Universidad de las Ciencias Informáticas

Facultad 7

Implementación de una herramienta para viabilizar el proceso de pruebas de caja blanca

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

Autores: Yurién Ricardo Fuentes Guerra

Ernesto Jordán Borjas

Tutora: Ing. Lourdes Escalona Peral

Ciudad de La Habana, Julio de 2008

"Año 50 de la Revolución"

“La vida es muy peligrosa. No por las personas que hacen el mal, sino por las que se sientan a ver lo que pasa.”

Albert Einstein

“La prueba es una parte inevitable de cualquier esfuerzo responsable para desarrollar un sistema software.”

William Howden

DECLARACIÓN DE AUTORÍA

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

Para que así conste firmamos la presente a los 8 días del mes de Julio de 2008.

Yurién Ricardo Fuentes Guerra

Ernesto Jordán Borjas

Lourdes Escalona Peral

DATOS DE CONTACTO

Ing. Lourdes Escalona Peral:
Profesor graduado de Ingeniero Informático en el año 2004 en la Universidad de Holguín. Ha impartido las asignaturas Ingeniería de Software 1 y 2 y Seminario de Tesis. Fue líder del proyecto Atención Primaria de Salud durante dos años consecutivos y asesora de calidad de la facultad durante un año.
Ha tutorado tesis de perfil de análisis y diseño, implementación de sistema y atendiendo al área de calidad, Pruebas de software.
Posee la categoría docente de Instructor y cursa la maestría Gestión de Proyectos en la UCI. Se ha desempeñado como Jefe de Dpto. de la Especialidad de la Facultad 7, en los últimos 2 años.

Empresa: UCI Dirección: Carretera a San Antonio Km. 2 1/2 Reparto Torrens, Infraestructura productiva de la UCI, Ciudad Habana.
Teléfono: 835-8131. e-mail: lescalonap@uci.cu.

AGRADECIMIENTOS

A nuestro comandante en jefe Fidel Castro y a la Revolución por permitir que vivamos como personas dignas.
A mi mamá por aconsejar y estar presente en cada paso que dan sus hijos.
A mi novia por estar a mi lado en los buenos y malos momentos.
A mis amigos.
A Yurién por querer cambiarlo todo.
En general, a todos los que de una forma u otra han ayudado en mi trabajo.

Ernesto

Agradezco:
Al proyecto que representa la UCI por permitir algo no usual para los pobres, que la meta dependa solo del esfuerzo y la voluntad del estudiante.
A nuestra tutora Lourdes por su ética y profesionalidad.
A Ernesto por esforzarse.
A mis compañeros de todos los días.
A esos especiales amigos que tengo.
A mi novia por contagiarme de virtudes.
A mi familia por su cariño y apoyo.
A mi abuela y mi mamá por haberme enseñado que, como dijera Martí: “El hombre crece con el trabajo que sale de sus manos.”

Yurién

DEDICATORIA

A toda mi familia, en especial a mi mamá, mi papá y mis hermanos.

Ernesto

A mi mamá, que ha sido yo más que ella.

Yurién

RESUMEN

En los proyectos productivos de la Facultad 7 de la Universidad de las Ciencias Informáticas, las revisiones de software se basan principalmente en las pruebas de caja negra. Los integrantes de estos proyectos conocen poco sobre las pruebas de caja blanca y estas casi no se realizan. En los equipos de programadores se definen estándares de codificación pero no se comprueba adecuadamente su cumplimiento. Tampoco se realiza el control y la optimización de los parámetros de calidad del código fuente.

En consecuencia con lo anterior, se decidió implementar una herramienta que viabilice el proceso de pruebas de caja blanca.

La herramienta obtenida aplica la Técnica del Camino Básico hasta la generación de caminos independientes. Es capaz, además, de comprobar el cumplimiento en el código fuente, de los estándares de codificación establecidos y determinar parámetros de calidad. Se enriquece con la generación de reportes y con una interfaz amigable. Procesa código en lenguaje C#.

Específicamente, los resultados que obtiene son: la complejidad ciclomática de cada función, la matriz del grafo de flujo y un conjunto de caminos independientes. Se pueden definir y comprobar estándares de codificación que abarcan la mayoría de los estilos posibles en la escritura de código. Además, se obtienen y tabulan algunos parámetros de calidad como las diferentes variantes de la definición de Líneas de Código.

La implementación se realizó en lenguaje C# para la plataforma Microsoft .NET Framework y el entorno de desarrollo utilizado fue el SharpDevelop en su versión 2.2.1.2648.

PALABRAS CLAVES

Calidad de software, pruebas de caja blanca, código fuente, camino básico, complejidad ciclomática, parámetros del código, estándares de codificación.

TABLA DE CONTENIDO

INTRODUCCIÓN

Con la creación de la Universidad de las Ciencias Informáticas (UCI) el gobierno de Cuba pretende llevar a cabo un proceso amplio de informatización de la sociedad e ir construyendo poco a poco una productiva industria del software donde se obtengan productos de calidad para utilizarlos en el país y exportarlos hacia el mercado mundial y principalmente del área latinoamericana como parte de los principios de integración de la Alternativa Bolivariana para las Américas (ALBA).

Se han obtenido algunos resultados durante estos años en la producción de software para diferentes propósitos que han sido distribuidos principalmente en la República Bolivariana de Venezuela, país que los adquiere para realizar su proceso de informatización. Las principales empresas e instituciones estatales cubanas han realizado sus solicitudes a la UCI y se han mostrado satisfechos por los servicios recibidos desde este centro.

Por la necesidad de controlar la calidad de los productos y servicios que se brindan, en la UCI se ha creado el Departamento Central de Calidad que cuenta con grupos que asesoran la producción en cada facultad. En el Grupo de Calidad de la Facultad 7 se está llevando a cabo un proceso de capacitación y asesoramiento general en cada proyecto productivo para orientarlos hacia dos fines, la cultura de Calidad de software y la necesidad de realizar la Gestión de la calidad durante todo el desarrollo de productos informáticos.

La producción de la facultad 7 tiene un amplio perfil. Su principal especialidad es el desarrollo de software y servicios informáticos para la salud. Existen además, un área dedicada al procesamiento de imágenes y otra, como se explicó anteriormente, que garantiza la gestión de la calidad. Este perfil incluye que la programación se realiza para diversas plataformas y sistemas operativos, mediante diferentes lenguajes y técnicas de programación.

Para conocer el estado real del tema Calidad de software, se han hecho auditorías a todos los proyectos con resultados que indican la necesidad de mejoras en todos los procesos. En las evaluaciones realizadas se ha detectado que no se realizan pruebas documentadas al software. Solo se llevan a cabo algunas pruebas de caja negra, prácticamente no se utilizan las pruebas de caja blanca y en algunos casos se conoce muy poco del tema. Actualmente, en algunos proyectos se definen estándares de codificación pero no se comprueba su cumplimiento. En resumen, no se está controlando adecuadamente la calidad del código que se genera.

Entre los principales aspectos que validan un software, está el estado óptimo del código fuente con que se construyó. Para ello es preciso que todos los desarrolladores realicen la programación respetando estándares que se definen, documentan y comprueban. Para verificar la eficiencia de la aplicación en su etapa de desarrollo, se aplican las pruebas de caja blanca que deben su nombre a

que hace incidencia directa sobre el código. Éstas comprueban el funcionamiento y detectan errores en los bucles, condiciones, decisiones y todo lo relacionado a estructuras y variables. Es definitoria la técnica del camino básico definida por Thomas J. McCabe en el año 1976 que se basa plenamente en la teoría de grafos (1). Este procedimiento tiene como resultado un parámetro llamado complejidad ciclomática que tiene varios significados:

- Determina la complejidad lógica del software.
- Determina el número máximo de casos de prueba que se deben diseñar para probar el código.

Del código se pueden extraer otros parámetros como la densidad de comentarios y cantidad de líneas. Los comprendidos en las métricas de Halstead (2) y Chidamber & Kemerer (3) son muy utilizados también. Son parámetros que si se logran mantener con valores adecuados, se obtiene un código útil y eficiente.

Para llevar a cabo estas actividades, es ventajoso utilizar herramientas informáticas que automaticen estos procesos. Dichas aplicaciones son capaces de realizar cálculos grandes en muy poco tiempo y con gran exactitud. Con ello se lograría analizar gran cantidad de código en instantes y con independencia del lenguaje y plataforma a que pertenezca. Esto brinda confianza y certeza en el grado de calidad.

Entre las herramientas estudiadas se aprecia diversidad en cuanto a los lenguajes y plataformas que soportan. Existe, así mismo, diferencia en su disponibilidad. Algunas son libres y otras propietarias. Algunas son eficientes y útiles pero su interfaz y modo de operar no son sencillos.

Resumiendo, en los proyectos productivos de la facultad 7 se aprecia que existe diversidad en lenguajes de programación y plataformas así como deficiencias en la aplicación de pruebas de caja blanca y la comprobación de estándares de codificación y el control de parámetros de calidad en el código que se genera.

Como parte del Grupo de Calidad de la Facultad 7 se tomó como problema a resolver:

¿Cómo implementar una herramienta que viabilice el proceso de pruebas de caja blanca?

Este trabajo toma como objeto de estudio: Los procesos de revisiones de software.

El campo de acción es: El proceso de pruebas de caja blanca.

El objetivo general planteado es: Implementar una herramienta que viabilice el proceso de pruebas de caja blanca.

Para este fin se realizan las siguientes actividades:

- Investigación sobre las técnicas, lenguajes y plataformas de programación existentes y determinación de cuáles son más adecuados para cumplir los objetivos del trabajo.

- Implementación de las funcionalidades establecidas para la herramienta.
- Descripción de las principales clases y algoritmos del sistema.
- Análisis de los resultados obtenidos.

El contenido del trabajo se encuentra distribuido en el documento de la siguiente manera:

Capítulo 1. Fundamentación teórica:
Se realiza un análisis valorativo del estado del arte correspondiente a las técnicas de programación y lenguajes usados en el mundo, en Cuba, en la UCI y en la facultad 7.
Se analizan y explican las técnicas de programación, lenguaje y librerías usadas para la implementación de la herramienta.

Capítulo 2. Descripción y análisis de la solución propuesta:
En este capítulo se realiza una valoración del diseño que propone el analista y se abordan los componentes que pueden ser rehusados. Además se establecen posibles integraciones con otros sistemas que principalmente son editores de código. Se abordan los aspectos técnicos referentes al objetivo de cada módulo. Se definen las características generales de la herramienta que se implementa y se establecen los estándares de codificación a utilizar.

Capítulo 3. Implementación del sistema:
Se detallan los aspectos fundamentales que definen la estructura de implementación del sistema. Se analizan los algoritmos más importantes. Se describen las clases que modelan la solución y sus características. Se explican las estructuras de datos empleadas para manejar la información.

