nueva plataforma:

• Integración: Proporcionar Mecanismos para que una empresa pueda ofrecer servicios a otras empresas o clientes de una forma sencilla y rápida. En general, este tipo de servicios se suelen denominar B2B: Business to Business y B2C: Business to Client.

• Nuevos dispositivos: La forma más común de acceso a Internet hasta ahora ha sido el ordenador personal con sus limitaciones de movilidad. Pero recientemente han ido apareciendo una serie de dispositivos que permiten el acceso a servicios Internet de forma rápida y directa, como por ejemplo agendas electrónicas, teléfonos móviles, WebTV, videoconsolas, etc. Esto supone un cambio radical en la

forma de acceder a este tipo de servicios. Con estos objetivos, Microsoft .NET es una plataforma para construir, ejecutar y experimentar la tercera generación de aplicaciones distribuidas, que consiste en los siguientes elementos:

- Un modelo de programación basado en XML.
- Un conjunto de servicios Web XML, como Microsoft .NET My Services para facilitar a los desarrolladores integrar estos servicios.
- Un conjunto de servidores que permiten ejecutar estos servicios (como .NET Enterprise Servers).
- Para facilitar esta integración y el desarrollo de este Software en el cliente para poder utilizar estos servicios (como Windows XP, agendas electrónicas, etc.).

Herramientas para el Desarrollo como Visual Studio.Net. En la siguiente figura se muestran los siguientes elementos que componen la plataforma .NET.


**Figura 1-7 Elementos que componen la plataforma.NET**

Una parte importante de esta plataforma es el software de los dispositivos clientes y servidores, que ha sido el mercado habitual de Microsoft. Para los dispositivos clientes, Microsoft planea integrar .NET en cualquier dispositivo imaginable, como PCs con Windows, agendas electrónicas con Pocket PC, teléfonos móviles, su consola de videojuegos X-Box, en WebTV, etc. Esto supone para las empresas aumentar el numero de potenciales clientes que puedan utilizar su servicios (ya no están limitados al PC). Para poder ejecutar estos servicios, Microsoft introduce una serie de software englobado dentro de los .NET Enterprise Servers, como es el Application Center, Commerce Server, etc.

Estos servicios se ofrecerán al cliente a través de distintos canales, lo que Microsoft ha denominado Experiencias de Usuarios. Así, Microsoft ha pensado que MSN sea el canal para clientes domésticos y Central es el canal de comercio electrónico para empresas.

### 1.3.2 Framework o Arquitectura.NET

Una definición general de la arquitectura .NET podría ser la siguiente [4]: "Una plataforma independiente del lenguaje para el desarrollo de servicios Web". (HORALLO)

La arquitectura .NET (.NET Framework) es el modelo de programación de la plataforma .NET para construir y ejecutar los servicios .NET. El objetivo de esta arquitectura es la de reducir la complejidad en el desarrollo de este tipo de aplicaciones, permitiendo a los desarrolladores centrarse en escribir la lógica específica del servicio a desarrollar. Esta arquitectura está compuesta por librerías tal como muestra la


**Figura 1-3-1 Composición de la Arquitectura de .NET**

El ejecutivo del lenguaje común (CLR: Common Languaje Runtime) es un soporte que permite ejecutar los servicios .NET en cualquier máquina que lo disponga. Esta basado en la idea de Java, que también tiene un módulo de ejecución independiente del sistema operativo donde se vaya a ejecutar. La gran diferencia con Java es que este ejecutivo es multilenguaje, esto es, no está limitado a un único lenguaje como Java. Esto permite al desarrollador utilizar una amplia variedad de lenguajes como C++, Visual Basic y C#.

Las librerías básicas proporcionan una serie de funcionalidades que son necesarias a la hora de desarrollar los servicios Web. Las clases básicas gestionan las operaciones más básicas como las comunicaciones, entrada/salida, seguridad, etc. Las clases XML y de datos gestionan el acceso a base de datos y la gestión de datos en XML. El objetivo de las librerías Servicios Web XML es la de dar soporte para el desarrollo de aplicaciones distribuidas que ofrezcan servicios XML a otras entidades. Las Web forms permiten desarrollar la parte gráfica de una aplicación para la Web, mientras las Windows Forms están orientadas a implementar la parte gráfica de las aplicaciones clásicas para Windows.

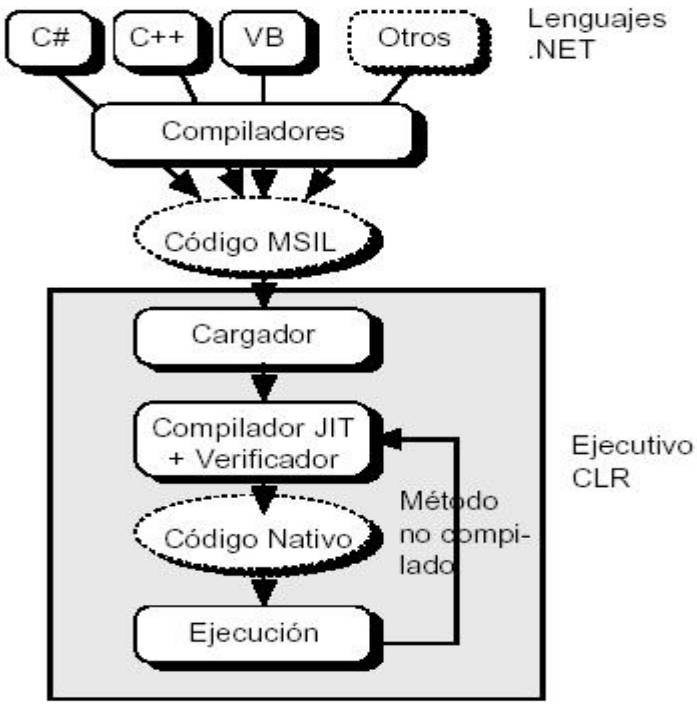


**Figura 1-1-2 Lenguajes .NET**

Los compiladores producen código MSIL (Microsoft Intermediate Language), que es un lenguaje intermedio que se puede ejecutar en la máquina virtual. Este código no es interpretado por el ejecutivo, sino que es compilado de nuevo en tiempo de ejecución (JIT: Just in Time) al código nativo de la máquina. Este código compilado no se ejecuta independientemente sino dentro de este ejecutivo. Esto se denomina código manejado, lo cual permite que el ejecutivo controle ciertos aspectos de la aplicación que ejecuta como son seguridad, gestión de memoria, compartición de datos, etc. Aparte de Microsoft, existe actualmente un proyecto de Software Abierto para implementar toda esta arquitectura en Linux que se denomina MONO [5]. El objetivo es portar el ejecutivo CLR a Linux e implementar un compilador C#. Esto es muy interesante, ya que rompería uno de los objetivos de Microsoft, que era que la plataforma .NET sólo se ejecutase en sus sistemas operativos Windows.

### 1.3.3 Herramienta de Desarrollo Visual Studio.NET.

El objetivo principal de este entorno de desarrollo es la de simplificar el desarrollo de aplicaciones Windows y servicios Web permitiendo la elección del lenguaje de programación más adecuado (Visual Basic, C++ o C#).

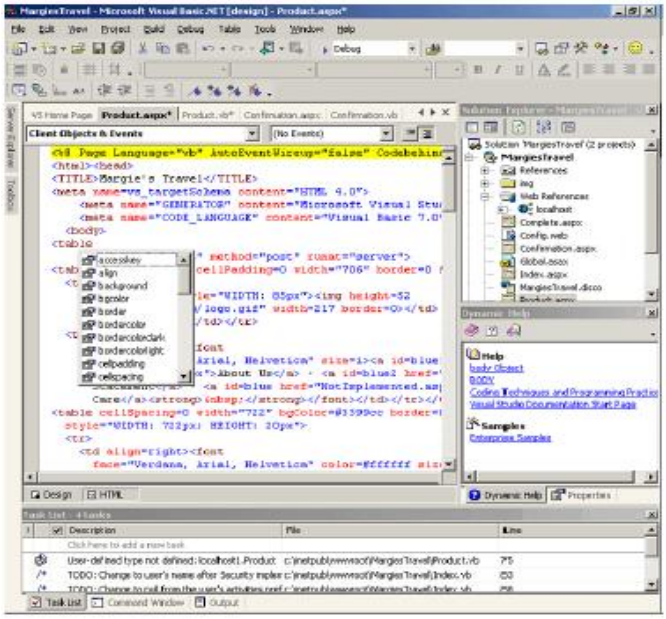

**Figura 1-3-3 Ambiente.NET**

Aparte de poder elegir el lenguaje de programación hay que decidir qué tipo de aplicación se va a desarrollar, y en este caso se distingue ya entre las aplicaciones Windows tradicionales y los servicios Web, todo en el mismo entorno.

Visual Studio .NET sigue teniendo muchas de las características de versiones anteriores, como el entorno de edición, compilación y depuración integrado, gestión de proyectos complejos, diseño con notación UML. Pero con lo que respecta a la plataforma .NET se enumeran las principales características o mejoras al desarrollo que proporciona este entorno de desarrollo:

• Ejecutivo común: Como se ha comentado antes todos los lenguajes en la arquitectura .NET utilizan un módulo de ejecución común con librerías comunes. Con esto se termina con los distintos módulos de ejecución para cada lenguaje (como vbrun.dll para Visual Basic o msvc42.dll para Visual C++).

• Clases unificadas: Hasta ahora, cada lenguaje tenía su conjunto de clases o librerías para poder desarrollar programas Windows (las MFC en C++, VB Framework en Visual Basic). Esto implicaba que para cambiar de lenguaje, era necesario, aparte de conocer la sintaxis del lenguaje, conocer las librerías a utilizar. En la nueva plataforma .NET estas librerías o clases son comunes para todos los lenguajes, con lo que los desarrolladores no tienen que aprender una nueva librería cuando cambian de lenguaje.

• Integración multilenguaje: Además de los puntos anteriores, se incluye la posibilidad de llamada a métodos de otros objetos desarrollados en otros lenguajes e incluso su herencia. Esto permite desarrollar objetos en el lenguaje más apropiado para el problema a solucionar.

• ASP.NET: Esta librería proporciona un nuevo modelo para la creación de aplicaciones Web. Esto permite crear gráficamente páginas Web utilizando una serie de controles (desde el tipo campo de edición, hasta calendarios). Estos servicios se compilarán en el servidor y al cliente se le genera en tiempo de ejecución la página HTML apropiada para el navegador que se utilice.

• ADO.NET: Esta librería proporciona un acceso común a los datos, ya sea en bases de datos o XML.

• Plataforma abierta: A este entorno de desarrollo se le pueden añadir herramientas o nuevos lenguajes de programación, de tal forma que estén perfectamente integrados en Visual Studio.

De esta forma se van a poder utilizar distintos lenguajes de programación como Eiffel, Perl, Java e incluso lenguajes tan venerables como Cobol y Fortran. Además de todo este soporte, Microsoft ha desarrollado un lenguaje nuevo de programación basado en C++ denominado C# (C sharp).

## 1.4 Metodologías de desarrollo de Software

Todo desarrollo de software es riesgoso y difícil de controlar, pero si no se aplica una metodología de por medio, lo que se obtiene son clientes insatisfechos con el resultado y desarrolladores aún más insatisfechos. Sin embargo, muchas veces no se toma en cuenta el utilizar una metodología adecuada, sobre todo cuando se trata de proyectos pequeños de dos o tres meses. Lo que se hace con este tipo de proyectos es separar rápidamente el aplicativo en procesos, cada proceso en funciones, y por cada función determinar un tiempo aproximado de desarrollo(PRESSMAN).

Cuando los proyectos que se van a desarrollar son de mayor envergadura, ahí si toma sentido en basar su desarrollo en una metodología, y empezar a buscar cual sería la más apropiada para nuestro caso. Lo cierto es que muchas veces no se encuentra la más adecuada y se termina por hacer o diseñar la propia metodología, algo que por supuesto no esta mal, siempre y cuando cumpla con el objetivo (JACOBSON 1992). Muchas veces se realiza el diseño del software de manera rígida, con los requerimientos que el cliente solicitó, de tal manera que cuando el cliente en la etapa final (etapa de prueba), solicita un cambio se hace muy difícil realizarlo, pues de hacerlo, se alteran muchas cosas que no se había previsto, y es justo éste, uno de los factores que ocasiona un atraso en el proyecto y por tanto la incomodidad del desarrollador por no cumplir con el cambio solicitado y el malestar por parte del cliente por no tomar en cuenta su pedido. Obviamente para evitar estos incidentes se debe haber llegado a un acuerdo formal con el cliente, al inicio del proyecto, de tal manera que cada cambio o modificación no perjudique al desarrollo del mismo.

Por experiencia, muchas veces los usuarios finales, se dan cuenta de las cosas que dejaron de mencionar, recién en la etapa final del proyecto, pese a que se les mostró un prototipo del software en la etapa inicial del proyecto. Los proyectos en problemas son los que salen del presupuesto, tienen importantes retrasos, o simplemente no cumplen con las expectativas del cliente.

Para dar una idea qué metodología utilizar y cual se adapta más a nuestro medio, se hará mención a tres de las que se consideran las más importantes, tal como: RUP, XP y MSF.

Rational Unified Process (RUP), llamada así por sus siglas en inglés Rational Unified Process, divide en 4 fases el desarrollo del software:

Inicio, El Objetivo en esta etapa es determinar la visión del proyecto.

Elaboración, En esta etapa el objetivo es determinar la arquitectura óptima.

Construcción, En esta etapa el objetivo es llevar a obtener la capacidad operacional inicial.

Transición, El objetivo es llegar a obtener el release del proyecto.

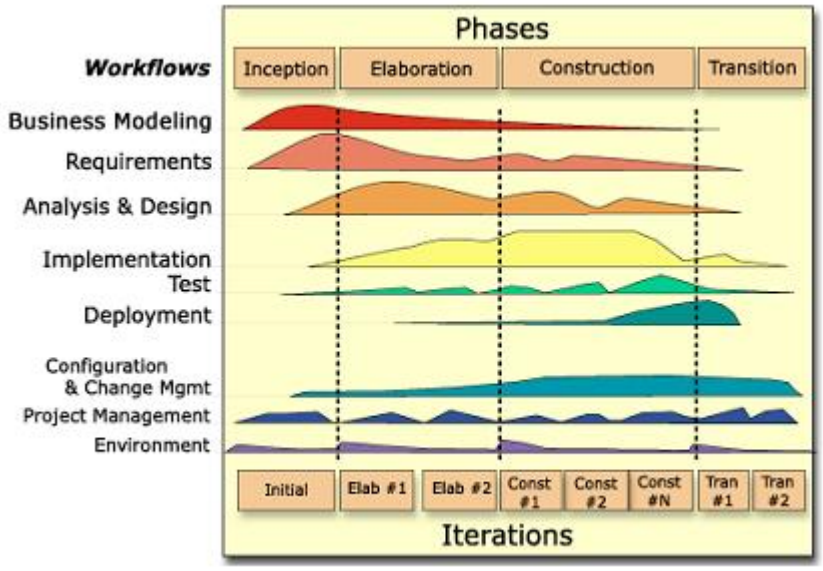


**Figura 1-4-1 Fases e Iteraciones de RUP**

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los Objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes. Vale mencionar que el ciclo de vida que se desarrolla por cada iteración, es llevada bajo dos disciplinas:

**Disciplina de Desarrollo**

**Ingeniería de Negocios**: Entendiendo las necesidades del negocio.

**Requerimientos**: Trasladando las necesidades del negocio a un sistema automatizado.

**Análisis y Diseño:** Trasladando los requerimientos dentro de la arquitectura de software.

**Implementación:** Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.

**Pruebas:** Asegurándose que el comportamiento requerido es el correcto y que todo los solicitado esta presente.

**Disciplina de Soporte**

Configuración y administración del cambio: Guardando todas las versiones del proyecto.

**Administrando el proyecto**: Administrando horarios y recursos.

**Ambiente:** Administrando el ambiente de desarrollo.

**Distribución:** Hacer todo lo necesario para la salida del proyecto

Es recomendable que a cada una de estas iteraciones se les clasifique y ordene según su prioridad, y que cada una se convierte luego en un entregable al cliente. Esto trae como beneficio la retroalimentación que se tendría en cada entregable o en cada iteración.

Los elementos del RUP son:

- Actividades, Son los procesos que se llegan a determinar en cada iteración.
- Trabajadores, Vienen hacer las personas o entes involucrados en cada proceso.
- Artefactos, Un artefacto puede ser un documento, un modelo, o un elemento de modelo.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

**Extreme Programing (XP)**

Es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizada para proyectos de corto plazo fundamentalmente, la cual consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

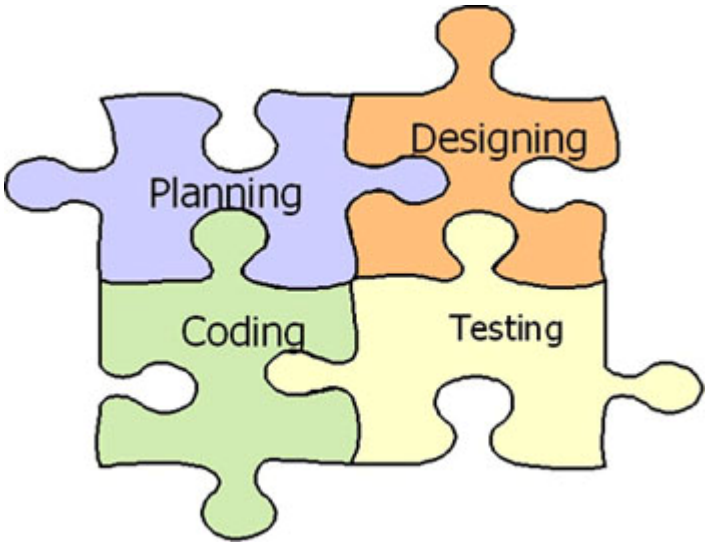


**Figura 1-4-2 Metodología Extreme Programing**

Características de XP, la metodología se basa en:

Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que adelantándonos en algo hacia el futuro, podamos hacer pruebas de las fallas que pudieran ocurrir. Es como si nos adelantáramos a obtener los posibles errores. Refabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa.

¿Qué es lo que propone XP?

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantivo del proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.
- El cliente o el usuario se convierten en miembro del equipo.

**Derechos del Cliente**

- Decidir que se implementa.
- Saber el estado real y el progreso del proyecto.
- Añadir, cambiar o quitar requerimientos en cualquier momento.
- Obtener lo máximo de cada semana de trabajo.
- Obtener un sistema funcionando cada 3 o 4 meses.

**Derechos del Desarrollador**

- Decidir como se implementan los procesos.
- Crear el sistema con la mejor calidad posible.
- Pedir al cliente en cualquier momento aclaraciones de los requerimientos.

- Estimar el esfuerzo para implementar el sistema.
- Cambiar los requerimientos en base a nuevos descubrimientos.

**Lo fundamental en este tipo de metodología es:**

- La comunicación, entre los usuarios y los desarrolladores.
- La simplicidad, al desarrollar y codificar los módulos del sistema.
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.

**Microsoft Solution Framework (MSF)**

Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.


**Figura 1- 4- 1Metodología MSF**

MSF tiene las siguientes características:

- Adaptable**:** es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- Escalable**:** puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.

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

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el modelo de Aplicación.

Modelo de Arquitectura del Proyecto: Diseñado para acortar la planificación del ciclo de vida. Este modelo define las pautas para construir proyectos empresariales a través del lanzamiento de versiones.

Modelo de Equipo: Este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles.

Modelo de Proceso*:* Diseñado para mejorar el control del proyecto, minimizando el riesgo, y aumentar la calidad acortando el tiempo de entrega. Proporciona una estructura de pautas a seguir en el ciclo de vida del proyecto, describiendo las fases, las actividades, la liberación de versiones y explicando su relación con el Modelo de equipo.

Modelo de Gestión del Riesgo: Diseñado para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.

Modelo de Diseño del Proceso: Diseñado para distinguir entre los objetivos empresariales y las necesidades del usuario. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible a través de un enfoque iterativo. Las fases de diseño conceptual, lógico y físico proveen tres perspectivas diferentes para los tres tipos de roles: los usuarios, el equipo y los desarrolladores.

Modelo de Aplicación: Diseñado para mejorar el desarrollo, el mantenimiento y el soporte, proporciona un modelo de tres niveles para diseñar y desarrollar aplicaciones software. Los servicios utilizados en este modelo son escalables, y pueden ser usados en un solo ordenador o incluso en varios servidores.

Con este análisis hecho anteriormente sobre metodologías y procesos de desarrollo de software se afirma que:

- La Metodología RUP es más adaptable para proyectos de largo plazo.
- La Metodología XP en cambio, se recomienda para proyectos de corto plazo.
- La Metodología MSF se adapta a proyectos de cualquier dimensión y de cualquier tecnología.

Para el desarrollo de este sistema se escogió RUP y la notación UML debido a el alcance que se espera tenga este proyecto y la documentación que se necesita de cada uno de sus artefactos, los cuales debe estar bien definidos y documentado para facilitar el posterior desarrollo o a ampliación del sistema con nuevos requisitos en determinados intervalos de tiempo. También jugó un papel fundamental el conocimiento de los integrantes del equipo de desarrollo y demás colaboradores cercanos sobre esta metodología, ahorrando consigo tiempo de estudio e investigación. Por otro lado vale aclarar que aunque el proyecto no es grande, su desarrollo no es a corto plazo, pues a este se le incorporan constantemente nuevos requisitos funcionales pedidos por el cliente, lo que mantiene activo el proyecto en un periodo de tiempo considerable de desarrollo, aunque liberando versiones listas para usar, en sus distintas iteraciones.

# CAPÍTULO 2: Descripción y Análisis de la solución propuesta.

**Introducción:**

En este capitulo se plantea la solución propuesta para llevar a cabo la implementación del sistema, usando métodos sofisticados y eficientes de programación, como son los algoritmos de procesamiento de información, se hace mención además al diseño de clases propuesto por el analista, así como a los componentes o clases reutilizables, implementadas en el sistema y se da para finalizar una breve descripción de las clases propuestas por el analista.

## 2.1 Valoración crítica Del diseño propuesto por el analista.

El diseño de clases propuesto por el analista, esta compuesto por diversos subsistemas (ver figura 2-1) entre los que se encuentran:

**Subsistema de Reporte:** Es el encargado de gestionar todo lo concerniente a los reportes en la aplicación.

**Subsistema de Comunes:** En este subsistema se encuentran todos las clases que serán comunes en el sistema, estas clases son de gran importancia, para los reportes, para hacer la búsqueda y análisis de las patentes descargadas por el sistema, y de las que se encuentran en la base de datos local.

**Subsistema de Presentación**: En este subsistema se encuentran todo lo concerniente a interfaces de usuario, componentes implementados por el equipo de desarrollo etc.

**Subsistema de Búsqueda:** Este subsistema de búsqueda es uno de lo más importantes en la realización del proyecto, pues el mismo es el encargado de gestionar todo lo concerniente a la búsqueda y descarga de las patentes en las bases de datos de Internet.

**Subsistema de Configuración:** Este subsistema es importante pues en el se encuentran las clases que gestionan todo lo concerniente a la configuración del sistema, ya sea de usuario y contraseña, de Proxy, de puerto etc.

**Subsistema de Autenticación:** Este subsistema contiene las clases que intervienen en el proceso de autenticación del sistema, así como la validación de los mismos.

**Subsistema de Procesamiento Html:** Este es uno de los principales subsistemas del proyecto, pues este subsistema es el encargado de la búsqueda y análisis de las patentes descargadas, en la búsqueda realizada por el usuario, es también el encargado de crear dos ficheros, uno denominado Procite y otro Excel con las especificaciones establecidas por el cliente.

En este epígrafe se hará un análisis exhaustivo del diseño de clases de este subsistema, explicando detalladamente cada clase que resulte implicada en el mismo.

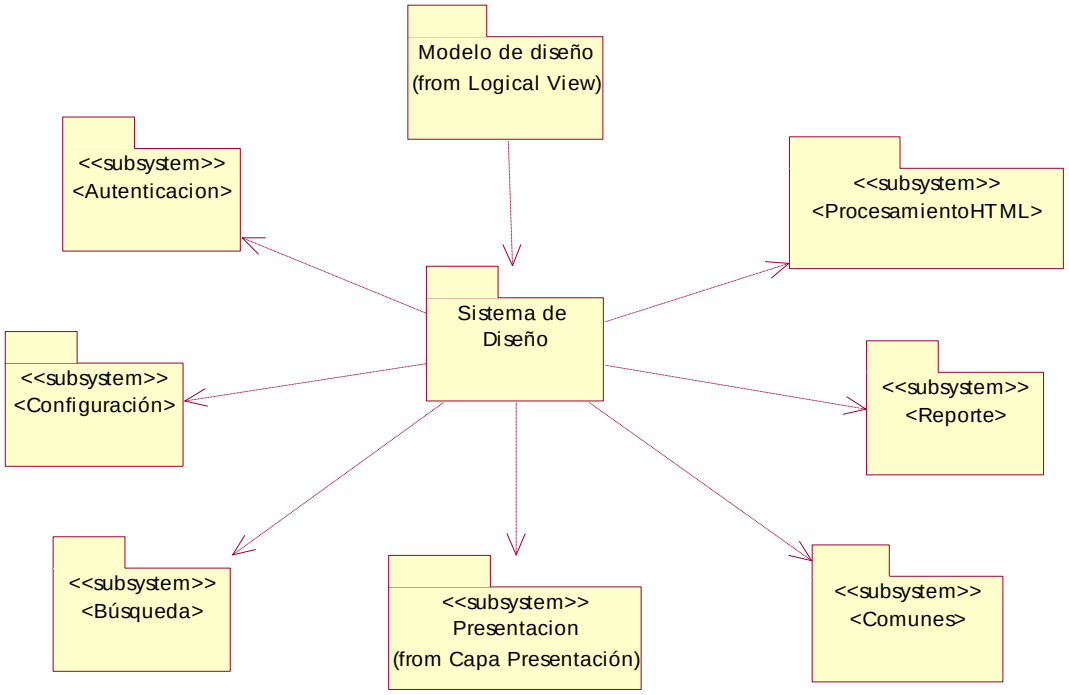


**Figura 2-1 Diseño Clases del Sistema**

### 2.1.1 Subsistema de Procesamiento de Html

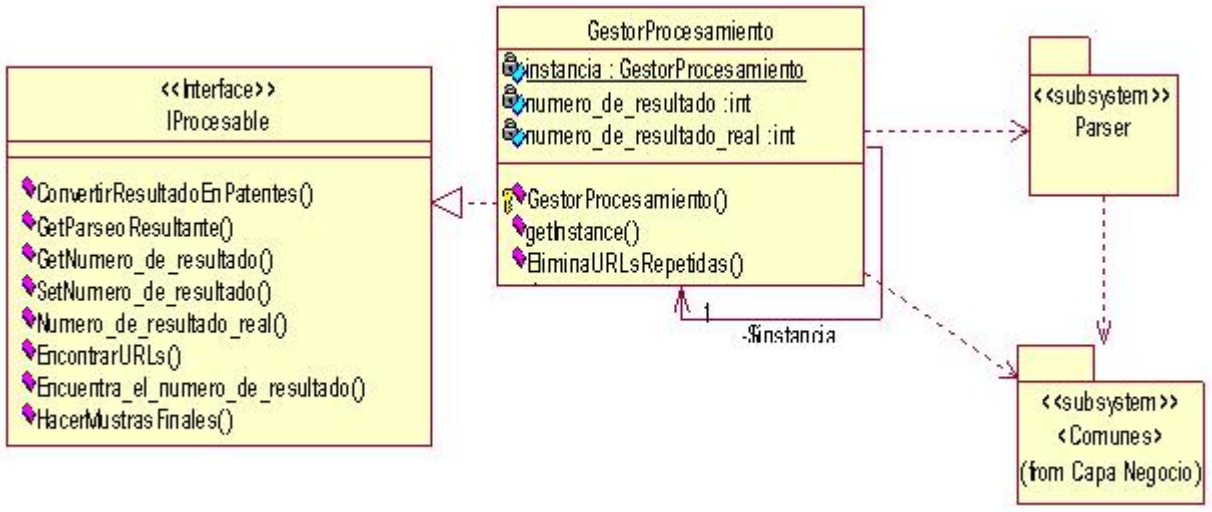


**Figura 2-2 Subsistema de Procesamiento HTML**

El subsistema de Procesamiento Html, esta compuesto por varias clases y varios subsistemas, como se muestra en la figura 2-2.

**Interface IProcesable:** En esta interface se encuentran todos los métodos que son comunes al procesamiento de html de una descarga, como lo son numero total de patentes encontradas en la búsqueda, manejar y controlar las direcciones urls de cada patente que se va a descargar, así como obtener el numero real de patentes.

**GestorProcesamiento**: Esta clase implementa la interface IProcesable, y su función principal es obtener una lista con las urls de las patentes que serán descargadas por el sistema, así como obtener el número real de patentes encontradas, el total de las patentes encontradas en la búsqueda realizada por el cliente, eliminar las urls que se repiten, pues así de esta manera se evita duplicados, tanto en la base de datos como en el resultado final de la búsqueda, también es la responsable de obtener o de crear los ficheros Procite y Excel, en conjunto son los objetivos principales de esta clase.

**Subsistema Parser:** Este subsistema es realmente importante pues las clases contenidas en el, son las que realizan la búsqueda y selección de los campos de las patentes validos para el cliente, el mismo posee varias clases (ver figura 2-3).

**Interface IParser:** Esta interface es la que posee los métodos que serán heredados por las clases que implementen esta interface.