Capítulo 4. Validación de la solución propuesta:
En este capítulo se diseñan y ejecutan pruebas unitarias para comprobar los resultados obtenidos. Luego se analizan estos resultados y se llega a conclusiones de lo logrado en el trabajo. En dependencia de los resultados se proponen nuevas iteraciones y posibles cambios en el diseño.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se analizan las características de las diferentes técnicas y lenguajes de programación utilizados actualmente en el mundo y en la UCI. Se analiza el paradigma de programación orientado a objetos y los lenguajes principales exponentes del mismo así como las plataformas que los soportan. Se exponen algunas de las herramientas de desarrollo, que más se ajustan a los propósitos de ese trabajo. Finalmente se realiza un estudio crítico y valorativo de las librerías usadas en la programación de la herramienta.

1.1 Paradigma de programación aplicado

La Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es el paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo, y encapsulamiento (4).

Este paradigma expresa un programa como un conjunto de objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

De esta forma, un objeto contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases e incluso frente a objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos. A su vez, los objetos disponen de mecanismos de interacción llamados métodos que favorecen la comunicación entre ellos. Esta comunicación a su vez facilita el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan ni deben separarse el estado y el comportamiento.

Los métodos (comportamiento) y atributos (estado) están estrechamente relacionados por la propiedad de conjuntos. Esta destaca que una clase requiere de métodos para poder tratar los atributos con los que cuenta. El programador debe pensar indistintamente en ambos conceptos, sin separar ni darle mayor importancia a ninguno de ellos, hacerlo podría producir el hábito erróneo de crear clases contenedoras de información por un lado y clases con métodos que manejen a las primeras por el otro. De esta manera se estaría realizando una programación estructurada camuflada en un lenguaje de programación orientado a objetos y no se alcanzarían los resultados esperados.

La POO es un concepto avanzado en la programación, que en sus inicios hizo que antiguos programadores cambiaran su forma de pensar a la hora de programar, pero una vez que se domina y se aprovecha al máximo se obtienen grandes ventajas.

Las clases tienen características que son propias y pudieran servir en muchos otros contextos; de ahí que se pueden reutilizar tal como son o quizás con alguna pequeña adaptación. Los elementos en instrucciones están mejor organizados y por esto encontrar errores es más sencillo.

La herencia es otra de las grandes ventajas al poder derivar clases de otras con todas sus propiedades y características, ampliarlas o rediseñarlas en las nuevas clases que se denominan hijas de la primeras. Esta última posibilidad ahorra bastante tiempo en la implementación de clases.

Por ejemplo: Existe una clase *Triangulo* (se ha escrito sin tilde por ser conveniente no incluir signos en los identificadores) que tiene dos atributos *base* y *altura* respectivamente e implementa un método *Area* que retorna la multiplicación de ambos: *A = (base * altura)*. De *Triangulo* heredan *TrianguloEquilatero*, *TrianguloIsosceles*, *TrianguloEscaleno* y *TrianguloRecto* que tendrían igualmente *base, altura, Area* y agregan otras propiedades. Como se sabe, el área de un triángulo es: *A = (base * altura) / 2*. En la implementación del método *Area* hubo un error y al corregirlo tendría que modificarse en todos los hijos de *Triangulo* si no existiera la herencia.

1.2 Lenguajes de programación

A continuación se aborda sobre los principales lenguajes de programación de alto nivel y orientados a objeto, utilizados por la mayoría de los programadores en el mundo.

1.2.1 El lenguaje C++

El lenguaje C++ se comenzó a desarrollar en 1980 por Bjarne Stroustrup de la AT&T[1] primero con el nombre de “C with classes” debido a que su objetivo era mejorar algunas características del C pero manteniendo su basamento en el mismo. Luego, en 1983 adquiere el nombre de “C++” debido a su esencia de C y con mejoras como su operador de incremento numeral (++). Este se estandarizó hacia 1989 por sendos comités de la ANSI[2] y la ISO[3] y se enriqueció con la aparición de la STL[4] desarrollada por Alexander Stepanov y publicada en 1994.

[1] Bajo el nombre American Telephone and Telegraph es la más importante empresa de telecomunicaciones en Estados Unidos.

[2] Instituto Nacional Estadounidense de Estándares (en Inglés American National Standards Institute) es una organización sin ánimos de lucro que supervisa el desarrollo de estándares para productos, servicios, procesos y sistemas en los Estados Unidos.

Es un lenguaje de alto nivel pero manteniendo la posibilidad de realizar operaciones a nivel de máquina. Es imperativo, orientado a objetos y de propósito general al que se le han añadido nuevos tipos de datos, clases, plantillas, mecanismo de excepciones, sistema de espacio de nombres, funciones *inline*, sobrecarga de operadores, referencias para el manejo de punteros, manejo de memoria persistente y algunas utilidades adicionales de su librería estándar. Simplifica y enriquece el uso de comentarios, la declaración de enumeradores, la declaración de variables y la declaración de estructuras (5).

Es versátil, flexible, conciso y muy eficiente. Por muchos años fue el preferido en el desarrollo de aplicaciones. Se ha utilizado para implementar el núcleo de sistemas como Windows y Java.

1.2.2 El lenguaje Java

Este es un lenguaje desarrollado por la compañía Sun Microsystem en los años noventa. Está inspirado en C++ y se proyectó con la finalidad de obtener un producto de pequeñas dimensiones, simple y portátil sobre diferentes plataformas y sistemas operativos ya sea a nivel de código fuente como a nivel de código binario.(6)

Los compiladores de Java generan archivos de código binario especial (Java bytecode) que es un lenguaje de máquina a bajo nivel y puede incluso ser interpretado directamente por un procesador.

Las aplicaciones desarrolladas en Java se deben ejecutar en la Máquina Virtual de Java, en inglés Java Virtual Machine (JVM). Este es un programa que se puede ejecutar sobre varios sistemas operativos y su función es servir de puente o mediador que puede entender tanto el Java bytecode como el sistema sobre el que se quiere ejecutar el mismo. Permitiendo así la portabilidad de programas escritos en este lenguaje. JVM posee instrucciones para los diferentes tipos de tareas:

- Carga y almacenamiento.
- Operaciones aritméticas.
- Conversiones de tipos.
- Creación y manipulación de objetos.
- Gestión de pilas con métodos como *Push* y *Pop*.

---

[3] Organización Internacional para la Estandarización (en Inglés International Organization for Standardization) es un organismo encargado de promover normas internacionales de fabricación, comercio y comunicación para todas las ramas industriales excepto la eléctrica y la electrónica.

[4] Librería estándar de plantillas (en Inglés Standard Template Library) es un conjunto de estructuras de datos y algoritmos genéricos.

- Transferencias de control de programa (*branching*).
- Invocación y retorno a métodos.
- Lanzamiento de excepciones.

Entre sus principales características se encuentra que es un lenguaje de propósito general y orientado a objetos. Su sintaxis ha sido trabajada mejorando la de C++ logrando mayor sencillez y legibilidad. Presenta mayor robustez al simplificar la gestión de memoria y eliminar las complejidades del manejo explícito de punteros. Presenta capacidades avanzadas de ejecución *multi-hilo* y proporciona mecanismos de carga dinámica de clases en tiempo de ejecución. Se puede compilar y ejecutar en cualquier plataforma de sistema operativo por ejemplo en Windows, Solaris o Linux gracias a su máquina virtual. Su desarrollo ha sido rápido y exitoso debido a la gran cantidad de grandes empresas colaboradoras que han dado su aporte para enriquecerlo.

Es un lenguaje gratuito ya que Sun Microsystems distribuye gratuitamente su producto base, el J2SE (en Inglés Java 2 Standard Edition). Este es un paquete que incluye herramientas para generar programas Java con depurador, compilador y herramienta para la documentación de código. Además posee la JVM, la jerarquía de clases (API[5] de Java) con su código fuente y la documentación necesaria. (7)

La opinión de la mayoría de los programadores es que se torna más potente en el desarrollo de aplicaciones Web y ha sido mayormente orientado en ese sentido por sus promotores.

La ejecución de programas escritos en Java suele comportarse más lenta que la de aplicaciones de otro lenguaje haciendo un uso voraz de recursos como memoria y procesador. Esto se hace más notorio si la ejecución se basa en cálculos matemáticos complejos o si la aplicación presenta un diseño cargado de componentes visuales.

1.2.3 El lenguaje C#

Actualmente se está utilizando con gran efectividad el nuevo lenguaje C# desarrollado por la empresa Microsoft Corporation, como una recopilación de lo mejor de C++ y Java. Este lenguaje pertenece a la POO, tiene una sintaxis muy parecida a Java y posee la potencia de C++. Tiene algunas ventajas sobre los restantes lenguajes de alto nivel orientados a objeto. El estándar del lenguaje C# por

[5] Interfaz de Programación de Aplicaciones (en Inglés Application Programming Interface) es un conjunto de funciones que ofrece una librería para usar en programas como una capa de abstracción de datos.

excelencia está comprendido en la especificación ECMA-334[6] de la Ecma International[7] que hasta 1994 se llamó European Computer Manufacturers (8). Entre sus principales especificaciones están:

- El ECMA-262 como especificación para el lenguaje de programación ECMAScript.
- El ECMA-334 como especificación para el lenguaje de programación C#.
- El ECMA-335 como especificación para el CIL[8].

1.2.3.1 La plataforma .NET

En 1998 un equipo de trabajo de Microsoft Corporation comenzó a trabajar en el proyecto Próxima Generación de Servicios Windows (en Inglés Next Generation Windows Services) el cual se fusionó con el grupo encargado de liberar Visual Studio 7 con el objetivo de desarrollar un entorno común de ejecución para todos los lenguajes cubiertos por esta herramienta de desarrollo que permitiera a terceras empresas crear lenguajes adaptados al entorno. Luego, en el año 2000 Microsoft publicó este trabajo denominado Microsoft.NET. Este conjunto novedoso agrupa las siguientes tecnologías:

- Plataforma .NET.
- SDK de la plataforma .NET.
- Visual Studio .NET.
- Servicios Web.
- Servidores empresariales.(9)

La plataforma .NET sirve de mediador entre el programador y las particularidades del sistema operativo para el que se programen las aplicaciones. Una vez terminado el programa, su ejecución se realizaría sobre esta plataforma que entonces mediaría entre el mismo y el sistema operativo. De esta forma, un sistema desarrollado para .NET pudiera ejecutarse en cualquier sistema operativo que tenga instalada una versión de este framework como también se le denomina.

Está diseñado para utilizar los servicios web XML[9] como mecanismo principal de comunicación entre aplicaciones. Posee avanzadas funciones en tiempo de ejecución lo que permite que cualquier aplicación pueda ser convertida en servicios web XML. Permite escribir programas en cualquiera de los

---

[6] Estándar internacional que especifica la representación y semántica de programas escritos en C# así como la sintaxis y restricciones de este lenguaje.

[7] Organización internacional que se basa en membrecías de estándares para la comunicación y la información.

[8] Código intermedio generado a partir de un programa escrito en cualquier lenguaje de programación de alto nivel sobre la plataforma .NET y que luego se convierte a lenguaje de máquinas de bajo nivel.