**ParserSpain**: Esta clase es una de la más importante en el desarrollo de este subsistema y hereda de la interface IParser, esta clase es la encargada de realizar la búsqueda y selección de las patentes, pero mas específicamente en la base de datos de España, y es la encargada de devolver la lista de patentes de la descarga y también tiene la misión de crear dos ficheros, que son en conjunto el resultado final del subsistema y el del modulo de procesamiento.

**ParserEEUU:** Esta clase es una de la más importante en el desarrollo de este subsistema y hereda de la interface IParser, esta clase es la encargada de realizar la búsqueda y selección de las patentes, pero mas específicamente en la base de datos de Estados Unidos, y es la encargada de devolver la lista de patentes de la descarga y también tiene la misión de crear dos ficheros, que son en conjunto el resultado final del subsistema y el del modulo de procesamiento.

**FachadaParser**: Esta clase es la clase controladora global, ella contiene los métodos de creación de los ficheros, en cada una de las bases de datos, ya sea de Estados Unidos o de España.

**TagsUtiles:** Esta clase es una clase que en ella se encuentran agrupados todos los métodos que son comunes al proceso de búsqueda y selección de los campos validos de la patentes, así como poner los campos seleccionados con el formato especificado por el cliente.

**URLEEUU:** En esta clase se encuentran agrupadas funcionalidades tales como devolver la cantidad de patentes total encontradas en el proceso de descarga, así como devolver una lista de urls de la base de datos de Estados Unidos tanto en la búsqueda simple como en la avanzada, también es la encargada de eliminar las urls de las patentes repetidas.

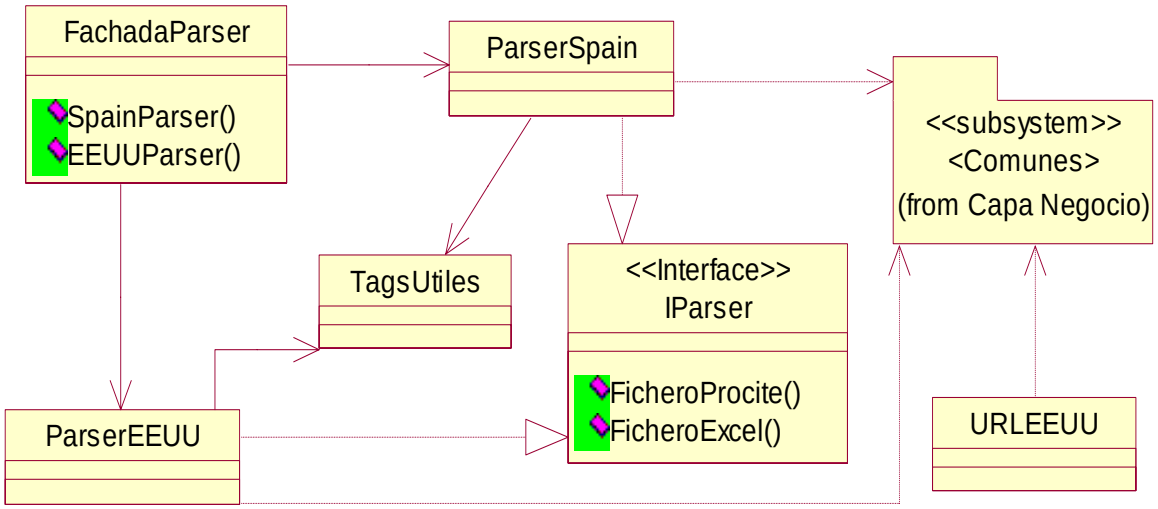


**Figura 2-3 Subsistema Parser**

**Subsistema Comunes:** Este subsistema es importante pues posee clases que serán utilizadas en otros subsistemas (ver figura 2-4) tal es el caso de la clase entidad Patente que ella es usada en el proceso de búsqueda y selección de los campos validos de la patentes así como en el proceso de los reportes.

**Patente:** Esta clase es la encargada de poseer los datos o campos de las patentes que van a perdurar en la base de datos.

**TFichero:** Esta clase es la encargada de gestionar las operaciones con los ficheros, ya sean de escribir, de leer, obtener un arreglo con las líneas del fichero leído.

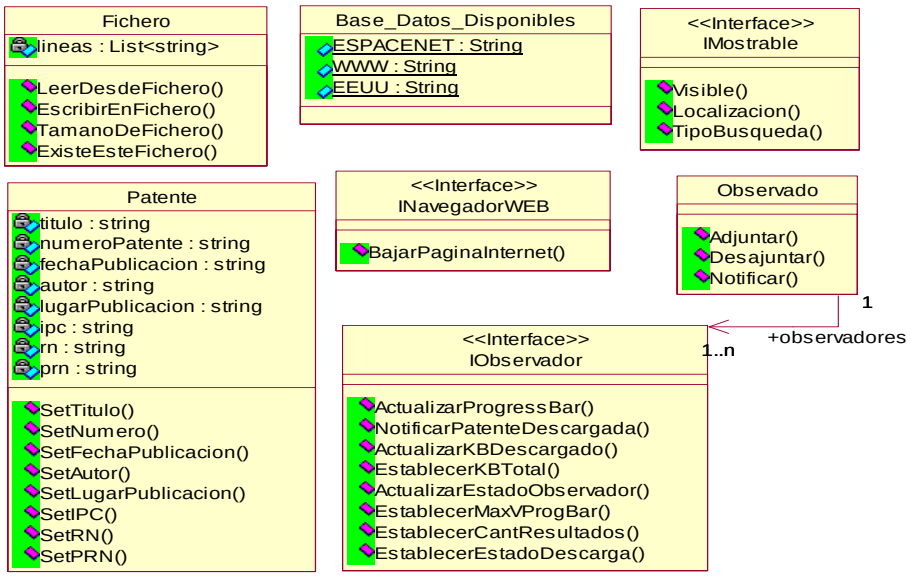


**Figura 2-4 Subsistema Comunes**

El diseño de clases propuesto por el analista para el modulo de Procesamiento Html, es un diseño bueno, pues en primer lugar permite la implementación de todos los requisitos contenidos en el modelo de análisis, cumple con todos los requisitos especificados por el cliente, da soporte al sistema, permite la reutilización de código, brinda la posibilidad de mantener el sistema, sin necesidad de cambiar la arquitectura del mismo, este fue creado bajo una estructura arquitectónica reconocible como es la Arquitectura 3 Capas, la cual define la estructura general del software , las relaciones entre los subsistemas y componentes del software y las reglas que especifican las relaciones entre los elementos que conforman la arquitectura, además posee componentes que representan características funcionales independientes, tal es el caso de la librería de clase Parser, y el componente Outlook, pero como todo diseño nunca es perfecto siempre se puede mejorar, de tal forma que agrupe mas funcionalidades del sistema y que el mismo sea mas robusto, mas fácil de mantener y adaptar a los cambios pedidos por el cliente.

.

## 2.2 Análisis de posibles implementaciones, componentes o módulos ya existentes.

El análisis de posibles implementaciones de componentes que se utiliza para el desarrollo de este sistema es de gran importancia ya que ahorra tiempo de trabajo, se puede obtener uno o varios componentes totalmente independientes que se utilizan en el sistema, lo que significaría que se tendrían algunas piezas claves del mismo sin pasar por su proceso de construcción y solo quedaría integrarlas con el resto del sistema lo cual se va a tratar en el epígrafe siguiente, entre ellas están(SABALLO 2007)

**Componente Outlook:** es el nombre que se le da al ensamblado que se utiliza como menú izquierdo en la interfaz gráfica principal para mostrar las distintas opciones del sistema, el nombre viene dado por su parecido al original creado por Microsoft en su producto Outlook que es un cliente de correo incluido en su paquete office. Este componente proporciona al sistema una gran flexibilidad y provee una buena apariencia grafica la cual se puede personalizar fácilmente, esta forma de menú es muy usada en estos días, con implementaciones diferentes, por los desarrolladores de aplicaciones Windows. Ver figura 2.5

**Componente TreeListView:** es el nombre que se le da al componente que se utiliza para mostrar reportes, el nombre viene dado por la mezcla de los controles básicos TreeView y ListView y con esta unión podemos lograr potencialidades graficas más personales y especializadas para llevar al usuario final la mayor satisfacción y comodidad posible, pues en cada ítem del Listview se encuentra un TreeView y en eso consiste la potencialidad de esta unidad compuesta. Estos dos componentes forman parte de la Tesis (Saballo 2007) basada en el rol de programador. Ver figura 2.5

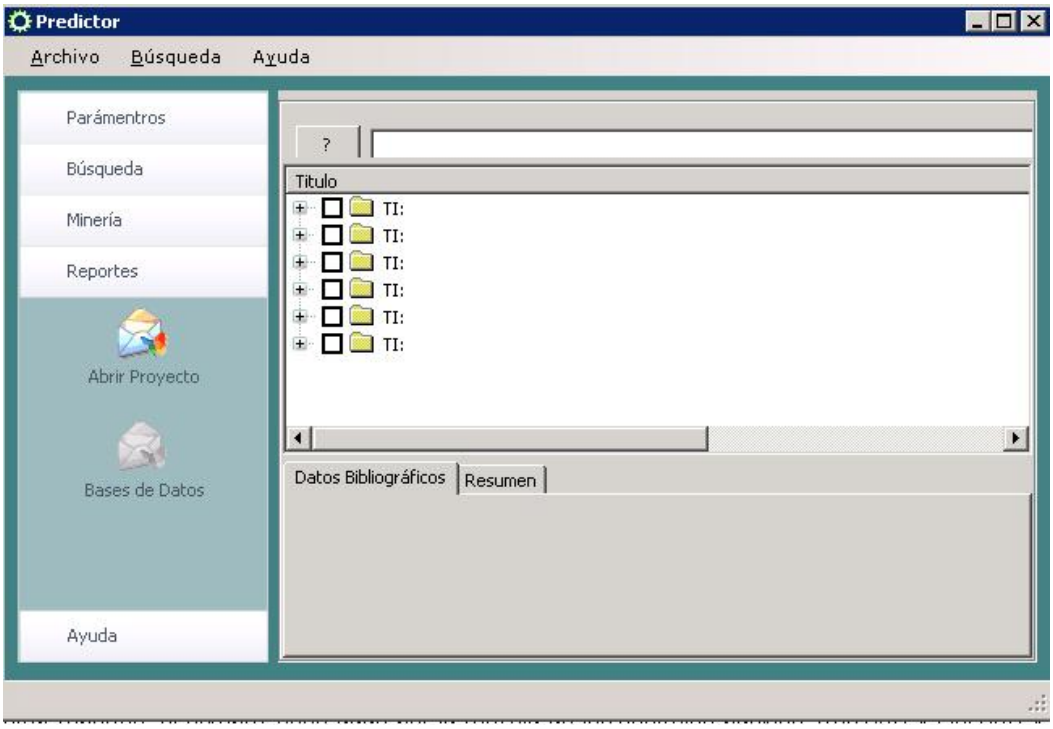


**Figura 2-5 A la derecha en la imagen se encuentra el componente Outlook y a la izquierda el Treelistview**

**Componente Parser:** es el nombre que se le da al ensamblado que se encarga del procesamiento del HTML de las páginas Web ya descargadas de Internet. Es donde están los algoritmos para la búsqueda, limpieza y extracción de los datos bibliográficos de las patentes.

### 2.3 Estrategias de integración de los componentes ya existentes.

Para llevar a cabo la integración de los distintos componentes ya existentes que se usan en este sistema primeramente se probaron de forma independiente, lo cual se conoce como pruebas unitarias o pruebas individuales, verificando el comportamiento del componente de forma externa, y analizando la implementación interna de la unidad, ya que se dispone del código fuente de los mismos los cuales son implementados por otras personas u organizaciones.

Para llevar a cabo la tarea de integración, se realizó un análisis de los casos de uso, que aportan más valor a la arquitectura del sistema y se dejó definido, el orden de integración de estas unidades de manera que se le diera cumplimiento a los requisitos funcionales correspondientes a estos casos de uso. Quedando de la siguiente forma, una vez creado el esqueleto de la aplicación, en el formulario principal se cargaran todos los controles, componentes existentes y los creados por el equipo de trabajo. Se establecieron las relaciones e implementaciones necesarias para integrar:

- Componente Outlook
- Componente Parser
- Componente TreeListView

El componente Outlook brinda la posibilidad de visualizar las distintas opciones de las que dispondrá el sistema, las cuales permitirán al usuario comprobar cada uno de los requisitos funcionales, en este caso el componente Outlook es una unidad de interfaz grafica, lo cual tiene un gran impacto en la visualización del avance del proyecto, lo cual también contribuyó a que se escogiera de primero a la hora de integración con el resto del sistema, en el caso del componente Parser, que es un ensamblado utilizado en la capa del negocio[4], el mismo es fundamental para ver los resultados del principal objetivo de este sistema que es la descarga automática de patentes, ya que este Parser es el encargado de procesar el html de las páginas Web. Luego se pasa a la integración del componente TreeListView para mostrar los reportes al usuario, y con ello tener una primera versión del producto, en este caso un sistema que pueda realizar las funcionalidades más básicas que dieron origen a su creación.

## 2.4 Descripción de los algoritmos no triviales a implementar. Análisis de complejidad de los mismos

Para que este modulo tuviera la funcionalidad requerida por los clientes se hizo necesario hacer un estudio exhaustivo de los algoritmos de búsquedas existentes, ya que la búsqueda de elementos dentro de un array es una de las operaciones mas importantes en el procesamiento de la información, y permite la recuperación de datos previamente almacenados. El tipo de búsqueda se puede clasificar como interna o externa, según el lugar donde pueda estar almacenada la información (en memoria o en dispositivos externos). Todos los algoritmos de búsquedas tienen dos finalidades:

- Determinar si el elemento buscado se encuentra en el conjunto en el que se busca.
- Si el elemento está en el conjunto, hallar la posición en la que se encuentra.

En este epígrafe nos centramos en los algoritmos de búsqueda interna, entre los algoritmos estudiados y analizados de búsqueda se encuentran:

- Algoritmo de Búsqueda Binaria o Dicotómica.
- Algoritmo de Búsqueda por Interpolación.
- Algoritmo de Búsqueda Secuencial.
- Algoritmo de Búsqueda Secuencial con Centinela.

**Algoritmo de Búsqueda Binaria o Dicotómica:**

Este algoritmo divide en cada iteración el vector en dos realizando la búsqueda en una sola de las mitades, disminuyendo de esta forma el espacio de búsqueda, para que este algoritmo pueda ser utilizado el vector o el array debe de estar ordenado. Su complejidad es O (log n).

**Algoritmo de Búsqueda por interpolación:**

Este algoritmo es una modificación del algoritmo anterior, en el algoritmo de búsqueda binaria, siempre se selecciona el elemento central del vector o el array, para compararlo con el elemento a buscar y dividir el vector, es posible realizar una modificación a este algoritmo de tal forma que el elemento no sea el central, sino aquel, que se corresponda con el elemento buscado, si la distribución de valores de este es uniforme. Al igual que el algoritmo anterior para poder usar o implementar este algoritmo el vector o el array debe de estar ordenado. La complejidad de este algoritmo es también de O (log n)

**Algoritmo de Búsqueda secuencial y algoritmo de Búsqueda secuencial con centinela.**

El algoritmo de búsqueda más sencillo recorre el vector desde el primer elemento hasta el último y si encuentra el valor buscado retorna su posición. Obviamente, se trata del único algoritmo posible si el vector no está ordenado. Sin embargo, si el vector está ordenado resulta muy ineficiente. Este algoritmo

realiza siempre el mismo número de iteraciones independientemente de la posición en que se encuentre el valor buscado. El algoritmo de búsqueda secuencial resulta muy ineficiente puesto que no se aprovecha del hecho de que el vector esté ordenado. Es posible modificarlo para que, teniendo en cuenta esa circunstancia, finalice en cuanto localice el elemento o se determine que éste no existe. Esta modificación se conoce como búsqueda secuencial con centinela.

El número de iteraciones siempre es igual al tamaño del vector y, puesto que todas las operaciones del interior del bucle tienen coste unitario, la complejidad del algoritmo de búsqueda secuencial es O (n).

Para la implementación de este modulo seleccionamos el algoritmo de búsqueda secuencial con centinela, independientemente de que este algoritmo no es el mejor de todos los algoritmos existentes para realizar la búsqueda en cuanto a eficiencia, pero si es el mas indicado para la implementación, pues para poder usar los demás algoritmos, la lista contenedora de elementos debe de estar ordenada , y en este caso no conviene que este ordenada, pues de esta forma se perderían datos de las patentes descargadas previamente por el sistema.

**2.5 Estructuras de datos apropiadas para la implementación de estos algoritmos.**

En la implementación de este sistema se utilizan estructuras de datos como listas genéricas que provee el lenguaje C# en su versión 2.0, para llevar a cabo el almacenamiento de colecciones de objetos en tiempo de ejecución, estas listas están presentes en muchas partes de la implementación del sistema ya que constituye la estructura de dato medular del mismo, ejemplo de uso de estas tenemos las representaciones que se hacen en el modelo diseño de las relaciones de agregación o composición entre clases u otras como asociaciones, estas últimas usadas con menos intensidad, o simplemente encontramos estas listas controlando una colección de datos simples como cadenas de textos, o simples números, etc. Esta estructura de dato, ya implementada por el framework de .NET es muy parecida a otras listas que puedan tener otros IDEs en su biblioteca de componentes, pues es una estructura de datos que los desarrolladores crean siguiendo determinados estándares y principios, esto facilita su uso, y provee las funcionalidades que se puedan necesitar en un momento determinado, como son adicionar, eliminar, insertar elementos de las mismas, por mencionar algunas entre las muchas que tienen. Otra estructura de dato utilizada en el sistema de forma implícita son los árboles, a través del componente

TreeListView en el subsistema de reporte, para poder mostrar a través de los nodos[5] del árbol de reporte los distintos datos bibliográficos de las patentes, facilitando la visualización de estos por parte de los usuarios del sistema. También se tiene como estructura de dato, el uso de ficheros textos, para almacenar las secuencias de caracteres, que son extraídas de las páginas Web descargadas, así como los datos bibliográficos de las patentes listos para analizar, los datos de configuración de conexión a Internet entre otros.

### 2.6 Descripción de las nuevas clases u operaciones necesarias.

| **TagsUtiles** | |
|---|---|
| **Controladora** | |
| **Atributo** | **Tipo** |
| No tiene | No tiene |
| **Responsabilidades** | |
| Nombre: | Descripción: |
| `ContieneTagTDAbierto(string linea)` | Este método te devuelve si una línea contiene el tag TD Abierto |
| `ContieneTagTDCerrado(string linea)` | Este método devuelve si una linea contiene el Tag TD Cerrado |
| `ContieneTagNoBRAbierto(string linea )` | Este método devuelve si una linea contiene el tag NOBR abierto |
| `ContieneTagNoBRCerrado(string linea)` | Este método devuelve si una linea contiene el tag Nobr Cerrado |
| `ContieneTagTRAbierto(string linea)` | Esta método devuelve si una linea contiene el tag TR abierto |
| `ContieneTagTRCerrado(string linea)` | Este método linea devuelve si una linea contiene el tag TR cerrado |
| `ContieneTagH2Abierto(string linea)` | Este método devuelve si una linea contiene el tag H2 abierto |
| `ContieneTagH2Cerrado(string linea)` | Este método devuelve si una linea contiene el tag H2 cerrado |
| `ContienetagStrongAbierto(string linea)` | Este método devuelve si una linea contiene el tag Strong abierto |
| `ContienetagStrongCerrado(string linea)` | Este método devuelve si una linea contiene el tag Strong cerrado |
| `ContieneTagScriptAbierto(string linea)` | Este método devuelve si una linea contiene el tag Script abierto |
| `ContieneTagScriptCerrado(string linea)` | Este método devuelve si una linea contiene el tag Script cerrado |
| `ContieneClasificacion(string linea)` | Este método devuelve si una linea contiene la palabra clasificación |

[5] Un nodo es la unidad sobre la que se construye el árbol y puede tener cero o más nodos hijos conectados a él.

| | |
|---|---|
| LimpiarTextos(string linea) | Este método devuelve un string sin tag html, es decir le pasas una línea con tag y te la devueve sin ellos |
| ContieneTagBAbierto(string linea) | Este método devuelve si una linea contiene el tag B abierto |
| ContieneTagBCerrado(string linea) | Este método devueve si una línea contiene el tag B cerrado |
| LLevarAMayuscula(string linea) | Este método te convierte una cadena a Mayuscula |
| USPatent(linea) | Este método te devuelve si una cadena contiene el campo numero de patente |
| ADDUS(string linea) | Este método te devuelve un campo de la patente con un formato especifico |
| ADDAU(string linea) | Este método te devuelve un campo de la patente con un formato especifico |
| ADDPD(string linea) | Este método te devuelve un campo de la patente con un formato especifico |
| ADDTI(string linea) | Este método te devuelve un campo de la patente con un formato especifico |
| TagsFontAbierto(string linea) | Este método devuelve si una linea contiene el tag Font abierto |
| TagsFontCerrado(string linea) | Este método devuelve si una linea contiene el tag Font cerrado |
| TagsCenterCerrado(string linea)Este | Metodo devuelve si una cadena contiene el tag center cerrado |
| TagsCenterAbierto(string linea) | Metodo devuelve si una cadena contiene el tag center abierto |
| TagsPAbierto(string linea) | Este metodo devuelve si una cadena contiene el tag P abierto |
| TagsPCerrado(string linea) | Este metodo devuelve si una cadena contiene el tag P abierto |
| EstaResumen(string linea) | Este metodo devuelve si una linea contiene el campo Resumen |
| ADDAB(string linea) | Este metodo devuelve un campo de la patente con un formato especifico |
| Assignee(string linea) | Este metodo devuelve si en una linea esta en campo de la patente assignee |
| ADDAssignee(string linea) | Este metodo devuelve un campo de la patente con un formato especifico |
| EstaAppL(string linea) | Este metodo devuelve si una cadena contiene el campo de la patente especificado |
| ADDAppl(string linea) | Este metodo devuelve un campo de la patente con un formato especifico |
| EstaFiled(string linea) | Este metodo devuelve si una linea contiene el campo de la patente especificado |
| ADDFiled(string linea) | Este metodo devuelve un campo de la patente con un formato especificado |
| CurrentCSClass(string linea) | Este metodo devuelve si una linea contiene el campo de la patente especificado |

| | |
|---|---|
| ADDCurrentCSClass(string linea) | Este metodo devuelve un campo de la patente con un formato especificado |
| EstaInventors(string linea) | Este metodo devuelve si una linea contiene el campo de la patente especificado |
| AddAF(string linea) | Este metodo devuelve un campo de la patente con un formato especificado |
| EstaBR(string linea) | Este metodo devuelve si una linea contiene el campo de la patente especificado |
| EstaCurrentInternational(string linea) | Este metodo devuelve si una linea contiene el campo de la patente especificado |
| ADDCurrentCSClassInternational(string linea) | Este metodo devuelve un campo de la patente con un formato especificado |
| LineaExcel(string linea) | Este metodo devuelve una cadena con un formato especificado por el cliente |

**Tabla 1 Descripción de la Clase TagsUtiles**

| **ParserEEUU** | |
|---|---|
| **Controladora** | |
| **Atributo** | **Tipo** |
| LineasFichero | List<string> |
| LineasSeleccionadas; | List<string> |
| ListadoPatentes; | List<string> |
| Patentes | List<CPatente> |
| **Responsabilidad** | |
| **Nombre** | **Descripcion** |
| CargarLineasEnArreglo(string files) | Este método se usa para cargar en una lista las líneas html almacenada en el fichero que le pasamos por parámetro |
| LLenarLineasSeleccionadas(string path,string advanced) | Este método se usa para hacer una búsqueda secuencial, con el objetivo de obtener los campos de la patente, path: la dirección del fichero, advanced: te indica si el fichero proviene de una base de avanzada, o simple |
| MostrarListadoPatentes(string path,string advanced) | Este método se usa para obtener los campos de la patente ya sin los tag html, path: la dirección del fichero, advanced: te indica si el fichero proviene de una base de avanzada, o simple |
| FicheroProcite(string files,string advanced) | Este método se usa para crear un fichero con el nombre de Procite, donde se guardan todas las patentes descargadas con un formato especificado por el cliente<br>advanced: te indica si el fichero proviene de una base de avanzada, o simple |
| FicheroExcel(string files,string advanced) | Este método se usa para crear un fichero con el nombre de Excel, donde se guardan todas las patentes descargadas con otro formato especificado por el cliente<br>advanced: te indica si el fichero proviene de una base de avanzada, o simple |
| **CParserEEUU()** | Constructor de la clase |

**Tabla 2 Descripcion de la Clase ParserEEUU**

| **ParserSpain** | |
|---|---|
| **Controladora** | |
| **Atributo** | **Tipo** |
| `Lineas;` | List<string> |
| `Construidas;` | List<string> |
| `FicheroFinal` | List<string> |
| `ListPantent` | List<Patente> |
| **Responsabilidad**: | |
| Nombre: | Descripción: |
| `LineasConstruidas(string files)` | Este método se hace para cargar en una lista los elementos contenidos en el fichero, para ello hace una búsqueda secuencial en el fichero files: la dirección del fichero a trabajar |
| `FormatoDeseado(string linea)` | Este método devuelve una cadena con un formato especificado por el cliente. |
| `NuevoFormato(string files)` | Este método devuelve una cadena eliminando los caracteres encontrados en el html: files :dirección del fichero |
| `FicheroExcel(string files)` | Este método crea un fichero en la dirección especificada, guardando en el una lista de patentes descargadas por el sistema, con un formato especifico. |
| `FicheroProcite(string files)` | Este método crea un fichero en la dirección especificada, guardando en el una lista de patentes descargadas por el sistema, con otro formato especifico. |
| **ParserSpain()** | Constructor de la clase |

**Tabla 3 Descripción de la Clase ParserSpain.**

| **FachadaParser()** | |
|---|---|
| **Controladora** | |
| **Atributo** | **Tipo** |
| | |
| | |
| Responsabilidades | |
| Nombre: | Descripción: |
| `SpainParser(string files, string advanced)` | Este método permite la creación de un fichero Procite y otro Excel, en el cual se van a encontrar las patentes descargadas desde Internet, en la base de datos de España, con un formato especifico en cada uno de los ficheros, dicho formato es especificado por el cliente<br><br>Files: directorio donde se encuentran los ficheros descargados.<br>Advanced: indica el tipo de búsqueda |
| `void EEUUParcer(string files, string advanced)` | Este método permite la creación de un fichero Procite y otro Excel, en el cual se van a encontrar las patentes descargadas desde Internet, en la base de datos de Estados Unidos, con un formato especifico en cada uno de los ficheros, dicho formato es especificado por el cliente<br><br>Files: directorio donde se encuentran los ficheros descargados.<br>Advanced: indica el tipo de búsqueda |

**Tabla 4 Descripción de la Clase FachadaParser**

| **IParser** | |
|---|---|
| **interface** | |
| **Atributo** | **Tipo** |
| | |
| **Responsabilidad**: | |
| Nombre: | Descripción |
| `FicheroProcite(string files,string advanced)` | Este método permite la creación de ficheros donde se guardaran las patentes descargadas de Internet |

| | |
|---|---|
| `FicheroExcel(string files, string advanced)` | Este método permite la creación de ficheros donde se guardaran las patentes descargadas de Internet |

**Tabla 5 Descripción de la Interfaz**

| **Patent** | |
|---|---|
| **Entidad** | |
| **Atributo** | **Tipo** |
| TI | `string` |
| PN | `string` |
| PD | `string` |
| AU | `string` |
| AF | `string` |
| IPC | `string` |
| IPCE | `string` |
| RN | `string` |
| PRN | `string` |
| AB | `string` |
| Responsabilidad | |
| Nombre: | Descripción: |
| TI | Este método devuelve el titulo de la patente |
| PN | Este método devuelve el numero de la patente |
| PD | Este método devuelve la fecha de la patente |
| AU | Este método devuelve el inventor de la patente |
| AF | Este método devuelve el solicitante de la patente |
| IPC | Este método devuelve la clasificación internacional |
| IPCE | Este método devuelve la clasificación nacional |
| RN | Este método devuelve el numero de registro de la patente |
| PRN | Este método devuelve la Fecha de solicitud la patente |
| AB | Este método devuelve el resumen de la patente |
| `Patent()` | El constructor de la clase |

**Tabla 6 Descripción de la Clase Patente.**

| **TFichero** |
|---|
| **Controladora** |

| **Atributo** | **Tipo** |
|---|---|
| Lines | List<String> |
| **Responsabilidad**: | |
| Nombre: | Descripción |
| `LeerDesdeFichero(string camino)` | Este método permite leer desde un fichero.<br>Camino: dirección donde se encuentra en fichero a leer |
| `EscribirEnFichero(string camino)` | Este método permite escribir<br>En un fichero.<br>Camino: dirección donde se encuentra en fichero a leer. |
| `EscribirEnFichero(string texto,string camino)` | Este método permite escribir<br>En un fichero.<br>Camino: dirección donde se encuentra en fichero a leer<br>Texto: Texto a escribir en el fichero |
| `TamañoDeFichero(string camino)` | Este método permite determinar el tamaño de un fichero.<br>Camino: Camino: dirección donde se encuentra en fichero a leer |