[9] Lenguaje de marcas extensible (en Inglés Extensible Markup Language) muy usado para la transmisión de datos entre aplicaciones.

lenguajes soportados por la plataforma, incluso utilizar simultáneamente varios lenguajes en un mismo programa. Entre los lenguajes que se han adherido a esta familia se encuentra el ya mencionado C#. Agrupa además al Microsoft Visual Basic, C++, Java, Pascal, entre otros conformando un grupo de más de veinte. Para su óptimo aprovechamiento se desarrolló Visual Studio 2005 (esta suite de desarrollo será abordada en la sección 1.3.1) que ha evolucionado hasta las actuales versiones de Visual Studio 2008.

Acciones como conexiones a bases de datos y la creación de componentes visuales se encuentran empaquetadas en componentes que tiene implementadas todas las funciones necesarias para estos propósitos. Solo basta con arrastrarlos hacia la aplicación y utilizar sus ventajas. Realiza una adecuada gestión de memoria haciendo las aplicaciones más confiables. Utiliza la ejecución en paralelo aportando mayor eficiencia en tiempo de ejecución.

Tiene un componente de seguridad capaz de monitorear las acciones que pueden ser sensibles sobre el sistema operativo controlando quién escribe y ejecuta el código y con qué propósitos lo hace. Inicialmente el framework no era gratuito, más tarde Microsoft liberó la versión 2.0 y actualmente se encuentra disponible para su descarga desde el portal web de esa empresa (10).

1.2.3.2 Comparación del C# con C++ y Java

A continuación se expone una comparación del C# frente a otros lenguajes, utilizando sus principales características, muy discutidas por diferentes programadores que lo han utilizado para realizar disímiles tareas y en condiciones muy diferentes.

Ventajas frente a C/C++:

- Compila a código intermedio (CIL) independiente del lenguaje en que haya sido escrita la aplicación e independiente de la máquina donde vaya a ejecutarse.
- Realiza la recolección automática de basura.
- Elimina el uso de punteros, en C# no son necesarios aunque permite utilizarlos.
- Posee capacidades de reflexión.
- No hay que preocuparse por archivos de cabecera ".h".
- Es flexible en cuanto al orden de definición de las clases y las funciones.
- No hay necesidad de declarar funciones y clases antes de invocarlas.
- No existen las dependencias circulares.
- Soporta definición de clases dentro de otras.
- No existen funciones ni variables globales, todo pertenece a una clase.

- Todos los valores son inicializados antes de ser usados (automáticamente por defecto, o manualmente desde constructores estáticos).
- No se pueden utilizar valores no booleanos (enteros, coma flotante) para condicionales.
- Es mucho más limpio y menos propenso a errores.(11)

Ventajas frente al C++ y Java:

- Concepto formalizado de los métodos *get* y *set*, con lo que se consigue código mucho más legible.
- Gestión de eventos usando delegados mucho más fácil de implementar.

Ventajas frente a Java:

- El rendimiento es, por lo general, mucho mejor.
- CLI está estandarizado, mientras que los *bytecode* de java no lo están.
- Soporta más tipos primitivos (en Inglés *value types*), incluyendo tipos numéricos sin signo.
- Se usan indizadores que permiten acceder a cualquier objeto como si se tratase de un arreglo.
- Compilación condicional.
- Aplicaciones *multi-hilo* más sencillas de manejar.
- Soporta la sobrecarga de operadores, que aunque pueden complicar el desarrollo son opcionales y algunas veces muy útiles.
- Permite el uso de punteros cuando realmente se necesiten, como al acceder a librerías que no se ejecuten sobre la máquina virtual.

De manera subjetiva es posible decir que este nuevo lenguaje utilizado en aplicaciones como Microsoft Visual Studio 2005 (más adelante será abordada esta herramienta de desarrollo) simplifica la labor de la programación y minimiza la posibilidad de errores porque todo se ve más sencillo y real. Este lenguaje se desarrolló pensando en las necesidades del programador.

Entre las pocas desventajas expuestas por el C# está que es muy orientado a ambientes del sistema operativo Windows. Y no es el más recomendado para sistemas embebidos y núcleos de sistemas.

1.2.3.3 Utilización del lenguaje C# y la tecnología .NET en la facultad 7 de la UCI

En la facultad 7 de la UCI existe un perfil variado que se dedica a la producción de software para los sistemas de salud. Una de las especialidades presentes en estos proyectos es el procesamiento digital de imágenes que basa su desarrollo en el uso de la tecnología .NET, aprovechando todas las ventajas que esta brinda y obteniendo resultados relevantes avalados por la calidad y aceptación de los diferentes sistemas desarrollados hasta ahora. Dentro de ellos, el ya conocido Alas PACS. Este

producto consiste en un Sistema para el almacenamiento, transmisión y visualización de imágenes médicas desarrollado por el Grupo de procesamiento de imágenes y señales (GPI) de la UCI.
Adicionalmente se encuentran en desarrollo los siguientes proyectos:

- AlasNEURO Diana. Sistema para el Diagnóstico Neuro-Psicológico Automatizado Diana.
- Sistema de procesamiento, fusión y visualización tridimensional de neuroimágenes.
- Sistema para el control y seguimiento de la actividad quirúrgico oftalmológico.

1.2.4 Selección de un lenguaje de programación

Finalmente se ha comprobado que el lenguaje C++ es muy potente y estable pero complicado para algunos propósitos. Java es muy similar al C# y su sintaxis es amigable pero eventualmente es menos eficiente en la ejecución de programas de escritorio frente a los desarrollados en C#. Además, este último aporta características novedosas que simplifican grandemente las labores de implementación. Se enriquece con la potencia de C++ y la sencillez y modernidad de Java pero mejorando lo que debe ser mejorado. Ha alcanzado gran auge y se utiliza para desarrollar gran cantidad de aplicaciones desde mediano a gran tamaño, obviando esfuerzo de programadores en actividades de rutina. Está debidamente estandarizado. Se usa en múltiples grupos de proyecto en la facultad 7 de la UCI. Es gratis tanto el C# como la plataforma que lo soporta.
Resulta, por tanto, el adecuado para las aspiraciones y necesidades de este trabajo.

1.3 Entornos de desarrollo

Un Entorno de Desarrollo Integrado, Integrated Development Environment (IDE) es un programa que agrupa un conjunto de herramientas que viabilizan la labor de los programadores. Los componentes esenciales que los definen son:

- Editor de texto (código).
- Compilador.
- Intérprete.
- Depurador.
- Herramientas de automatización (completamiento de código y navegación rápida dentro del código).
- Sistema de control de versiones.
- Diseñador de interfaces gráficas.
- Agregación de herramientas externas.
- Soporte para varios lenguajes de programación y plataformas de desarrollo.

No todos poseen las anteriores propiedades ni las implementan al mismo nivel, pero esas son las más usuales.

A continuación se exponen los IDE más usados para generar aplicaciones en lenguaje C# para la plataforma .NET. Se realiza una descripción de los mismos teniendo en cuenta los aspectos antes expuestos. Finalmente se da una valoración de cada uno y se escoge el adecuado para la actividad de implementación de este trabajo.

1.3.1 Entornos propietarios

Microsoft Visual Studio:

Es un IDE para sistemas Windows desarrollado por Microsoft Corporation. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET[10] y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Visual Studio 2005 permite a los desarrolladores crear sitios, aplicaciones y servicios web y aplicaciones de escritorio en cualquier entorno que soporte la plataforma. Sobre el mismo se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles (12).

Los programadores de Windows encuentran los nuevos formularios *Windows Forms* intuitivos y muy eficaces. Estos son compatibles con cualquier lenguaje basado en .NET. Con la herencia visual, los programadores pueden simplificar enormemente la creación de aplicaciones basadas en Windows, centralizando en formularios primarios la lógica común y la interfaz de usuario para toda la solución. Utilizando delimitadores y acoplamiento, los programadores pueden generar formularios redimensionables automáticamente, mientras el editor de menús permite crear menús de manera visual directamente desde el *Diseñador de Windows Forms*.

Visual Studio .NET continúa siendo el punto de referencia para la productividad de los programadores. Con un único entorno de programación (IDE) integrado para todos los lenguajes, las organizaciones de programación pueden aprovechar las ventajas de un cuadro de herramientas, un depurador y una ventana de tareas comunes, reduciendo enormemente la curva de aprendizaje del programador y garantizado que siempre puedan elegir el lenguaje más apropiado para sus tareas y conocimientos. Con la función para completar instrucciones de *IntelliSense* y la comprobación automática de errores de sintaxis, Visual Studio .NET informa a los programadores cuándo el código es incorrecto y proporciona el dominio inmediato de las jerarquías de clases y las API. Con el Explorador de

[10] Plataforma de desarrollo de aplicaciones Web desarrollado y distribuido por Microsoft Corporation que brinda la posibilidad de desarrollar aplicaciones Web dinámicas y estáticas así como servicios Web XML.

soluciones, los programadores pueden reutilizar fácilmente código a través de diferentes proyectos e incluso, generar soluciones multilenguaje que satisfagan con mayor eficacia sus necesidades empresariales.

Gracias al entorno totalmente extensible, se puede disfrutar de las ventajas de una activa comunidad de componentes y complementos de otros fabricantes, con componentes y controles que contribuyen a personalizar y ampliar el entorno de acuerdo con las necesidades.

Los asistentes para aplicaciones, las plantillas de proyecto y los ejemplos de código fuente de Visual Studio .NET permiten crear aplicaciones con rapidez para Windows, la Web y dispositivos con una inversión inicial mínima. La ayuda dinámica y Microsoft Developer Network (MSDN®)[11] proporcionan asistencia basada en la tarea y el lenguaje de programación actuales, garantizado que los programadores sepan aprovechar las oportunidades de la plataforma Microsoft .NET en el lenguaje que hayan elegido. Las macros de Visual Studio, similares a las macros de Microsoft Visual Basic para Aplicaciones en Office, permiten la automatización de tareas rutinarias dentro del IDE, mejorando aún más la productividad global de los programadores de Visual Studio.

La versión Visual Studio 2005 se puede adquirir en las siguientes ediciones:

- Visual Studio 2005 Team System: Para grandes equipos de desarrollo.
- Visual Studio 2005 Professional: Para programadores independientes o pequeños grupos de programadores con múltiples propósitos.
- Visual Studio 2005 Standard y Visual Studio 2005 Express: Ediciones más simples para usuarios que se inician y estudiantes principalmente sin grandes propósitos.

Los precios de compra de las mismas oscilan entre los $300 dólares y los $1 400 dólares desde las ediciones Express y Standard hasta la Team System en dependencia del distribuidor, los elementos que incluya el paquete y otros términos a tener en cuenta(13).

C# Builder 2006:

CodeGear es una filial de la empresa estadounidense desarrolladora de herramientas para la programación Borland Software Corporation. La misma liberó, entre otros paquetes para diferentes lenguajes, el C#Builder 2006 para el lenguaje C#.

Se encuentra disponible en varias versiones especializadas como C#Builder Professional, C#Builder Enterprise, C#Builder Architect (14).

[11] Recurso muy útil para el desarrollo mediante herramientas de Microsoft que contiene información técnica sobre programación con código de ejemplo, documentación, artículos técnicos y guías de referencia.

En general esta opción se ajusta a proyectos con vista a obtener aplicaciones para la plataforma .NET con buenas capacidades de desarrollo empresarial y en la gestión y mantenimiento de proyectos de C# o aquellos que se encuentran en transición de Java y C++ a C#. Entre las características a destacar se encuentran el mapeado relacional de objetos, la persistencia de datos, el modelado UML[12], la implementación a nivel de diagramas de estados, la gestión de requisitos y la gestión de código fuente. Descargar la aplicación tiene un costo superior a los $1 000 dólares.

1.3.2 Entornos libres

SharpDevelop:

Es un IDE desarrollado por ICSharpCode Team que soporta el desarrollo de aplicaciones escritas en C#, Visual Basic.NET y BOO[13]. Está desarrollado en C# y es libre (ver sección 1.3.3.1). Proporciona todas las características necesarias para un entorno de programación de Windows, entre ellas:

- Autocompletado de código: Esta funcionalidad simplifica el esfuerzo y tiempo de codificación ya que el programador va escribiendo el inicio de las palabras y el editor de texto, ya sea automáticamente o mediante teclas de acceso rápido, despliega un menú contextual con las diferentes palabras que se asocian al lexema que se ha escrito y al contexto en que lo hace.
- Plantillas de proyectos: Estas plantillas son programas ya comenzados con gran parte de su encabezamiento y primeros elementos especificados y que el programador no tendrá que escribir reduciendo tiempo y esfuerzo de codificación.
- Depurador integrado: Esta es una de las ventajas que más se esperan de un buen IDE ya que permite probar el funcionamiento paso a paso de la sección del programa que se desee. Basta con poner los puntos de ruptura en los sitios escogidos y ordenar la depuración dando la orden a cada paso pudiendo observar los valores que van tomando las variables y pasar por alto llamadas a funciones o insertarse en la ejecución de las mismas. Esto es muy útil para probar algoritmos complejos y da la posibilidad de encontrar un error en mucho menos tiempo que analizando el algoritmo mentalmente.
- Diseñador de formularios: El diseñador de formularios permite conformar la interfaz visual de las aplicaciones de forma rápida y real porque la persona que diseña va viendo la apariencia de su diseño a medida que incorpora y ajusta elementos en la ventana. Paralelo a esto, la

[12] Lenguaje Unificado de Modelado (en Inglés Unified Modeling Language) para la especificación, construcción y visualización de elementos en el desarrollo de software.

[13] Lenguaje de programación orientado a objetos de tipos estáticos muy parecido a Phyton e integrable con Microsoft .NET y Mono.

herramienta va generando el código fuente correspondiente a cada componente. De esta forma también se ahorra tiempo y esfuerzo.

- Tiene herramientas para *ir a definición*, *encontrar referencias* y *renombrado* que permite desplazarse rápidamente por el código yendo a los puntos deseados con solo dar clic derecho y escoger una de estas opciones. La primera permite desde cualquier invocación a método o variable, ir a su definición. La segunda permite lo contrario, desde el nombre de la función o variable, encontrar todas sus invocaciones hechas en el código. La tercera permite que al cambiar el nombre del identificador de alguno de estos elementos, se realicen los cambios automáticamente a todas las ocurrencias de los mismos.
- Integración con herramientas externas: Muy utilizado a la hora de incorporar aplicaciones principalmente que se dedican a la administración y pruebas de código.
- Pose analizadores para ensamblado.

SharpDevelop es compatible con Visual Studio Express y Visual Studio 2005 por lo que es posible trabajar con proyectos indistintamente en uno u otro ambiente sin cambiar el formato de archivos de proyecto y de código. Su objetivo es brindar una opción para desarrolladores que no tienen la posibilidad de adquirir el Visual Studio.

Soporta además los siguientes lenguajes: HTML, ASP, ASP.NET, VB Script y XML con sintaxis destacada (escritura de las palabras en diferentes colores en dependencia de su tipo dentro del lenguaje) para todos ellos. Es capaz de convertir código C# a Visual Basic .NET y viceversa y de cualquiera de estos dos hacia Boo. Permite desarrollar proyectos para las siguientes plataformas: Microsoft .NET Framework 1.1 y 2.0, Microsoft .NET Compact Framework 1.0 y 2.0 de la familia .NET; Mono 1.1 y 2.0 de la familia Mono[14].

El proyecto se comenzó a desarrollar el 11 de septiembre de 2000 y actualmente se está implementando la versión 3.0 siendo SharpDevelop 2.2.1.2648 del 8 de agosto de 2007 la última versión disponible (15).

Actualmente se encuentra publicado en el sitio de su equipo de desarrollo y libre para descargar bajo la licencia GNU LGPL (ver sección 1.3.3.1). Como todo software libre ofrece además su código fuente. Cabe señalar que tiene un alto nivel de compresión ya que el paquete de instalación tiene algo más que 8 MB y el paquete de código fuente tiene aproximadamente 13 MB.

De manera general se puede llegar a la conclusión que SharpDevelop es una alternativa libre y gratuita para realizar programas para la plataforma .NET. Está totalmente basado en el ambiente de trabajo de

[14] Plataforma de desarrollo de código abierto iniciada en 2001 por Ximian e impulsado por Novell que se basa en .NET y permite crear aplicaciones para varios sistemas operativos.

Visual Studio. Posee las características básicas que se necesitan para caracterizarlo como un moderno entorno de desarrollo de aplicaciones de propósitos múltiples.

MonoDevelop:

Este es un entorno de desarrollo libre y gratuito desarrollado por Novell y los impulsores de Mono con el objetivo de adaptar las funcionalidades y ventajas de SharpDevelop a Gtk#[15]. Es usado por los desarrolladores de Mono en algunas distribuciones de Linux y de Mac OS.

Sus principales características son:

- Soporta los lenguajes C#, Visual Basic .NET, C/C++, Java y *Boo*.
- Manejo de clases.
- Completamiento de código.
- Funciones de navegación por el código.
- Diseñador de interfaz gráfica para GTK#.
- Control de versiones integrada con soporte para Subversion.
- Pruebas unitarias integradas con el componente NUnit.
- Posibilidad de crear proyectos ASP.NET.
- Editor y explorador de bases de datos integrado.
- Integración con la herramienta Monodoc para la generación de documentación sobre las clases.
- Compatible con Visual Studio.
- Control de empaquetamiento de código.
- Herramientas de líneas de comando para compilar y administrar proyectos.
- Agregación de herramientas externas para su enriquecimiento.
- Incluye ayuda de la aplicación para el usuario.

Actualmente se encuentra en ampliación y mejoramiento. Su primera versión es MonoDevelop 1.0 publicada bajo licencia GPL el 14 de marzo de 2008 en el sitio del equipo de desarrollo de MonoDevelop (16).

[15] Protegido por la licencia GNU LGPL es un conjunto de bibliotecas y rutinas para desarrollar interfaces gráficas principalmente para sistemas operativos de la familia Linux aunque se puede utilizar en otros.

1.3.2.1 Software Libre

El software libre es un movimiento bastante desarrollado que promueve algunas políticas en cuanto a la forma de distribuir software. La Fundación para el Software Libre, en Inglés Free Software Foundation (FSF) fue creada en 1985 por Richard Stallman y se dedica a eliminar las restricciones sobre la copia, redistribución, entendimiento y modificación de programas de computadoras.

Según las bases de esta organización, el software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. De forma precisa se refiere a que el usuario una vez adquirido el programa pueda (sin consecuencias legales negativas) usar el programa para cualquier propósito; estudiar su funcionamiento y adaptarlo a sus necesidades propias; distribuir copias del mismo; mejorar el programa y hacer públicas las mejoras de manera que toda la comunidad se beneficie. Para esto es necesario que se brinde junto con el programa ejecutable, el código fuente que dio origen al mismo (17).

Por otra parte, "libre" significa libertad para hacer con el software cualquier cosa una vez adquirido. Este tipo de aplicaciones suelen ser gratuitas pero no es un requisito obligatorio.

Para cumplir con estos principios se han publicado tres tipos de licencia:

- Licencia Pública General GNU (en inglés GNU General Public License, GNU GPL): Es la más usada desde su creación en 1980 y protege la libre distribución, modificación y uso del software. Evita que alguien se apropie de un software declarado libre y restrinja alguna de estas libertades.
- Licencia Pública General Limitada GNU (en inglés GNU Lesser General Public License, GNU LGPL): De manera amplia, esta licencia establece que puede vincularse software no libre con software libre. Su diferencia con GPL radica en que el usuario puede crear software propietario usando software libre siempre y cuando no se demuestre legalmente que el primero se deriva del segundo. Esto ocurre principalmente cuando programas no libres usan librerías libres agregándolas dinámicamente una vez compilados.
- Licencia de Documentación Libre GNU (GNU Free Documentation License, GFDL): Esta licencia asegura que los manuales, libros de texto y otros materiales de referencia protegidos por la misma se encuentren disponibles para copiar, redistribuir y modificar e incluso vender el material, modificado o no, manteniendo los términos de la licencia.

Este tipo de política para con el software consigue que toda la comunidad de usuarios afines trabaje de conjunto sobre los productos y lo mejoren aportando soluciones a nuevos problemas o superando las soluciones existentes a gran velocidad. De igual forma el conocimiento estaría al alcance de todos debido a la posibilidad de entender el funcionamiento de los programas manipulando su código fuente.

Además permite una mejor adaptación de los sistemas a las necesidades propias de cada usuario y una mayor accesibilidad por parte de los países con menos recursos monetarios.

1.3.3 Selección de un entorno de desarrollo

La herramienta a desarrollar es una aplicación típica de escritorio que no posee conexiones a bases de datos. Su función se basa en procesar grandes cantidades de código fuente de manera eficiente.

De la investigación se llegó a la conclusión de que para estos propósitos era factible utilizar SharpDevelop 2.2.1.2648 porque brinda las posibilidades que se requieren para un programa de este tipo. Su descarga desde Internet es gratuita y se instala de forma rápida en la computadora. Es libre y se asemeja bastante al Visual Studio en cuanto al ambiente de trabajo.

De esta forma, una vez analizadas las ventajas y desventajas de todas las posibles combinaciones de lenguaje – plataforma – IDE. Se decidió realizar la implementación mediante el conjunto formado por el lenguaje C#, la plataforma .NET y SharpDevelop como IDE.

1.4 Librerías para el análisis léxico

Una aplicación de análisis de código fuente requiere obligatoriamente de un componente de análisis léxico. Implementar un analizador léxico puede resultar agobiante aun más cuando se trata de un lenguaje con una gramática muy extensa como el C#. Este lenguaje presenta 87 palabras reservadas incluyendo las de la versión 3.0. Cualquier palabra que comience con letra, "_" ó "@" será considerada un identificador válido. Presenta 9 operadores aritméticos, 13 operadores relacionales, 3 operadores de manipulación de bits, 9 operadores de asignación, además de otros 8 operadores. Lógicamente se puede conformar con todo esto un número amplio de sentencias que pueden llegar a ser complejas ya que el C# es muy flexible.

Por lo antes expuesto se llega a la conclusión de que es necesario ahorrar tiempo y esfuerzo que pudiera perderse en la programación si se implementara el analizador léxico. Existen Bibliotecas de Acceso Dinámico, en Inglés Dynamic Linking Library (DLL)[16], también llamadas librerías de clases que encapsulan clases con utilidades comunes a varios programas y son la forma más eficiente de aplicar el principio de la reutilización. Se decidió entonces buscar una de estas librerías que facilite el trabajo de programación.