**Tabla 7 Descripción de la Clase TFichero**

| **UrlEEUU** | |
|---|---|
| **Controladora** | |
| **Atributo** | **Tipo** |
| | |
| **Responsabilidad**: | |
| Nombre: | Descripción: |
| `CargarURLEUUU(string file)` | Este método se usa principalmente para guardar en una lista las urls, de las patentes, que van a hacer descargadas desde Internet, provenientes de la Base de Datos de EEUU. |
| `HrefNETACGI(string linea)` | Este método devuelve si una cadena determinada contiene el principio de la urls.<br>Línea: La cadena a examinar |
| `A(string linea)` | Este método devuelve si una cadena determinada es realmente una urls.<br>Línea: Cadena a examinar |
| `QuitarHREF(string linea)` | Este método se usa para formar las urls de resultado.<br>Línea: Cadena a examinar |
| `QuitarSegundaParte(string linea)` | Este método se usa para formar las urls de resultado.<br>Línea: Cadena a examinar |

| | |
|---|---|
| `EliminarAmpersan(string linea)` | Este método se usa para eliminar de una cadena los caracteres especiales que esta posee.<br>Línea: Cadena a examinar |
| `CargarURLEUUUAvanzada(string file)` | Este método se usa principalmente para guardar en una lista las urls, de las patentes, que van a hacer descargadas desde Internet, provenientes de la Base de Datos de EEUU. |
| `EliminarUrls(List<string> p)` | Este método se usa para eliminar las urls repetidas.<br>P: Lista de urls |
| | |

**Tabla 8 Descripción de la Clase URLEEUU**

| **GestorProcesamiento** | |
|---|---|
| **controladora** | |
| **Atributo** | **Tipo** |
| instance | GestorProcesamiento() |
| numeroreal | int |
| numeroresultado | int |
| **Responsabilidad**: | |
| Nombre: | Descripción: |
| `GestorProcesamiento` | Constructor de la Clase |
| `Getinstance` | Este método devuelve una instancia de la clase Procesamiento, esto se hace con el objetivo de devolver una única instancia de la clase Procesamiento |
| `EliminarUrlsRepetidas(List<string> p)` | Este método se usa para eliminar las urls repetidas.<br>P: Lista de urls |
| `GetNumeroResultado()` | Este método se usa para obtener el numero de patentes encontradas en la búsqueda |
| `GetNumero_Real_Resultado()` | Este método se usa para obtener el numero real de resultados encontrados en la búsqueda |
| `Encontrar_el_Numero_Resultado(string camino)` | Este método se usa para encontrar el numero de resultado;<br>Camino: dirección donde se encuentra guardado el fichero. |
| `ConvertirResultadosenPatente(List<string> p)` | Este método se usa principalmente serial izar los campos de la patente. |

**Tabla 9 Descripción de la Clase GestorProcesamiento**

| **IProcesable** | |
|---|---|
| **interface** | |
| **Atributo** | **Tipo** |
| No Tiene | No Tiene |
| **Responsabilidad**: | |
| Nombre: | Descripción: |
| `Getinstance` | Este método en la interface no devuelve nada, pero en las clases que lo implementan devolvería una instancia de esa clase. |
| `EliminarUrlsRepetidas(List<string> p)` | Este método en la interface no devuelve nada, pero en las clases que lo implementan devolvería una lista de urls sin duplicados. |
| `GetNumeroResultado()` | Este método en la interface no devuelve nada, pero en las clases que lo implementan devolvería el número total de resultados. |
| `GetNumero_Real_Resultado()` | Este método en la interface no devuelve nada, pero en las clases que lo implementan devuelve la cantidad real de resultados encontrados. |
| `Encontrar_el_Numero_Resultado(string camino)` | Este método en la interface no devuelve nada, pero en las clases que lo implementan seria la encargada de buscar la cantidad total de patentes encontradas en la búsqueda. |
| `ConvertirResultadosenPatente(List<string> p)` | Este método en la interface no devuelve nada, pero en las clases que lo implementan, seria la encargada de devolver una lista de patentes. |

**Tabla 10 Descripción de la Interfaz IProcesable**

## 2.7 Estilos de Programación.

Un estilo de programación no es más que un conjunto de reglas o normas usadas para escribir código y que incluye una gran gama de aspectos dentro del proceso de codificación y es uno de los temas más discutidos entre los programadores. Mientras unos están convencidos de que es un simple problema de estética, otros compiten, como los productores modernos de compiladores, poniendo cada vez más detalles para facilitar la escritura y lectura del código e introducen ciertas normas de estilo.

Estos puntos son importantes pero hay una justificación con más relevancia: la forma de escribir los programas, debe ser una representación exacta de las estructuras del modelo que se define. Por supuesto, que el nivel que se logre al cumplir esta regla depende de las estructuras que defina o disponga

el lenguaje de programación. El mejor estilo es el que más aporte a la eficiencia, legibilidad y rapidez del proceso de desarrollo y el que potencie los programas más robustos y fáciles de usar (Rodríguez 2006).

Grady Booch, en su libro Object Oriented Design with Applications[6], enuncia como una de la causas de la complejidad del software *"… la dificultad de administrar el proceso de desarrollo, más miembros en un grupo de desarrollo significa mayor complejidad en las comunicaciones… cuando se trabaja en grupo, el reto mayor es mantener la unidad e integridad".* Se necesita una comunicación eficiente entre los miembros de un grupo de desarrollo, para esto es necesario que todos usen un código de comunicación común.

Un buen estilo asegura muchos detalles, algunas de las más relevantes son: los programas más comprensibles, más fácil extender o mantener los módulos de un sistema, las tecnologías desarrolladas son más fáciles de usar en varias plataformas de desarrollo. Las estructuras usadas en la codificación deben corresponderse exactamente con las estructuras obtenidas en el proceso de diseño del modelo. Esto no es posible exactamente por la mala calidad de los lenguajes de programación existentes con respecto a la implementación que hagan de las estructuras que definen las metodologías de diseño más elaboradas (Rodríguez 2006). Razones por lo que es tan dañino el hecho de que la mayoría de los programadores basen su aprendizaje en los lenguajes y no en métodos generales de diseño. Veamos algunos ejemplos. Si se le pregunta a una persona que no sea especialista en la ciencia de la computación que explique el procedimiento para encontrar un documento de patente en un listado de ellas, probablemente responda: "*reviso todas las patentes hasta que alguna se corresponda con la que ando buscando"*. Note que la frase "*hasta que"* define el elemento de control del algoritmo. Esta es una idea aceptable, probablemente un programador hubiera dicho: "*recorro las patentes mientras la actual sea distinta a la buscada o no se haya llegado al final de la lista (lista de patente), si la encuentro devuelvo el índice, y si no, menos uno".* Más elaborada para codificar. Sin embargo, si se le pide a un programador inexperto que codifique estas ideas probablemente el resultado seria:

```
public int Buscar(Patente elemento, List<Patente> lista_Elementos)
{
int resultado = -1;
for (int i = 0; i < lista_Elementos.Count ; i++)
if (lista_Elementos[i] == elemento)
```

[6] Redwood City, California. 1991, The Bejamin/Cummings Publishing Company, Inc. p.4

```
{
resultado = i;
break;
}

    return resultado;

    }
```

Las ideas originales no se parecen a este resultado. El **for,** en C# es mas flexible que el de otros lenguajes como Pascal pero la idea principal sigue siendo la misma un ciclo de variable de control y significa semánticamente que se va a repetir algo un número determinado de veces, y no *hasta que* ocurra algo. Aunque en el caso de C#, como en C++ o Java y otros pueda usarse como tal hasta que suceda algo, sin embargo en este caso esto no se cumple porque un **break,** rompe con la estructura definida anteriormente. Si se tratara de traducir exactamente estas instrucciones a lenguaje natural no se obtendrían resultados coherentes. Las instrucciones que rompen estructuras de programación son una mala práctica de codificación, deben usarse lo menos posible, porque introducen efectos no deseados en la comprensión directa de los algoritmos. Otras instrucciones que causan un efecto similar son: **goto**, **exit** y **continue**. No es un problema de eliminarlos por capricho, siempre es posible demostrar que el uso de estas estructuras es una mutilación de las ideas originales (Rodríguez 2006). Si se reescribe el algoritmo anterior:

```
    public int Buscar(Patente elemento, List<Patente> lista)

    {

    int resultado = -1;

    int i = 0;

    while (lista[i] != elemento && i < lista.Count)

    i++;

    if (i < lista.Count)

    resultado = i;
```

```
else
resultado = -1;
return resultado;
}
```

Esta versión es mucho más fácil de entender: nos paramos en la primera patente, mientras la actual no es la que se este buscando y queden patentes, se mueve para la siguiente; si se encontró entonces devuelvo su índice, de lo contrario se devuelve menos uno. Si se escribe poniéndole a cada frase su origen en el código: se detendría en la primera patente (i = 0), mientras (**while**) la actual no es la que se este buscando (list**[i] !=** elemento) y (**and**) queden patentes (i **<** lista**.**Count), se mueve para la siguiente (i++); si (**if**) se encontró (i **<** list**.**Count) entonces se establece su índice (resultado **=** i), de lo contrario (**else**) se establece menos uno (resultadp **=** -1), y se devuelve el resultado establecido. Sin comentarios.

### 2.8 Estándar de Codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código, debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código del sistema. Al comenzar un proyecto de software, es necesario establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. 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.

• Idioma: Se debe emplear un solo idioma y se propone el español, las palabras no se acentuarán.

• Palabras Reservadas: Las palabras reservadas van en minúsculas sin excepción alguna.

• Indentación: En el caso nuestro IDE (VS.NET 2005) se encarga de ajustar los espacios de forma automática. Los inicios “(“,” {“, “)” y cierre “(“, “}”, “)” de bloque deben estar alineados debajo de la declaración a la que pertenecen y es a gusto utilizarse o no en el caso de que exista una sola instrucción. Nunca colocar {en la línea de un código cualquiera, esto requiere una línea propia. El cuerpo de

declaraciones compuestas (declaraciones como “if”, “while”, “for”, etc.) debe anidarse a una profundidad de 4 espacios.

• Líneas en blanco: Coloque una línea en blanco antes y después de la declaración de una estructura o de una clase.

• Uso de espacios en blanco: Los signos lógicos y de operación deberán estar separados por un espacio antes y después del mismo, ejemplo: `int Total = 15 - 8 + 9;` no se debe utilizar entre el nombre de un método y el paréntesis abierto, `void HacerAlgo(int Posicion);` entre el cast y la expresión, ejemplo: `float Promedio = (float)(5 + 4) / 2`; después del corchete abierto y antes del cerrado de un arreglo, ejemplo `int Cantidad = arrElementos[i].`

• Comentarios de línea: Consiste en un “//” seguido de un texto. Debe incluirse un espacio simple entre el “//” y el texto. Se debe evitar comentar cada línea de código. Es mejor colocar el comentario precediendo al bloque de líneas de código. Si el comentario se aplica a un grupo de instrucciones debe estar seguido de una línea en blanco, si se refiere sólo a la siguiente instrucción, se suprime la línea. Este tipo de comentario también puede seguir al código que referencia, debiendo aparecer en la misma línea.

• Declaraciones de clases: Las directivas de visibilidad deben estar identadas con la definición de la clase .Estas directivas deben declararse en el orden siguiente: private, protected, public.

. • Variables locales: Las variables deben ser declaradas justo antes de ser utilizadas. Cuando sea posible, la inicialización de las variables locales ocurrirá en el momento de su declaración. Se debe evitar la declaración dentro del cuerpo de un ciclo.

• Declaraciones condicionales: Las declaraciones “if” – “else” no deben poseer más de 4 niveles de profundidad, de tenerlos no deben estar en una línea sino repartidos en igual número de líneas como niveles tenga, esto se explica porque es más fácil leer una declaración “if” – “else”de arriba para abajo que de izquierda a derecha. La cláusula else, siempre debe estar alineada con la cláusula “if”. ” //” Se sugiere utilizar cuando las condiciones sobrepasan los márgenes.

if (condition1 && condition2 && condition3)

{

…..

}

else

• Declaraciones switch: Las expresiones constantes dentro de las declaraciones “switch” deben estar ordenadas numérica o alfabéticamente dependiendo del tipo al que pertenecen. La cláusula default deberá alinearse con la cláusula “case “.

• Declaraciones “for”, “foreach”: El código de inicialización debe realizarse en la declaración del mismo a menos que sean necesario declararlo fuera para darle más tiempo de vida a la variable de control.

• Declaraciones “while”: Las declaraciones “while” deben usarse en lugar de “for” cuando el lenguaje natural a la hora de pensar en algo que se repita lo indique.

• Declaraciones “do”…”while”: El código de la declaración está acotado por las palabras reservadas: “do”... “while”. Se utilizara cuando se necesite explícitamente que se cumpla al menos una vez ciclo post condicional.

• Regla para identificadores: Los nombres deben formarse a partir de palabras en español únicamente. En el caso de variables y atributos en minúscula y en los nombres de métodos o propiedades la primera letra en Mayúscula en caso de ser una palabra compuesta la primera letra de la que empieza a continuación de la anterior, ejemplo: CambiarEstado (…)

# CAPÍTULO 3: Validación de la solución propuesta.

## Introducción

En este capitulo se abordará la validación de la solución propuesta por el programador, para que el sistema cumpla con los requerimientos especificados por el cliente. Para ello se van a desarrollar [7]Pruebas Unitarias con el objetivo de asegurar que el módulo funcione correctamente por separado, para luego con las Pruebas de Integración poder asegurar el correcto funcionamiento del sistema o subsistema analizado. La idea principal es escribir casos de prueba para cada función no trivial o método en el módulo de forma que cada caso sea independiente del resto.

El propósito principal de las Pruebas unitarias es aislar cada parte de programa, mostrar que las partes individuales son correctas, proporcionan un contrato escrito que el trozo de código debe satisfacer, estas pruebas aisladas proporcionan 5 ventajas básicas.

- **Fomentan el cambio:** Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura (lo que se ha dado en llamar refactorización), puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.

- **Simplifica la integración**: Puesto que permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente. De esta manera se facilitan las pruebas de integración.

- **Documenta el código**: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.

- **Separación de la interfaz y la implementación.**

[7] una **prueba unitaria** es una forma de probar el correcto funcionamiento de un módulo de código.

- **Los errores están más acotados y son más fáciles de localizar**: puesto que se tienen pruebas unitarias que pueden desenmascararlos.

Es cierto que estas pruebas tienen ciertas limitaciones, es importante darse cuenta de que las pruebas unitarias no descubrirán todos los errores del código. Por definición, sólo prueban las unidades por sí solas. Por lo tanto, no descubrirán errores de integración, problemas de rendimiento y otros problemas que afectan a todo el sistema en su conjunto. Además, puede no ser trivial anticipar todos los casos especiales de entradas que puede recibir en realidad la unidad de programa bajo estudio. Las pruebas unitarias sólo son efectivas si se usan en conjunto con otras pruebas de software. Estas pruebas unitarias se realizan con varios software en dependencia del lenguaje o de la plataforma en que se este desarrollando el sistema, por ejemplo JUnit se usa para realizar las pruebas en java, NUnit para realizar las pruebas en la plataforma.Net, en cualquiera de sus lenguajes, etc.