[16] Biblioteca de enlace dinámico que contiene funciones que pueden ser utilizadas desde los programas, y que son cargadas sólo en el momento en que se necesitan.

Entre tantas encontradas en la Web que se dedican a automatizar este proceso se filtraron aquellas que se ajustaban al lenguaje C# y estaban implementadas bajo la tecnología .NET. A continuación una breve presentación de las mismas.

C# Lex:
Esta herramienta es una versión para el C# sobre .NET del popular constructor dinámico de un lexer JLex. Brinda la posibilidad de especificar una gramática y obtener un analizador léxico para la misma (18). Es una aplicación de tipo consola y posee un manual muy poco útil. Sería incluso necesario especificar la extensa gramática del C# para obtener los resultados esperados.

Grammatica versión 1.4:
Este es un generador automático de *parser* para ser usado en implementaciones Java y C#. Lee la especificación textual de una gramática contenida en un archivo y genera el código fuente para un *parser* correspondiente (19). La gramática debe encontrarse en formato de Forma de Backus y Naur Extendida (EBNF según sus siglas en inglés)[17]. La desventaja principal al utilizar esta opción es tener que especificar la gramática de C# en EBNF.

C# Parser Library:
El Metaspec C# Parser Library está diseñado para proveer el *kernel* para aplicaciones que requieren procesar código fuente escrito en C#. El mismo está disponible en dos versiones, el Compilation Parser que sirve para aplicaciones que requieren convertir código fuente en código intermedio de forma rápida. La otra versión es Transformation Parser que mantiene toda la información referente al código del programa (incluyendo comentarios, espacios y las directivas de pre procesamiento) y es útil para aplicaciones con las funcionalidades de refactorización y formateo entre otros programas que realizan modificaciones sobre el código que es procesado (20). Esta es una buena librería de clases cuyo principal inconveniente radica en que está implementada en C++.

paxScript.NET:
Incluye intérpretes para los lenguajes C# y VB.NET permitiendo la existencia simultánea de ambos lenguajes. Está escrito en C# para .NET. Compila los programas en código binario y no genera un

[17] El Backus-Naur form (BNF) (también conocido como Backus-Naur formalism, Backus normal form o Panini-Backus Form) es una meta sintaxis usada para expresar gramáticas libres de contexto: es decir, una manera formal de describir lenguajes formales.

ensamblado dinámico. Se presenta como un componente (paxscript-net.dll) por lo que puede ser adaptado para su uso en varios entornos. (21)

paxScript.NET es un componente muy útil que resuelve el problema planteado de brindar una interpretación del lenguaje C# para ser usada en los análisis posteriores de todos los elementos que componen cualquier trozo de código fuente. La principal desventaja que se presenta es la ayuda escueta que se brinda junto con la librería de clases. Las explicaciones sobre como funciona cada aspecto de la librería no son suficientes y este elemento entorpece el propósito principal de ahorrar tiempo y esfuerzo en la tarea de implementar el análisis léxico del código que se va a procesar.

CSharpCodeLexer:

Esta librería brinda la posibilidad de utilizar sus clases y funcionalidades para implementar analizadores léxicos y se ajusta fácilmente a cualquier propósito de análisis sobre código C#. La misma se encuentra disponible en la página personal del autor. (22) Aunque no presenta un sistema de ayuda o manual de usuario bien establecido, su uso es sencillo y es muy ajustable a las diferentes necesidades de los programadores. Se brinda la DLL correspondiente al CSharpCodeLexer así como la correspondiente al ngineerLexicalAnalyzer. Esta última permite utilizar la anterior y hacer interesantes acciones con el código C# como, por ejemplo, crear páginas web HTML con el código analizado y presentarlo con todos los detalles de su formato tal como esté en el archivo que lo contenga. Además se brinda el código de dichos componentes para una mayor comprensión de su funcionamiento interno y con la posibilidad de mejorarlos. Su implementación está basada en el estándar ECMA-334. Cumple con un efectivo trabajo de detalles como el soporte de directivas de pre procesamiento.

Finalmente, analizando sus ventajas y desventajas y comparándolas con las de sus similares anteriormente expuestos, se decide utilizar CSharpCodeLexer para resolver el problema del análisis léxico de código C#.

Conclusiones

La programación orientada a objetos ha ido evolucionando y cada vez son más las posibilidades que esta variante brinda en el desarrollo de productos informáticos. El surgimiento del lenguaje C# y de la tecnología .NET han dejado atrás obstáculos que entorpecían el trabajo de los programadores. Al programar en C# para .NET mediante el SharpDevelop se contrastan estos grandes avances que finalmente repercuten en la optimización de tiempo, costo y esfuerzo así como en el incremento de la calidad en el desarrollo de software. La reutilización de componentes previamente programados, es una de las principales tendencias en los procesos ingenieriles para fines informáticos, hoy. Esta

permite ahorrar tiempo, esfuerzo y costos, así como minimizar errores en la programación de elementos que ya existen y están probados. De ahí la importancia de saber integrar elementos como el utilizado en este trabajo para el análisis léxico del código fuente. La idea fundamental consiste en dividir el problema general en pequeños problemas derivados, buscar los elementos existentes que resuelven cada uno de estos problemas y adaptarlos para un mejor resultado.

CAPÍTULO 2: DISEÑO DEL SISTEMA

Introducción

En este capítulo se realizará un análisis crítico y valorativo del diseño entregado por el analista y se explicarán los cambios necesarios que se realizaron para mejorar el proceso de implementación. Se explica el funcionamiento de cada módulo así como los elementos teóricos que se manejan. Finalmente se definen los estándares de codificación a utilizar por ambos programadores.

2.1 Valoración del diseño propuesto por el analista

El objetivo de la implementación es tomar el diseño e implementarlo en archivos de código fuente, librerías de clases y ejecutables. Para ello generalmente se hacen trazas definiendo un archivo de código fuente para cada clase del diseño. Además, se agrupan las clases que tengan mucha interacción entre sí o que respondan a intereses contextuales semejantes en librerías de clases. Las clases controladoras del programa se compilan en un ejecutable y este hace invocaciones a elementos de las librerías en tiempo de ejecución. En la implementación se agregan cosas que no es sencillo planificarlas durante el diseño debido a su complejidad o las particularidades del lenguaje de programación entre otros factores.

El sistema a desarrollar en este trabajo es una típica aplicación de escritorio que no usa conexiones remotas a otras estaciones de trabajo ni a servidores. No contiene bases de datos, no hace uso de servicios externos, no requiere de técnicas avanzadas de seguridad en cuanto al control de acceso y a la integridad de la información.

Por estas razones, el diseño se centra solamente en definir las clases que modelarán el problema. Crear un flujo de trabajo en el cual el usuario se sienta cómodo y obtenga los resultados de manera rápida y sencilla. El tema de las pruebas de Caja Blanca es complejo, sin embargo, el usuario debe inferir, sin tener que hacer mucho uso de la ayuda, las acciones a llevar a cabo para obtener lo que desea durante su trabajo en la aplicación. Para esto, la interfaz debe ser clara, los mensajes y etiquetas no deben ser ambiguos, las pantallas y los escenarios deben estar organizados contextualmente.

El diseño que sirve de punto de partida a este trabajo, organiza las clases en paquetes como se muestra en la Figura 1.

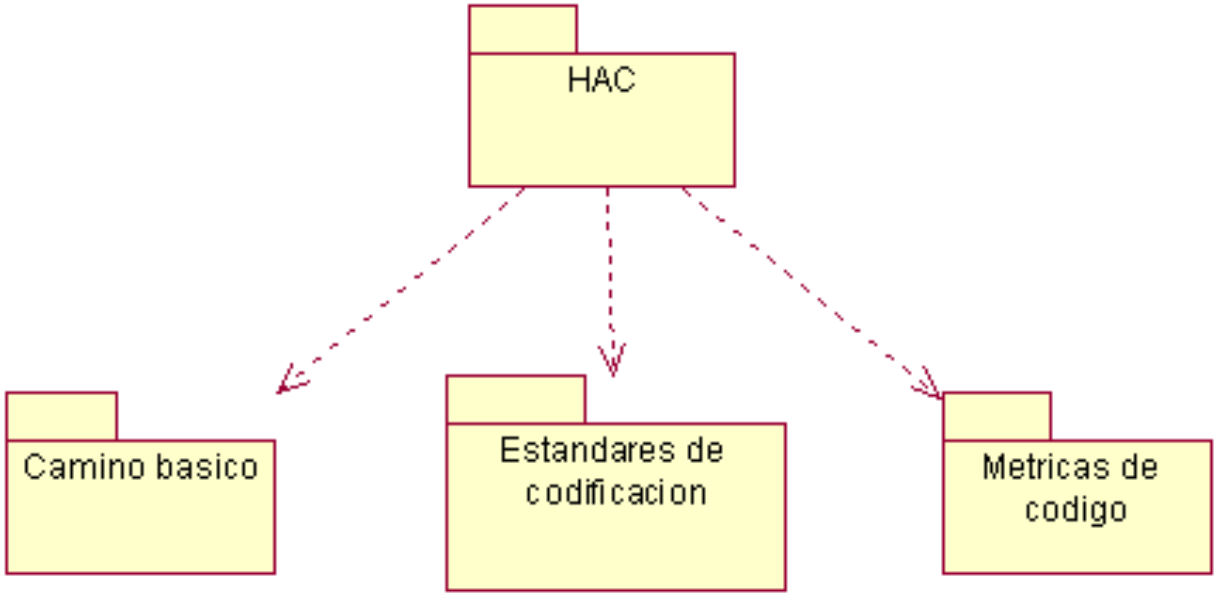


Figura 1 Organización por módulos de HAC.

Esta distribución por módulos es oportuna debido a que la herramienta se concibe para procesar código C# pero en tres direcciones diferentes que no se relacionan entre sí funcionalmente. En las próximas secciones se explicará en que consiste cada módulo que comprende HAC.

2.1.1 Módulo Camino Básico

En este paquete se colocan todas las clases y componentes necesarios para realizar todo lo referente al proceso de aplicación de la Técnica del Camino Básico. Se concibió para que el análisis se realice sobre el código de cada función basada esta idea en que, en realidad, esta técnica se aplica a cualquier segmento de código fuente. En la práctica es útil realizar este análisis sobre los algoritmos más críticos. De esta forma se tiene conocimiento de cuan complejas son las funciones de las clases analizadas.

Esta técnica fue propuesta por McCabe en el año 1976 y todavía es una de las más recomendadas para aplicarle al código fuente. Como resultado se obtiene el grado de complejidad lógica de los algoritmos. Mientras mayor es dicha complejidad, más difícil es de comprender el código y por tanto sería más difícil encontrar errores, hacer modificaciones, practicar la reutilización y trabajar en equipo.

La complejidad ciclomática es el resultado principal del proceso y da una medida cuantitativa para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba derivados garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del código.

Este procedimiento (23) se basa completamente en la teoría de grafos y utiliza el grafo de flujo de control del programa. Del grafo mencionado se extraen los caminos independientes que es cualquier camino del programa que introduce por lo menos una nueva secuencia de sentencias de procesamiento o una nueva condición.

El grafo se compone de nodos y aristas donde cada nodo agrupa un grupo de sentencias secuenciales que se ejecutan en orden desde la primera hasta la última. Las estructuras condicionales y los bucles tienen su forma particular de representarse como se muestra en la Figura 2.

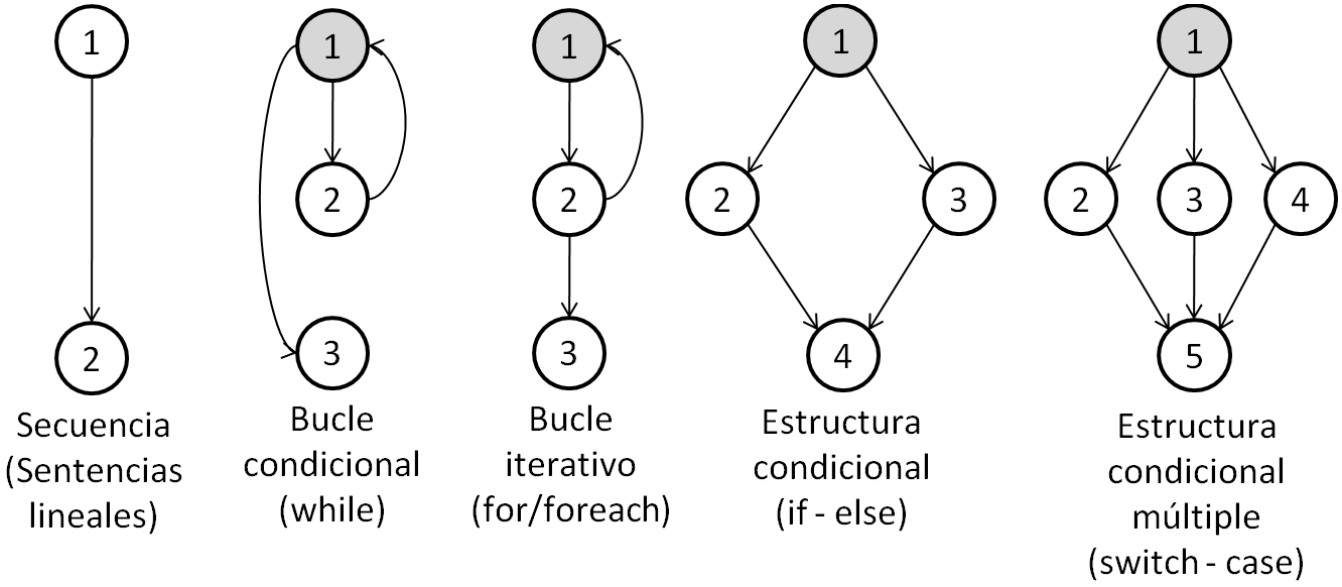


Figura 2 Representación en el grafo de flujo de sentencias lineales, bucles y estructuras condicionales.

Para conformar el grafo se identifican las estructuras mencionadas en la Figura 2 y se agrupan las sentencias correspondientes en los nodos, luego se conectan estos nodos en dependencia de sus relaciones mediante las aristas y por último se integran estos pequeños grafos en el grafo principal.

El método propone tres formas fundamentales para calcular la complejidad ciclomática *V(G)* a partir del grafo obtenido:

- Si se tiene que *R* es la sumatoria de todas las regiones del grafo incluyendo la región exterior, la complejidad ciclomática coincide con el total de regiones.
  $V(G) = R$
- Si se tiene que *A* es la cantidad de aristas y *N* la cantidad de nodos contenidos en el grafo respectivamente:
  $V(G) = A – N + 2$

- Si *P* es la cantidad de nodos predicado presentes en el grafo, teniendo en cuenta que nodo predicado es todo aquel para el cual se cumple que existan dos o más aristas que partan del mismo.
  V(G) = P + 1

El valor *V(G)* representa, en término de las pruebas de Caja Blanca, el número de casos de prueba que deben crearse para probar todas las sentencias del programa. Representa además el número máximo de caminos independientes del programa.

Corresponde entonces encontrar un conjunto básico de caminos independientes a partir de saber la cantidad de caminos que tendrán los conjuntos. Es importante aclarar que se pueden encontrar varios conjuntos básicos de caminos independientes y este hecho está determinado, entre otras cosas, por el tope de ejecución de los ciclos o en otras palabras, por la cantidad de iteraciones posibles y la cantidad de combinaciones que mediante los bucles se logra. Sin embargo, para cada conjunto se cumple que cada camino debe incorporar al menos una arista no visitada en el resto de caminos del conjunto.

El siguiente ejemplo muestra la aplicación de la técnica del camino básico sobre una función escrita en C# que hace declaraciones de variables, asignaciones de valores, presenta un ciclo *for* y dentro del mismo una estructura condicional *if - else* sencilla. Primeramente se asignan los nodos a las sentencias en dependencia de su tipo y luego se conforma el grafo uniendo los nodos como se explicó anteriormente. Se procede a encontrar el valor de las variables necesarias en el grafo y se calcula la complejidad ciclomática de las tres formas posibles. Finalmente se ofrece un conjunto de caminos independientes.

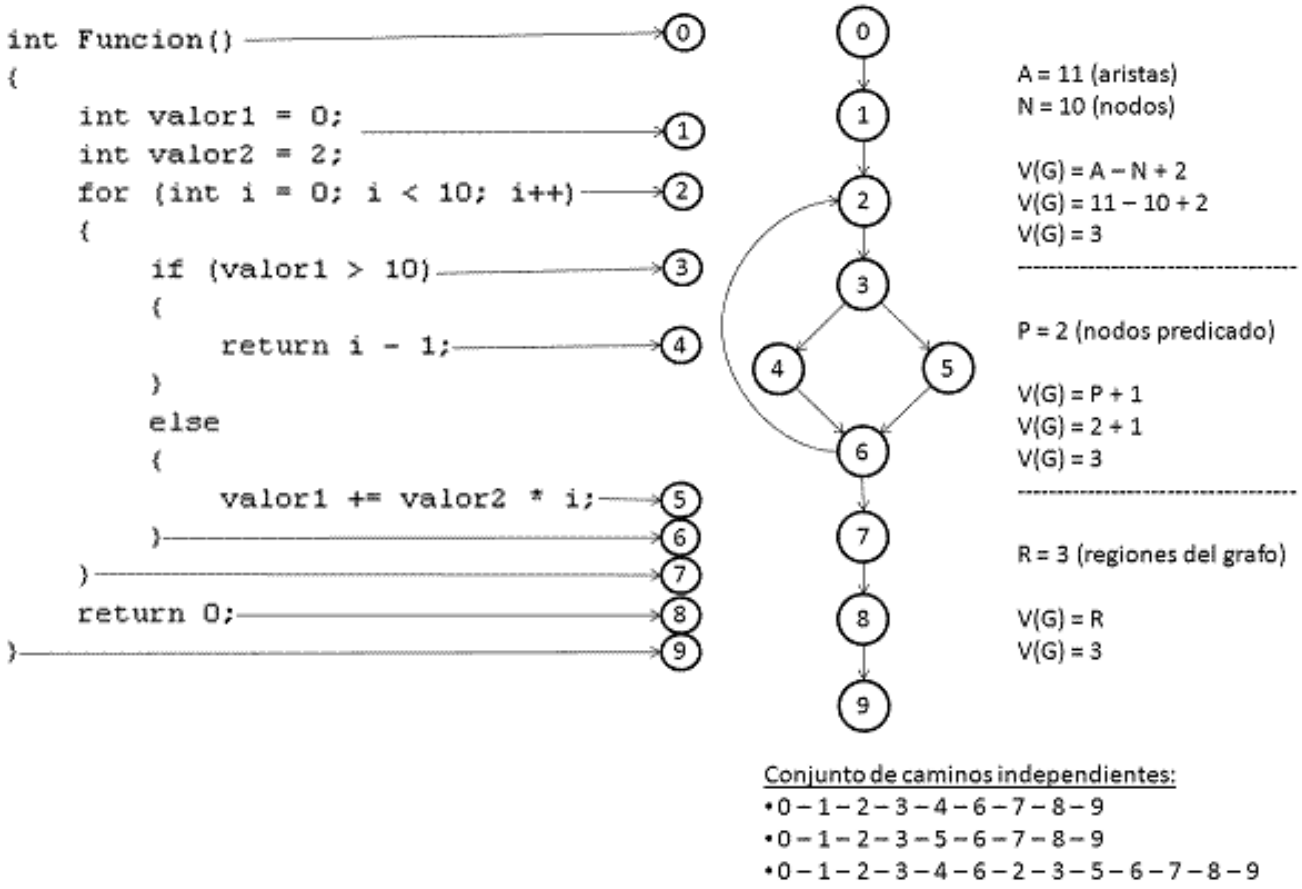


Figura 3 Ejemplo de aplicación de la Técnica del Camino Básico.

Para llevar a cabo este procedimiento en la herramienta se creó la clase *McCabe* en honor al creador de la técnica y que es la encargada de dirigir las operaciones de asignar nodos así como formar y exportar la matriz del grafo. Se apoya en clases como *Grafo* que posteriormente será detallada y se usa para manejar los nodos y sus relaciones. *Nodo* es la clase que representa la identidad que contiene los datos de los nodos. *FabricaNodo* se utiliza para construir un nodo.

Los resultados específicos que se obtienen en este paquete son:

- El número que representa la complejidad ciclomática para cada función.
- Un conjunto básico de caminos independientes. Se propone uno de los conjuntos básicos de caminos independientes teniendo en cuenta que pueden existir varios de estos conjuntos.
- La matriz del grafo de flujo.

2.1.2 Módulo Estándares de Codificación

En este módulo se agrupan todas las clases que se utilizan para la comprobación de estándares definidos en el código. Se diseña además la posibilidad de definir un estándar que pueda ser guardado como un archivo de configuración.

Los estándares se establecen mediante la combinación de valores de varios aspectos. Para conocer más sobre las especificidades de los estándares y sus elementos, ver la sección 2.4.

Identificadores:

Se tratan por separado los identificadores de tres tipos de elementos:

- Clases, Interfaces, Estructuras y Espacios de nombres.
- Variables.
- Funciones.

Todos lo identificadores tienen, sin embargo, los mismos aspectos para su configuración y estos son:

- Comienzo: Especificación del comienzo del identificador que puede ser: con un carácter específico, con mayúscula, con minúscula o libre.
- Separación de palabras internas: Especificación de cómo se separan las palabras internas en identificadores compuestos que puede ser: libre, con un *underscore* (“_”) o ninguna. La herramienta no comprueba el caso de *underscore* porque para esto habría que conocer si la palabra es compuesta.
- Comienzo de palabras internas: Especificación del comienzo de las palabras internas de los identificadores compuestos, que puede ser: con mayúscula, con minúscula o libre. La herramienta no comprueba el caso de mayúsculas porque para esto habría que conocer si la palabra es compuesta o no.
- Estilo: Especificación de un estilo predefinido para el identificador. Se proponen los estilos *lowerCamelCase*, *UperCamelCase* y personalizado. Los estilos no personalizados comprenden los tres aspectos anteriores y se conforman mediante una combinación específica de los mismos. La herramienta no comprueba completamente los estilos no personalizados ya que para esto habría que conocer si la palabra es compuesta o no.