## 3.1 Búsqueda o diseño de las pruebas de unidades que permitan validar la solución propuesta.

En este epígrafe se mostrara los diseños para las pruebas de unidades que permitirán validar la solución propuesta, para la implementación del modulo de Parseo del Sistema Predictor. Para ello se elaboró una clase de prueba que se llama Class_Test_FosSystem, esta clase es la que va a contener todos los métodos de prueba para poder desarrollar las pruebas en la herramienta NNunit.

### Casos de Prueba #1

Este caso de prueba se hace con el objetivo de demostrar que el método que posee la responsabilidad de seleccionar los campos de la patentes, funcione correctamente.

| **Caso de Uso** | Procesar Patentes | | | |
|---|---|---|---|---|
| **Caso de prueba** | ShowPatentField() ver Figure22 Anexo | | | |
| **Condiciones** | Para que este caso de prueba se cumpla exitosamente, el sistema debe de haber descargado previamente las patentes desde la Base Datos de Estados Unidos. | | | |
| **Clases Válidas** | **Clases Inválidas** | **Resultado Esperado** | **Resultado de la Prueba** | **Observ.** |
| ParserEU | ParserSpain()<br>TFichero()<br>FachadaParser()<br>URLEEUU() | Con esta prueba se espera que el sistema o el método devuelva los campos almacenados en la patente descargada por el sistema, proveniente de la Base de Datos EEUU | El resultado de la prueba es el esperado, pues el sistema devuelve los 9 campos de la patente satisfactoriamente. | Cuando se realizo esta prueba con la herramienta NUnit se obtuvo el resultado correcto<br>(Ver figura 3-1). |
| **Descripción de los valores usados en la prueba** | Para la ejecución de esta prueba, se usaron valores validos, en este caso se uso un fichero descargado por el sistema perteneciente a una de las patentes bajadas desde Internet. | | | |

**Tabla 11 Caso de Prueba 1**

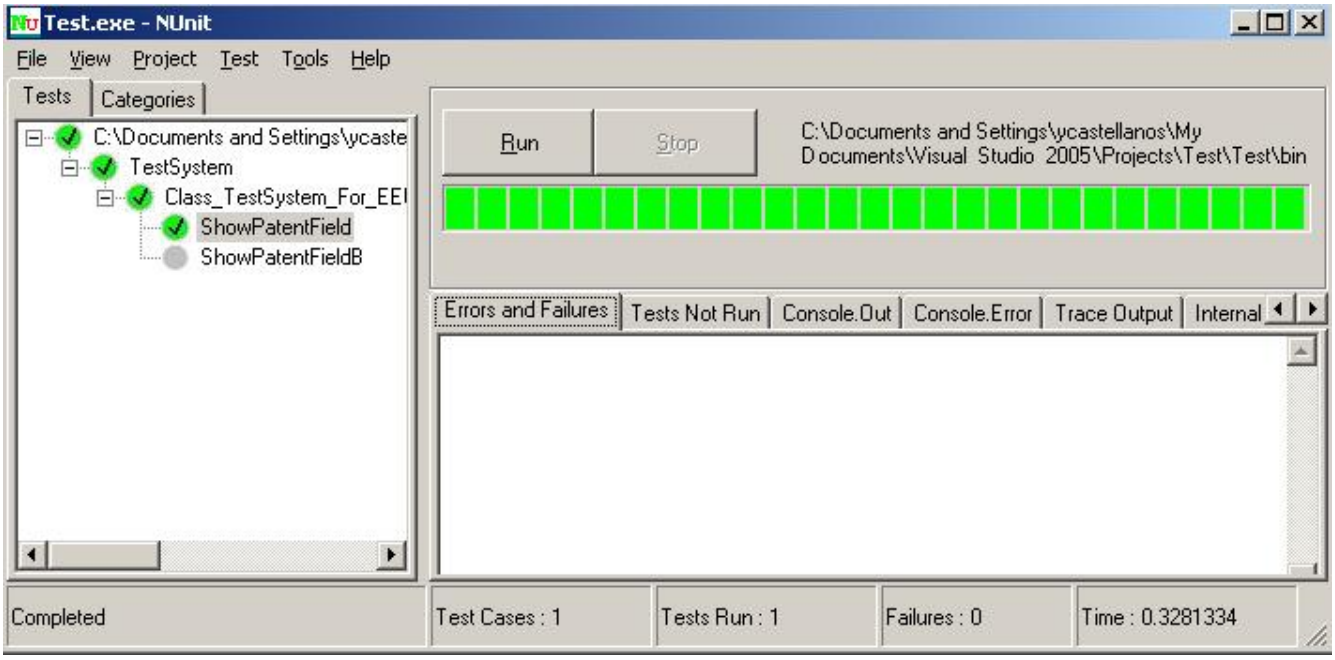


**Figura 3-1 Diseño del Caso de Prueba #1**

**Caso de Prueba #2**

Este caso de prueba se hace con el objetivo de demostrar que el método que posee la responsabilidad de seleccionar los campos de la patentes, funcione correctamente. En este caso específicamente este método debe retornar los campos de la una patente, perteneciente a la BDP de Estados Unidos.

.

| **Caso de Uso** | Procesar Patentes | | | |
|---|---|---|---|---|
| **Caso de prueba** | ShowPatentField() ver Figure22 Anexo | | | |
| **Condiciones** | Para que este caso de prueba se cumpla exitosamente, el sistema debe de haber descargado previamente las patentes desde la Base Datos de Estados Unidos. | | | |
| **Clases Válidas** | **Clases Inválidas** | **Resultado Esperado** | **Resultado de la Prueba** | **Observ.** |
| ParserEU | ParserSpain<br>TFichero<br>FachadaParser<br>URLEEUU | Con esta prueba se espera que el sistema o el método devuelva los campos almacenados en la patente descargada por el sistema, proveniente de la Base de Datos EEUU | El resultado de la prueba hecha al código fue el esperado, el caso de prueba funcionó correctamente. | Cuando se realizo esta prueba con la herramienta NUnit se obtuvo el resultado correcto<br>(Ver figura 3-2). |
| **Descripción de los valores usados en la prueba** | Para la ejecución de esta prueba, se usaron valores válidos, en este caso se uso un fichero descargado por el sistema correspondiente con la base de datos a la que pertenecía. | | | |

**Tabla 12 Caso de Prueba #2**

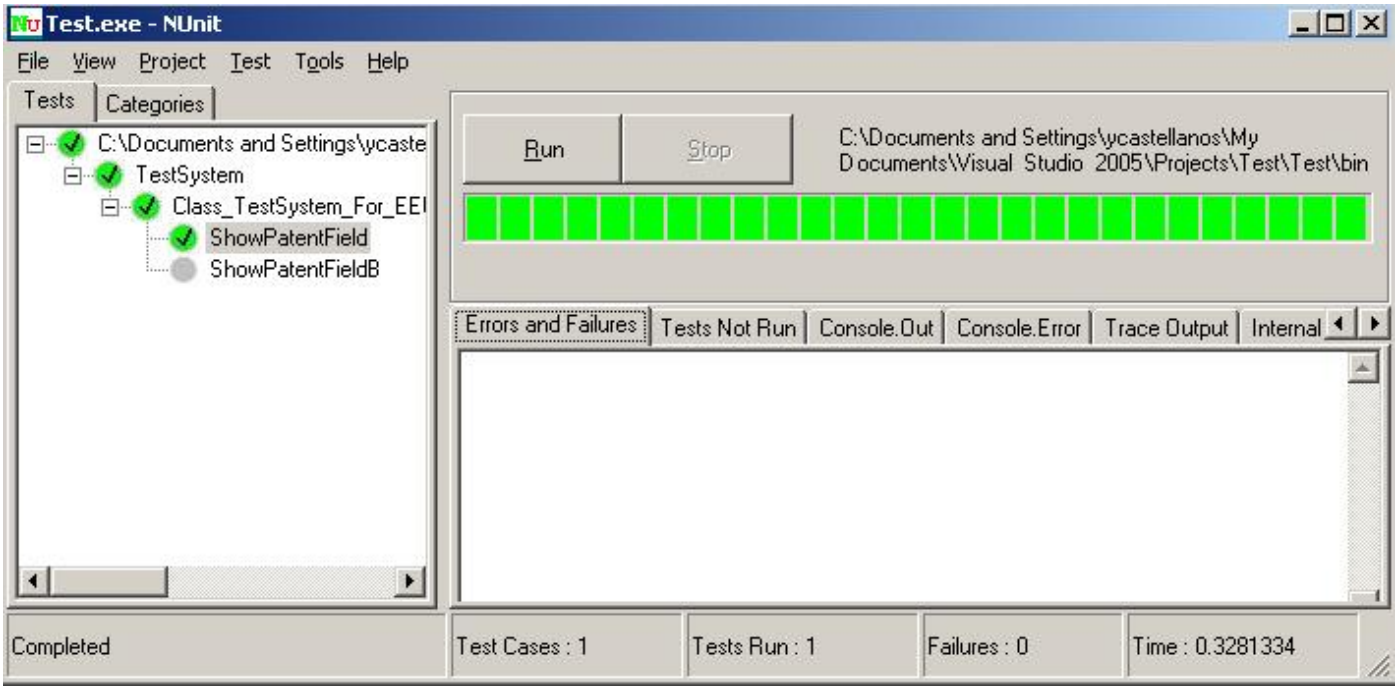


**Figura 3-2 Diseño del Caso de Prueba #2**

**Caso de Prueba #3**

Este caso de prueba se desarrolla con el objetivo de devolver la cantidad de patentes que descargo el sistema.

| **Caso de Uso** | Procesar Patentes | | | |
|---|---|---|---|---|
| **Caso de prueba** | CantidadPatentes() ver Figure23 Anexo | | | |
| **Condiciones** | Para que este caso de prueba se cumpla exitosamente, el sistema debe de haber descargado previamente las patentes desde cualquiera de las bases de datos, a las cuales se conecta el sistema. | | | |
| **Clases Válidas** | **Clases Inválidas** | **Resultado Esperado** | **Resultado de la Prueba** | **Observ.** |
| URLEEUU | ParserSpain<br>TFichero<br>FachadaParser<br>ParserEEUU | Con esta prueba se espera que el sistema o el método devuelvan la cantidad de patentes descargadas previamente por el sistema. | El resultado de la prueba hecha al código fue el esperado, el sistema devolvió la cantidad de patentes satisfactoriamente. | Cuando se realizo esta prueba con la herramienta NUnit se obtuvo el resultado correcto<br>(Ver figura 3-3). |
| **Descripción de los valores usados en la prueba** | Para la ejecución de esta prueba se usaron valores validos, pues se le paso la dirección Donde se encontraba el fichero descargado por el sistema. | | | |

**Tabla 13 Caso de Prueba #3**

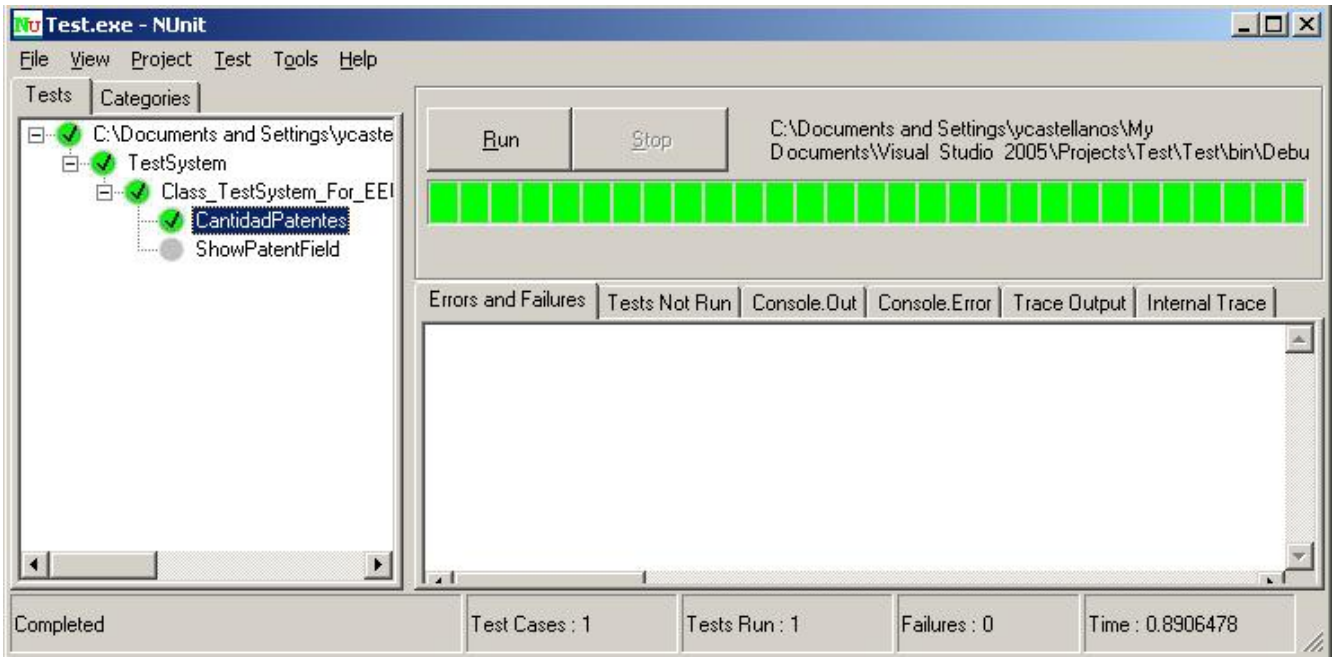


**Figura 3-3 Diseño del Caso de Prueba #3**

### Caso de Prueba #4

Este caso de prueba se realiza con el objetivo de demostrar que la limpieza del html se este efectuando correctamente.

| **Caso de Uso** | Procesar Patentes | | | |
|---|---|---|---|---|
| **Caso de prueba** | ParserText() Ver Figure24 Anexo | | | |
| **Condiciones** | Para que este caso de prueba se cumpla exitosamente, el sistema debe de haber descargado previamente las patentes desde cualquiera de las bases de datos, a las cuales se conecta el sistema. | | | |
| **Clases Válidas** | **Clases Inválidas** | **Resultado Esperado** | **Resultado de la Prueba** | **Observ.** |
| TagsUtiles | ParserSpain<br>TFichero<br>FachadaParser<br>ParserEEUU | Con esta prueba se espera que el sistema o el método de prueba devuelva un campo de la patente limpio, sin tags htmls | El resultado de la prueba hecha al código fue el esperado, el método devolvió el campo de la patente limpio, sin tags html. | Cuando se realizo esta prueba con la herramienta NUnit se obtuvo el resultado correcto<br>(Ver figura 3-4). |
| **Descripción de los valores usados en la prueba** | Para la ejecución de esta prueba se usó valores validos, en este caso se desarrollo la prueba con uno de los campos seleccionados por el sistema, perteneciente a una de las bases de datos a las cuales se conecta la aplicación. | | | |