Llaves:

Las llaves tienen varios aspectos a medir:

- Llave de apertura: Especifica la configuración para la colocación de las llaves de apertura que puede ser: al final de la línea cabecera del bloque, solitaria en la siguiente línea, en la siguiente línea seguida del código de la misma o libre.
- Llave de cierre: Especifica la configuración para la colocación de las llaves de cierre que puede ser: al final de la última línea del bloque, solitaria en la siguiente línea, en la siguiente línea seguida del código de la misma o libre.
- Caso de una sola línea: Establece el uso de llaves para estructuras cuyo cuerpo tenga una sola línea y puede ser: usar siempre las llaves, no usar las llaves o libre.

Indentación:

Los aspectos de la indentación son:

- Alineación: Especifica la alineación de las líneas indentadas que puede ser lineal al nivel de la línea cabecera, indentada en espacios o libre.
- Longitud de línea: Especifica la longitud que deben tener las líneas estas pueden ser: hasta que se termine una sentencia, con una sola sentencia, con cantidad de caracteres específica o libre.
- Corte de línea: Establece en qué elemento de la línea se debe realizar el corte, que puede ser: después de una coma, antes de un operador o libre.
- Alineación de la línea cortada: Especifica la alineación del segmento posterior de la línea cortada sus parámetros pudieran ser: lineal a la definición, con espacios, sin espacios o libre.

Líneas y espacios en blanco:

Los aspectos que corresponden a las líneas y los espacios en blanco son los que siguen:

Para las líneas en blanco:

- Entre funciones: Establece si se usará una línea en blanco entre la declaración de las funciones.
- Entre declaraciones de variables e implementaciones: Especifica si se usará una línea en blanco entre la declaración de variables y el resto de la implementación dentro del cuerpo de las funciones.
- Antes de estructuras de control: Especifica si se usará una línea en blanco antes de la implementación de cada estructura de control.
- Antes de comentarios: Especifica si se usará una línea en blanco antes de cada comentario.

Para los espacios en blanco:

- Entre palabras reservadas y paréntesis: Especifica si entre las palabras reservadas y el paréntesis de apertura que le sigue se colocará un espacio en blanco.
- Entre identificador de función y paréntesis: Especifica si entre el identificador de las funciones y el paréntesis de apertura que le sigue se colocará un espacio en blanco.
- Entre primer paréntesis y primer argumento: Especifica si entre el paréntesis de apertura y el primer argumento se colocará un espacio en blanco.
- Después de las comas: Especifica si se colocará un espacio en blanco después de las comas en la lista de argumentos.
- Después del último argumento y antes del último paréntesis: Especifica si entre el último argumento y el paréntesis de cierre que le sigue se colocará un espacio en blanco.
- Entre paréntesis consecutivos: Establece si entre los paréntesis consecutivos se colocará un espacio en blanco.
- Antes y después de operadores binarios: Establece si antes y después de los operadores binarios se colocará un espacio en blanco.
- Antes y después de operadores unarios: Establece si antes y después de los operadores unarios se colocará un espacio en blanco.
- Después de los *punto-y-coma* en un *for*: Especifica si se colocará un espacio en blanco luego de cada punto-y-coma en la declaración de los ciclos *for*.

Comentarios:

Los estándares para la escritura de comentarios en el código se basan en lo siguientes aspectos:

- Tipo de comentario: Establece el tipo de comentario que se debe usar, puede ser: libre, comentario de bloque o comentario de una sola línea.
- Alineación del comentario: Establece la posición donde se ubicará el comentario, los cuales pueden situarse: al final de la línea, encima de la línea e indentado al nivel de la misma, encima de la línea e indentado en espacios, encima de la línea e indentado sin espacios y libre. Para el caso del tipo de comentario de bloque no se comprueba el caso de alineación al final de la línea.
- Elementos a comentar: Especifica qué elementos se deben comentar. Se proponen las clases, interfaces, estructuras, espacios de nombre, variables y funciones.

Para modelar este Módulo Estándares de Codificación se definen las clases para cada aspecto de la configuración:

- ConfiguracionEstandares: Contenedora de todas las demás.
- ConfiguracionClases.
- ConfiguracionFunciones.
- ConfiguracionVariables.
- ConfiguracionLlaves.
- ConfiguracionIndentacion.
- ConfiguracionLineasYEspaciosBlanco.
- ConfiguracionComentarios.

Para diferenciar los incumplimientos en los que incurra el código al comprobar estándares se definen las clases:

- IncumplimientoIdentificador.
- IncumplimientoComentarios.
- IncumplimientoIndentacion.
- IncumplimientoLlaves.
- IncumplimientoLineasEspaciosBlanco.
- Incumplimiento (De esta heredan todas las anteriores).

La clase *EstandaresCodificacion* es la controladora en el módulo e implementa los métodos para comprobar cada aspecto de la configuración de los estándares y obtener los correspondientes incumplimientos.

2.1.3 Módulo Parámetros del Código

En este módulo se empaquetan todas las clases que se utilizan en la determinación de los parámetros de calidad del código fuente. Los parámetros a comprobar se seleccionan de algunas de las métricas que se han definido para comprobar la calidad en el código. A continuación una breve explicación de los parámetros seleccionados de las métricas.

Chidamber & Kemerer (24):
Este es un grupo de métricas del diseño orientado a objetos. Se seleccionaron los siguientes parámetros:

- *WMC*: Métodos Ponderados por Clase que es la sumatoria de las complejidades de cada función para una clase. En este caso, la complejidad es la complejidad ciclomática de McCabe.

- *DIT*: De la jerarquía de clases, Camino más Largo desde la Raíz hasta una Clase que es el nivel de la clase más profunda de la jerarquía.
- *NOC:* Número de Hijos de una Clase que es la cantidad de hijos para cada clase.
- *RFC*: Respuesta Máxima de una Clase que es el número máximo de métodos que se pueden ejecutar al invocar uno cualquiera en una clase.

Halstead:

Las métricas de Halstead (25) son un conjunto de medidas primitivas que determinan el tamaño del software asumiendo que el programa está compuesto por un conjunto de elementos que se clasifican en *operadores* u *operandos*.

Propone cuatro parámetros básicos:

- *n1*: Es la cantidad de operadores diferentes en el código. Operadores son todos los elementos que realizan acciones sobre los operandos y los operandos son los elementos que contienen información en el programa.
- *N1*: Es el número total de ocurrencias de operadores en el programa.
- *n2*: Es la cantidad de operandos diferentes.
- *N2*: Es la cantidad de ocurrencias de operandos.

A partir de estos cuatro parámetros básicos, mediante operaciones matemáticas se obtiene otros valores:

- *N*: Se define como el tamaño o longitud en palabras de un programa que representa la cantidad total de operadores y operandos que conforman el programa. Se define como:
  $N = N1 + N2.$
- *n*: La suma de operadores y operandos representa el vocabulario de un programa y se define como:
  $n = n1 + n2$
- *L*: Es la longitud estimada de un programa se define como:
  $L = N1 * \ln(n1) + N2 * \ln(n2)$
- *V*: Es el volumen de un programa que representa el volumen en bits necesarios para el programa y depende del lenguaje de programación. Se define como:
  $V = N * \ln(n)$
- *E*: Es el esfuerzo necesario que debe realizar un programador, teniendo en cuenta el número de discriminaciones mentales elementales, para implementar un programa. Se define como:
  $E = n1 * N2 * N * \ln(n) / (2 * n2)$

Líneas de código:

La cantidad de líneas de código es el parámetro más usado para interpretar el tamaño de un programa. No existe una definición exacta de qué se considera realmente una línea de código pudiendo excluir o incluir las líneas de comentario, las líneas declarativas y las líneas en blanco. En este trabajo se toman como líneas de código todas las líneas del programa incluyendo los casos antes mencionados.

- *L.Dec*: Representa las líneas declarativas que son aquellas que no ejecutan acciones, se usan solo para declarar elementos y cabeceras.
- *L.Pro*: Representan las líneas procedurales que son aquellas que no son declarativas ni son Líneas en Blanco ni Líneas de Comentarios.
- *L.Blac*: Representa las líneas en blanco que son aquellas que no tienen ningún elemento.
- *L.Com*: Representa las líneas de comentarios que son aquellas que solo tienen uno o varios elementos de comentarios.
- *C.Lin*: Es la suma de todos los parámetros anteriores y representa la cantidad total de líneas de código del programa. Esta forma de determinar las líneas de código da la posibilidad de hacer modificaciones en el futuro incluyendo o excluyendo alguna de las variantes.
- *C.Com*: Es la cantidad de ocurrencias de comentarios independientemente de las líneas en que aparezcan. Es un parámetro útil para conocer el grado de comentarios del código.
- *D.Com:* Es la densidad de comentarios o la proporción de comentarios por líneas de código que da la medida de cuan comentado está el código. (26)

Li – Henry:

Las métricas de Li- Henry (27) interpretan el tamaño de un programa de dos maneras diferentes a las anteriores expuestas y diferentes entre sí.

- *Tamaño1*: Es la cantidad total de *punto-y-coma* (“;”) en una clase.
- *Tamaño2*: es la cantidad de atributos locales y métodos en una clase.
- *NML*: Es el Número de Métodos Localmente Definidos y Heredados en una clase, es la cantidad de métodos de una clase incluyendo los definidos localmente y los heredados en los casos que las clases sean hijas de otras.

Las clases que modelan estos parámetros son:

- Chidamber_Kemerer.
- Halstead_Loc.

- JerarquiaCodigo.
- LiHenry.
- MEstaricas.
- MetricasClase.
- Operando.
- Operadores.
- Arbol.
- ArbolClases

Para generar los reportes de los parámetros se utilizan:

- Archivo.
- RClase.
- ReporteClase.
- ReportePorArchivo.
- Resumen.
- RFuncion.

2.1.4 Interfaces visuales

La aplicación no necesita un engranaje complejo de interfaces visuales. En general, el diseño visual se resuelve con una pantalla central y tres pantallas más, uno por cada módulo respectivamente. El módulo de los estándares agrega una pantalla para la configuración de estándares y tendría otra principal para la comprobación de los mismos. El aspecto de estas pantallas se presenta en las figuras siguientes.

La Figura 4 representa la pantalla principal de HAC desde donde el usuario puede realizar cualquier acción relacionada con las tres áreas principales de uso de la herramienta. Esta ventana brinda la posibilidad de abrir dentro de ella varias a la vez de cualquiera de los tres módulos específicos y realizar acciones paralelas en ellos. Ofrece herramientas de trabajo con múltiples ventanas. Brinda la posibilidad además de acceder a la ayuda de la aplicación.