**Tabla 14 Caso de Prueba #4**

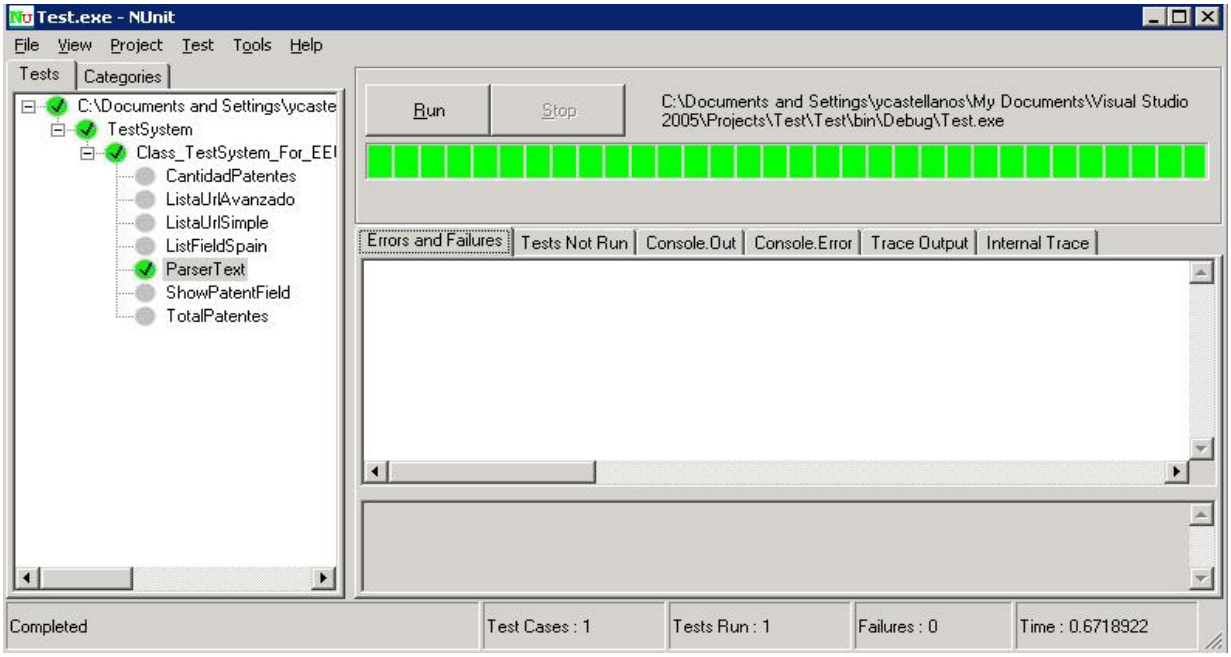


**Figura 3-4 Diseño del Caso de Prueba #4**

### Caso de Prueba #5

Este caso de prueba es importante, pues se realiza con el objetivo de obtener, la lista de las urls de la base de datos de EEUU avanzada, que serán descargadas por el sistema, y así de esta forma poder realizar todo el proceso de búsqueda y selección de los campos de la patentes.

| **Caso de Uso** | Procesar Patentes | | | |
|---|---|---|---|---|
| **Caso de prueba** | ListadoUrlAvanzado() ver Figure27 Anexo | | | |
| **Condiciones** | Para que este caso de prueba se cumpla exitosamente, el sistema debe de haber descargado previamente la página inicial de búsqueda, que es quien contiene la lista de las primeras 50 patentes. | | | |
| **Clases Válidas** | **Clases Inválidas** | **Resultado Esperado** | **Resultado de la Prueba** | **Observ.** |
| URLEEUU | ParserSpain<br>TFichero<br>FachadaParser<br>ParserEEUU | Con esta prueba se espera que el sistema o el método de prueba devuelvan una lista de urls de patentes, lista para ser descargadas por el sistema. | El resultado de la prueba hecha al código fue el esperado, el caso de prueba devolvió y comprobó el resultado mostrado por el método, analizado | Cuando se realizo esta prueba con la herramienta NUnit se obtuvo el resultado correcto<br>(Ver figura 3-5). |
| **Descripción de los valores usados en la prueba** | Para la ejecución de esta prueba se usaron valores que no presentaban problemas, en este caso se utilizo un fichero previamente descargado por el sistema, este fichero es el que contiene el listado de las urls. | | | |

**Tabla 15 Caso de Prueba #5**

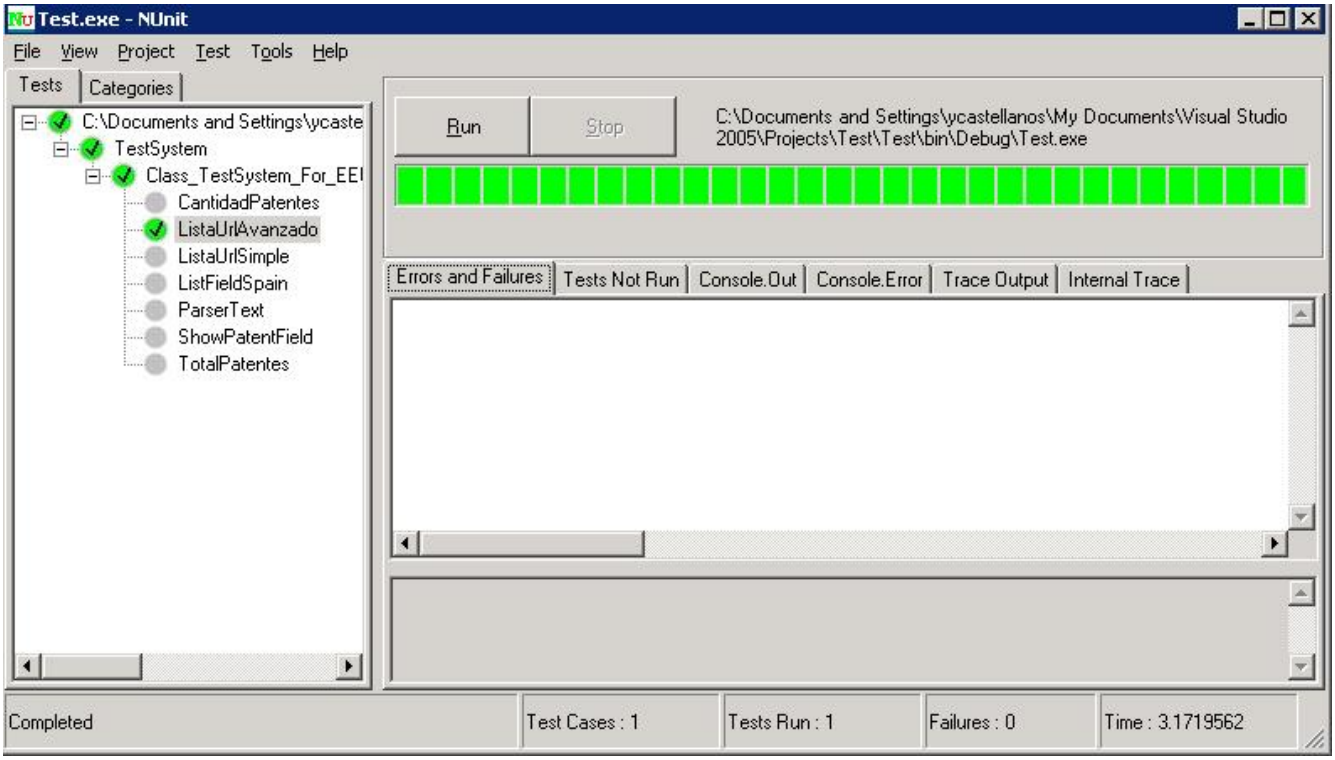


**Figura 3-5 Diseño del Caso de Prueba #5**

**Caso de Prueba #6**

Este caso de prueba es importante, pues se realiza con el objetivo de obtener, la lista de las urls de la base de datos de EEUU SIMPLE, que serán descargadas por el sistema, y así de esta forma poder realizar todo el proceso de búsqueda y selección de los campos de la patentes.

| **Caso de Uso** | Procesar Patentes | | | |
|---|---|---|---|---|
| **Caso de prueba** | ListadoUrlSimple() ver Figure25 Anexo | | | |
| **Condiciones** | Para que este caso de prueba se cumpla exitosamente, el sistema debe de haber descargado previamente la página inicial de búsqueda, que es quien contiene la lista de las primeras 50 patentes. | | | |
| **Clases Válidas** | **Clases Inválidas** | **Resultado Esperado** | **Resultado de la Prueba** | **Observ.** |
| URLEEUU | ParserSpain<br>TFichero<br>FachadaParser<br>ParserEEUU | Con esta prueba se espera que el sistema o el método de prueba devuelvan una lista de urls de patentes, lista para ser descargadas por el sistema. | El resultado de la prueba hecha al código fue el esperado, el caso de prueba devolvió y comprobó el resultado mostrado por el método, analizado | Cuando se realizo esta prueba con la herramienta NUnit se obtuvo el resultado correcto<br>(Ver figura 3-6). |
| **Descripción de los valores usados en la prueba** | Para la ejecución de esta prueba se usaron valores que no presentaban problemas, en este caso se utilizo un fichero previamente descargado por el sistema, este fichero es el que contiene el listado de las urls, lo que con la diferencia que pertenece a otra base de datos. | | | |

**Tabla 16 Caso de Prueba #6**

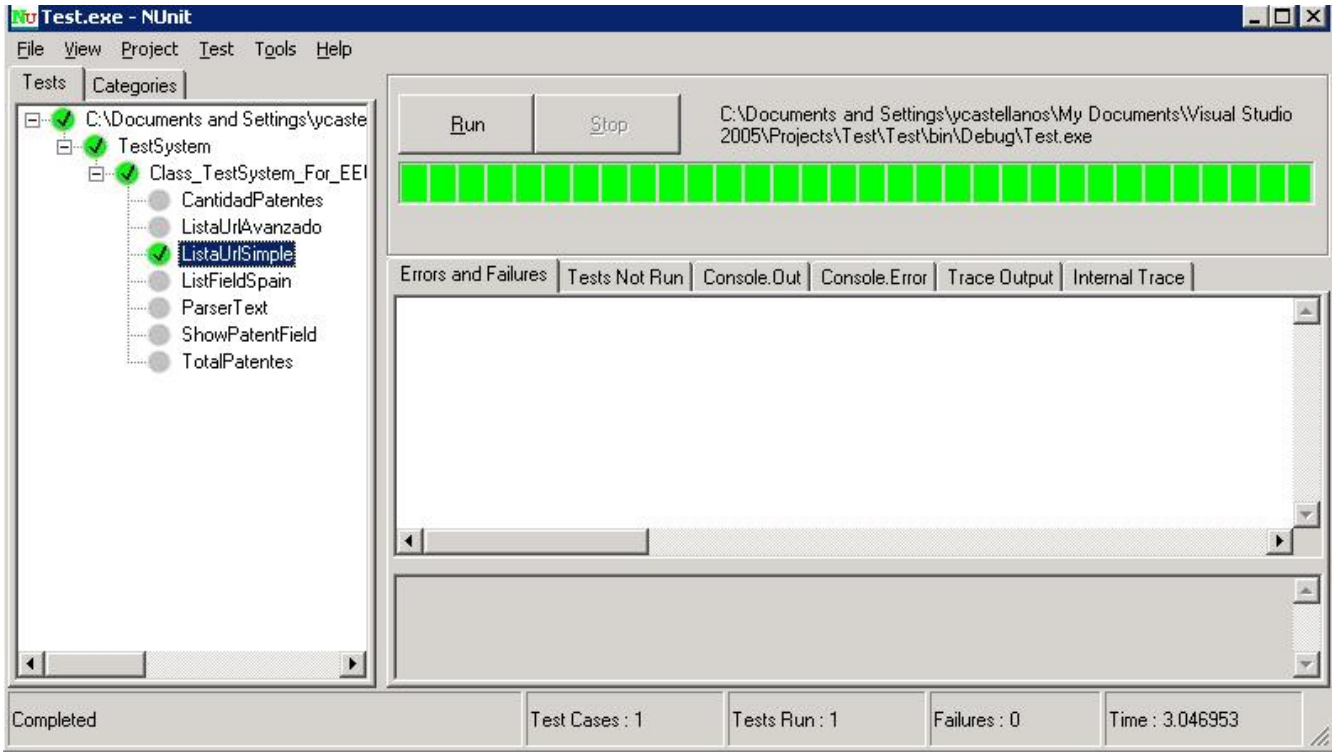


**Figura 3-6 Diseño del Caso de Prueba #6**

**Caso de Prueba #7**

Este caso de prueba es importante, pues este método devuelve una lista de campos de patentes, pero en la base de datos de España.

| | |
|---|---|
| **Caso de Uso** | Procesar Patentes |
| **Caso de prueba** | ListFieldSpain() ver Figure26 Anexo |
| **Condiciones** | Para que este caso de prueba se cumpla exitosamente, el sistema debe de haber |

| Clases Válidas | descargado previamente las patentes que serán analizadas. | | | |
|---|---|---|---|---|
| | **Clases Inválidas** | **Resultado Esperado** | **Resultado de la Prueba** | **Observ.** |
| ParserSpain | TFichero<br>FachadaParser<br>ParserEEUU<br>URLEEUU | Con esta prueba se espera que el sistema o el método de prueba devuelvan una lista de los campos de la patente en la base de datos de España. | El resultado de la prueba hecha al código fue el esperado, el caso de prueba devolvió y comprobó el resultado mostrado por el método analizado | Cuando se realizo esta prueba con la herramienta NUnit se obtuvo el resultado correcto<br>(Ver figura 3-7). |
| **Descripción de los valores usados en la prueba** | Para la ejecución de esta prueba, se usaron valores validos, en este caso se uso un fichero descargado por el sistema perteneciente a una de las patentes bajadas desde Internet. | | | |

**Tabla 17 Caso de Prueba #7**

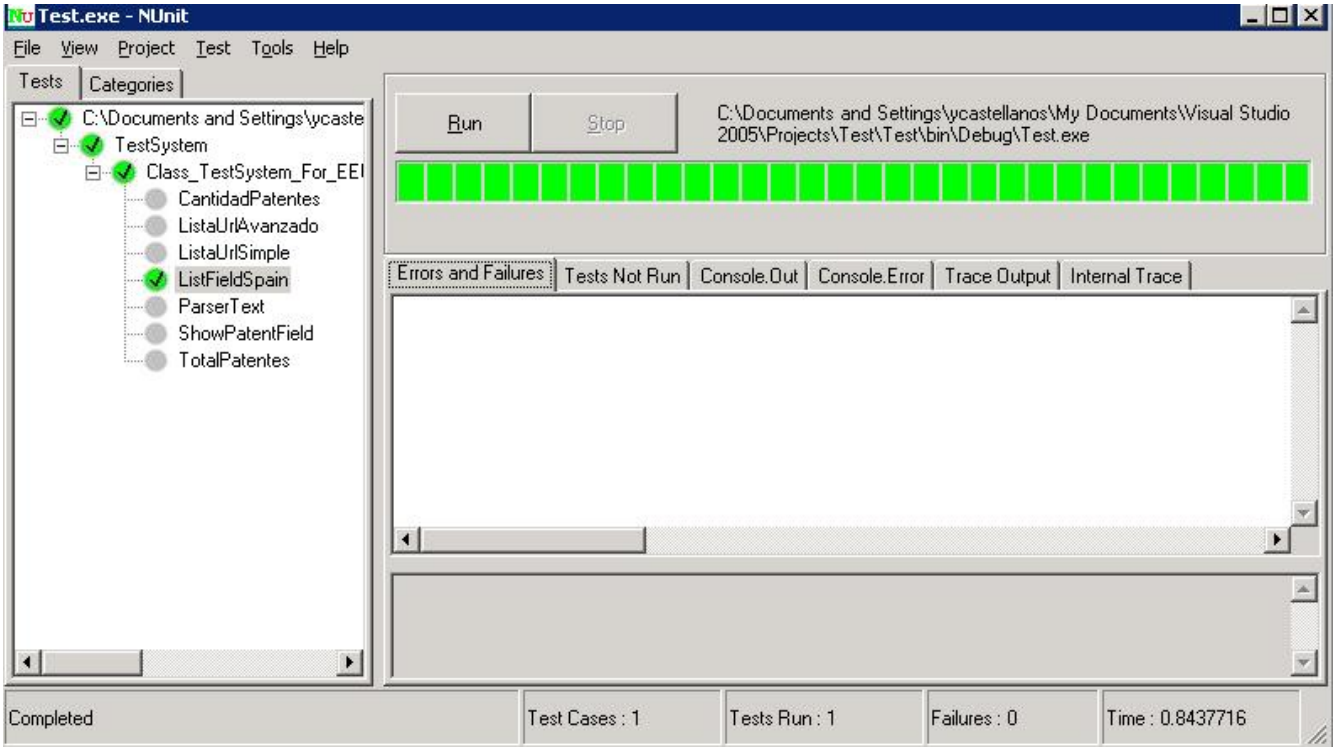


**Figura 3-7 Diseño del Caso de Prueba #7**

**Caso de Prueba #8**

Este caso de prueba es importante pues devuelve la cantidad de patentes que se encontraron en la búsqueda, y así de esta forma, mostrarle al usuario la cantidad de patentes que encontraron en la búsqueda que realizó el sistema.

| **Caso de Uso** | Procesar Patentes |
|---|---|
| **Caso de prueba** | TotalPatentes() ver Figure28 Anexo. |
| **Condiciones** | Para que este caso de prueba se cumpla exitosamente, el sistema debe de haber descargado previamente las patentes que serán analizadas. |

| Clases Válidas | Clases Inválidas | Resultado Esperado | Resultado de la Prueba | Observ. |
|---|---|---|---|---|
| URLEEUU | TFichero<br>FachadaParser<br>ParserEEUU<br>ParserSpain | Con esta prueba se espera que el sistema o el método de prueba devuelvan el número de patentes encontradas en la búsqueda. | El resultado de la prueba hecha al código fue el esperado, el caso de prueba devolvió y comprobó el resultado mostrado por el método analizado | Cuando se realizo esta prueba con la herramienta NUnit se obtuvo el resultado correcto<br>(Ver figura 3-8). |
| **Descripción de los valores usados en la prueba** | Para la ejecución de esta prueba, se usaron valores validos, en este caso se uso un fichero descargado por el sistema perteneciente a la pagina principal de búsqueda que es la contiene el total de patentes encontradas en la búsqueda realizada por el cliente. | | | |

**Tabla 18 Caso de Prueba #8**

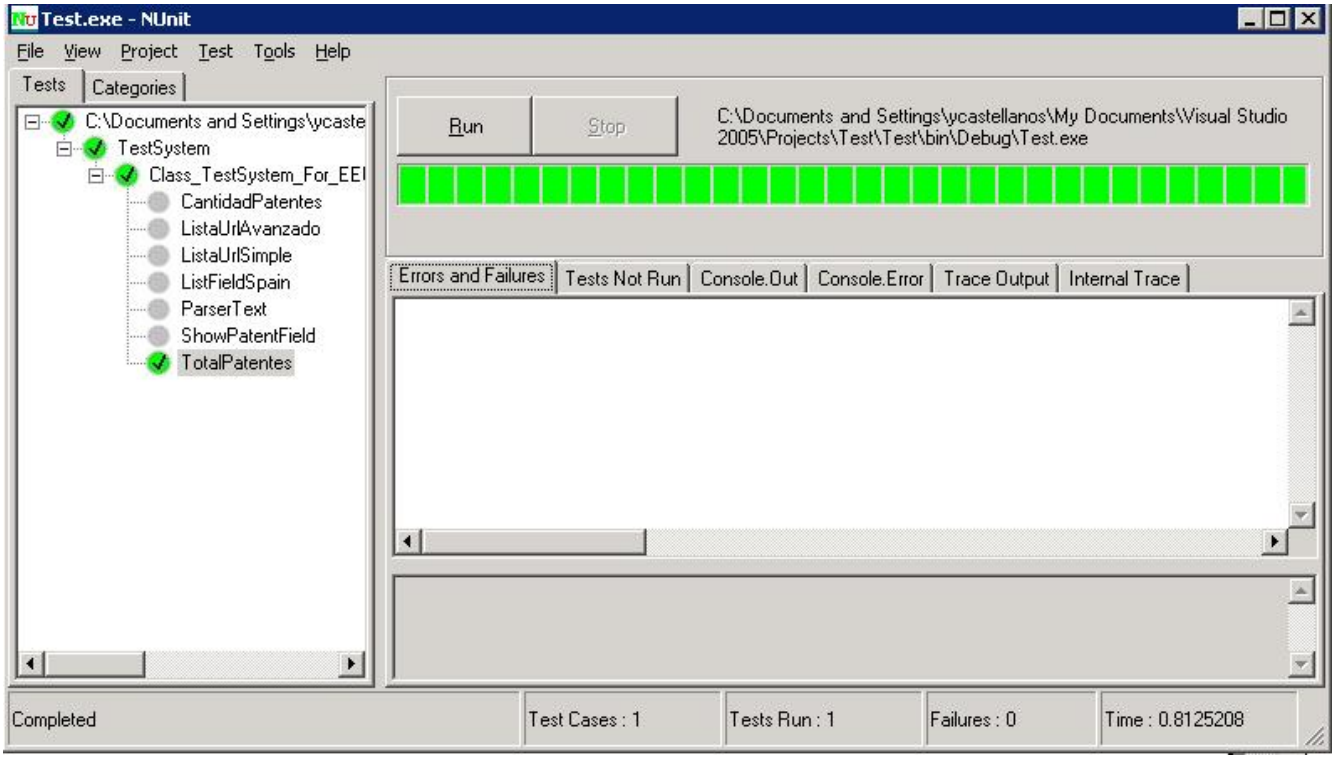


**Figura 3-8 Diseño del Caso de Prueba #8**

**Conclusiones Parciales del Capitulo 3.**

Al aplicarle las pruebas de unidad al Modulo de Parseo se llevaron a cabo las siguientes conclusiones:

- Todos los segmentos de códigos a los que se le aplicaron las pruebas de unidad cumplían el objetivo para el cual fue programado.
- Al aplicársele la prueba al Modulo de Procesamiento Html se probo que el mismo cumple con las funcionalidades para la cual fue diseñado.

## CONCLUSIONES GENERALES DEL TRABAJO

Con el desarrollo de este trabajo se llevó a cabo la implementación del Módulo de Parseo del Sistema Predictor. Se desarrolló un algoritmo para el procesamiento de información a partir de los estudios realizados, donde se escogió como punto de partida el algoritmo de Búsqueda Secuencial con Centinela como vía de solución a la principal funcionalidad de este Módulo que es la búsqueda y selección de los campos validos de las patentes definidos por el usuario. Se aplicó el estándar de codificación definido para lograr la uniformidad en el código de los desarrolladores, se elaboraron casos de prueba en la herramienta NUnit que permitieron verificar la eficacia del algoritmo desarrollado en el módulo. El desarrollo conjunto de estas tareas permitió la implementación del Módulo de Parseo del Sistema Predictor.

**RECOMENDACIONES**

- Migrar la solución hacia software libre o plataformas no propietarias.
- Continuar realizando investigaciones que permitan concluir la posibilidad de definir un algoritmo genérico para el procesamiento de la información proveniente de las distintas base de datos de patentes.
- Exportar hacia un fichero XML los datos de las patentes encontrados en la búsqueda.
- Implementar el módulo de acceso a datos del sistema.
- Realizar las Pruebas de Caja Blanca al Módulo de Parseo.

**GLOSARIO DE TERMINOS.**

**Abstracción**: Las características esenciales de un objeto que lo distinguen de todos los demás tipos de objetos y proporcionan así fronteras conceptuales definidas con nitidez en relación con la perspectiva del observador, la abstracción es uno de los elementos fundamentales del modelo de objeto (Booch 1996).

**Artefactos:** Pieza de información tangible, que puede ser usado en un proceso. Un artefacto puede ser un modelo, un elemento de un modelo, o un documento (Kruchten 2000).

**Patente:** Es un título que reconoce el derecho de explotar en exclusiva la invención patentada, impidiendo a otros su fabricación, venta o utilización sin consentimiento del titular.

**Paradigmas**: Los paradigmas son un conjunto de conocimientos y creencias que forman una visión del mundo (cosmovisión), en torno a una teoría hegemónica en determinado periodo histórico.

**Objeto:** es aquello que tiene estado (propiedades más valores), comportamiento (acciones y reacciones a mensajes) e identidad (propiedad que lo distingue de los demás objetos).

**Subsistema:** Una agrupación de elementos, de los que algunos constituyen una especificación del comportamiento ofrecido por los otros elementos contenidos (Ivar Jacobson 2000).

**OOP:** En sus siglas Programación Orientada a Objetos es, desde el punto de vista computacional un método de implementación en el cuál los programas son organizados como grupos cooperativos de objetos, cada uno de los cuales representa una instancia de alguna clase, y estas clases, todas son miembros de una jerarquía de clases unidas vía relaciones de herencia [Greiff 1994].

**Herencia:** Es el orden de las abstracciones organizado por niveles.

**CLI:** En sus siglas Common Language Infraestructure es una máquina virtual de lenguaje común de infraestructura, que contiene un cargador de clases, un compilador en tiempo de ejecución (JIT), y unas rutinas de recolección de memoria.

**Parseo:** La palabra Parseo en el diccionario de la Lengua Española no existe, pero en este trabajo se usa para agrupar las funcionalidades que se implementan sobre el código html de una patente descargada desde Internet, funcionalidades como búsqueda de distintos campos de la patente, búsqueda de las urls etc.

**BIBLIOGRAFIA**

BOOCH, G. Análisis y Diseño Orientado a Objetos. p.

IVAR JACOBSON, G. B., JAMES RUMBAUGH El proceso unificado de desarrollo de software, 2000.

PETER, W. The Ada Programming Language and Environment, unpublished draft. 1981. p.

PRESSMAN, R. Ingeniería de Software, Un enfoque práctico 5ta edición.

SABALLO, L. E. Sistema para la descarga y procesamiento automatizado de Patentes, 2007. p.

BOOCH, G. Object Solutions: Managing the Object-Oriented Project. Addison-Wesley. . 1996. p.

BOOCH, G. Software Architecture and the UML. 1998. p.

COAD, P. Object Models: Strategies Patterns and Applications. Prentice-Hall. . 1995. p.

CONALLEN, J. "Modeling Web Applications with UML" 1999.

ESCORSA, P. De la vigilancia tecnológica a la Inteligencia. Prentice- Hall. 2001. p.

FOWLER, M. Analysis Patterns: Reusable Object Models. Addison-Wesley. . 1996. p.

HORALLO, E. Introducción a Microsoft.NET.

JACOBSON, I. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley. 1992. p.

MENÉNDEZ, R. Metodologías Usadas en IWS 2002.

MEYER, B. Construcción de Software Orientado a Objetos. Prentice - Hall. p.

ODELL, J. M. Y. J. Object-Oriented Methods: A Foundation. Prentice-Hall. 1995. p.

RUMBAUGH, J. Object-Oriented Modelling and Design. Prentice-Hall. 1991. p.

Cormen, T. H. (2002). Introduction to Algorithms.

Hernandez, R. G. (2006). La información de marcas como indicador de innovación Tecnológica. Habana.

Greiff W. R. Paradigma vs Metodología; El Caso de la POO (Parte II). Soluciones Avanzadas. Ene-Feb 1994. pp. 31-39.

Erich Gamma, R. H., Ralph Johnson, John Vlissides, ForeWord by Grady Booch. Desing Patterns, Elements of Reusable Object-Oriented Software, 1994.

Piug. Patent Information Vendor Sites, 2006. [Disponible en: http://www.piug.org/vendor.html ].

UAM. Patentes. Bases de Datos Internacionales, 2006. [Disponible en: http://www.ibt.unam.mx/biblioteca/patentes.htm ]

OEPM. esp@cenet, 2006. [Disponible en: http://www.oepm.es/internet/bases_datos/espa.htm ].

esp@cenet. Introducción a espacenet Version 3, 2006 [Disponible en : http://es.espacenet.com/search97cgi/s97_cgi.exe?Action=FormGen&Template=es/ES/home.hts ]

esp@cenet. Introduction to esp@cenet, 2006. [Disponible en: http://es.espacenet.com/search97cgi/s97_cgi.exe?Action=FormGen&Template=es/ES/info.hts&INFO=intro ].

Delphion. PatentLab-II, 2006. [Disponible en: http://www.delphion.com/products/research/products-patlab ]

CDE. Xerka, 2006. [Disponible en: http://www.cde.es/index.php?option=com_content&task=view&id=10&Itemid=334 ].

CDE. Matheo Software, 2006. [Disponible en: http://www.cde.es/index.php?option=com_content&task=category&sectionid=13&id=67&Itemid=300 ]

CDE. Vigilancia de Patentes, 2006. [Disponible en: http://www.cde.es/index.php?option=com_content&task=view&id=20&Itemid=285 ].

Navarro, M N. Patentes: Protección e Información. Interés Empresarial, 2003. [Disponible en http://www.plantecnologico.com/pdf/patentes_proteccion_informacion.pdf.]

Mendoza, M. Metodologías de Desarrollo de Software, 2004. [Disponible en: http://www.informatizate.net/articulos/pdfs/metodologias_de_desarrollo_de_software_07062004.pdf ]

Dürsteler, J C Análisis de Patentes, 2005[Disponible en http://www.infovis.net/printMag.php?num=167&lang=1.]

Dürsteler, J C Análisis de Patentes, 2003. [Disponible en: http://www.cea.es/tecnologia/diagnosticos_tecnologicos/proteccion2.htm.]

BarzanallanaAsensio,R M. Paradigmas de programación y sus tipos, 2004 [ Disponible en: http://www.wikilearning.com/paradigmas_de_programacion_y_sus_tipos-wkccp-3618-3.htm.]

Potter, M. Algoritmos de Búsqueda, 2004[Disponible en:
http://www.conallen.com/technologyCorner/webextension/WebExtension091.htm.]