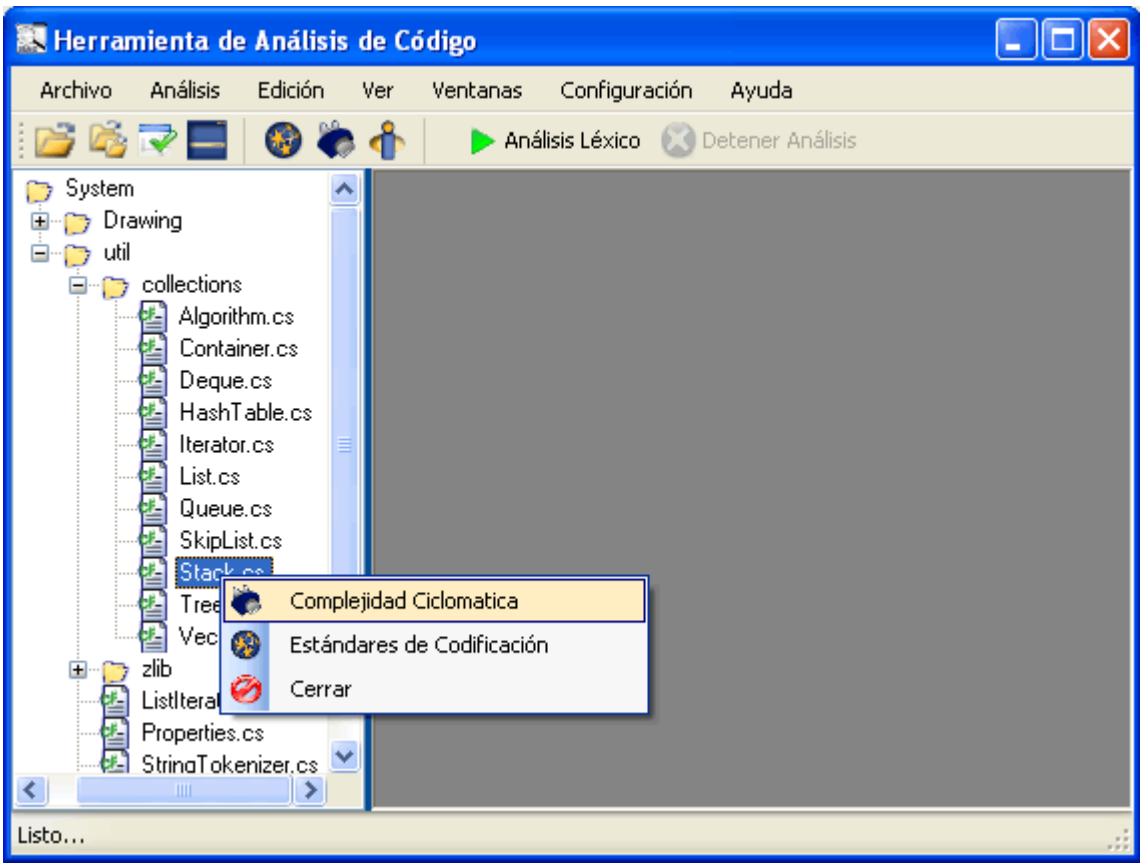


Figura 4 Pantalla del Escenario Principal de HAC.

La pantalla de la Figura 5 muestra cómo el usuario puede aplicar la técnica del camino básico a las funciones contenidas en las clases seleccionadas para procesar. Estas clases se organizan por espacios de nombres en forma de árbol. Una vez aplicada la técnica, se muestra el valor correspondiente a la complejidad ciclomática al lado de cada función. El usuario tiene la oportunidad de guardar la matriz del grafo de flujo correspondiente a cada función además de visualizar un conjunto básico de caminos independientes correspondiente a cada matriz. Además, el usuario puede importar una matriz existente y aplicar los mismos procedimientos que a los archivos.

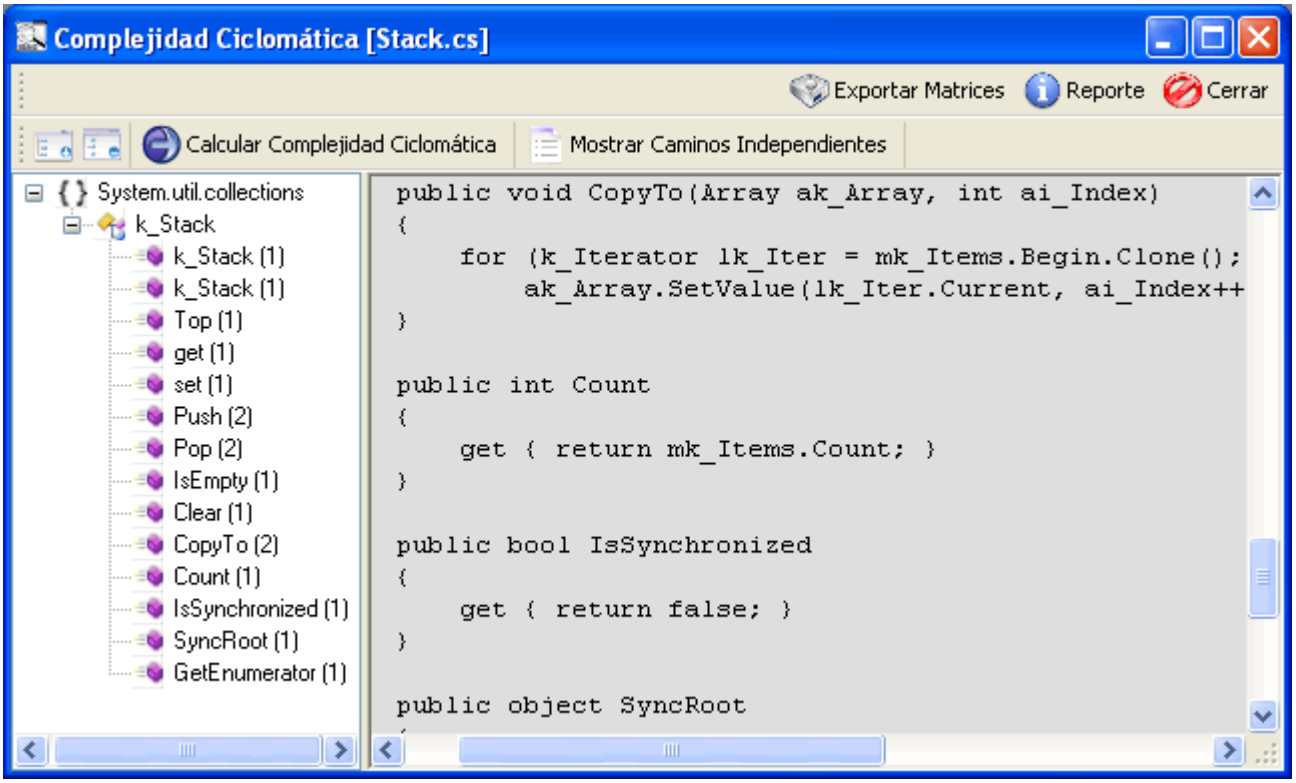


Figura 5 Pantalla principal del Módulo Camino Básico.

La pantalla de la Figura 6 da la idea de cómo el usuario puede seleccionar una combinación de los aspectos correspondientes a los estándares de codificación. Luego estos estándares pueden ser guardados en la máquina para su posterior uso en la comprobación de estándares. Incluso pueden abrirse estándares y hacer modificaciones sobre los mismos guardando nuevamente los cambios realizados.

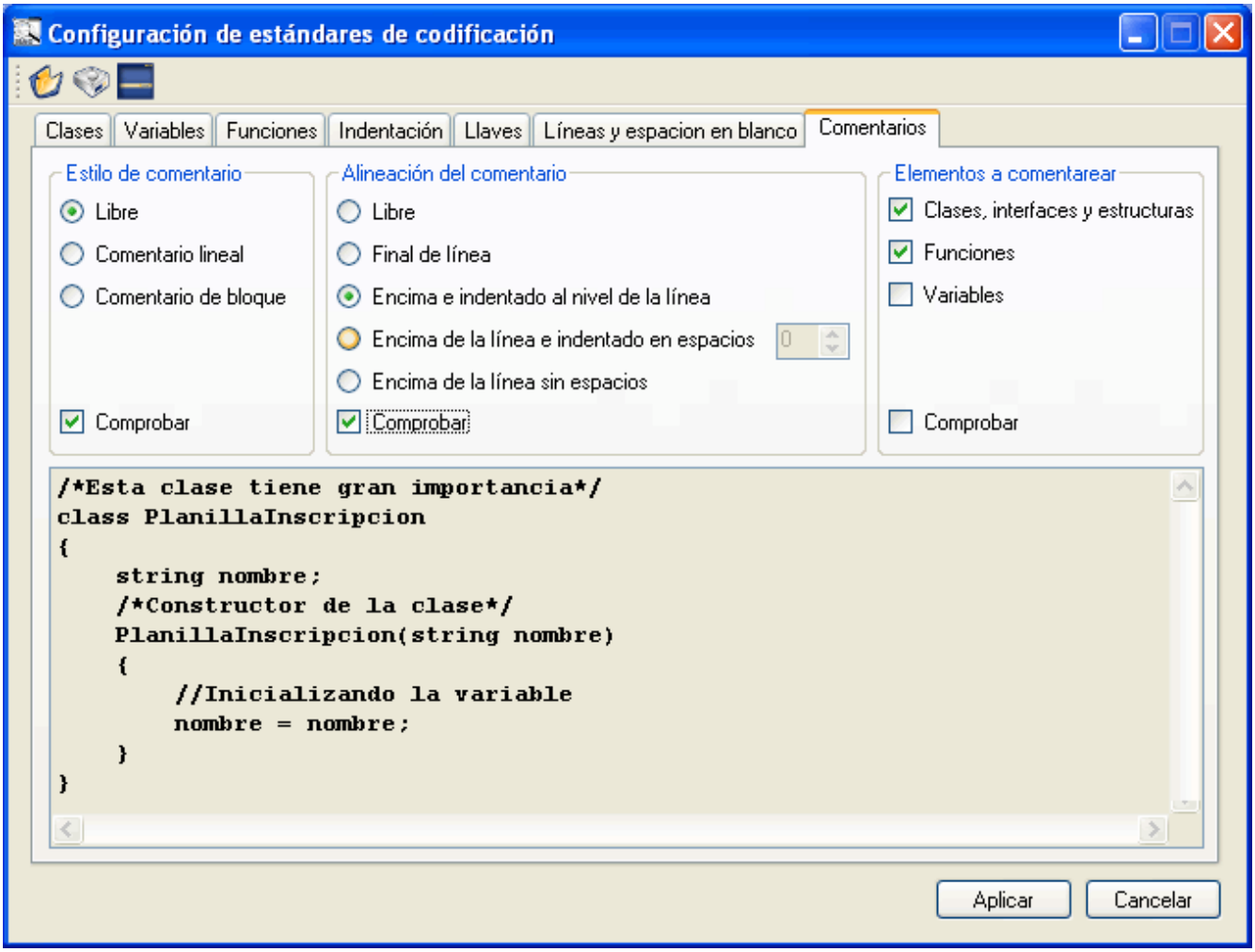


Figura 6 Pantalla de la Configuración de Estándares del Módulo Estándares de Codificación.

La Figura 7 muestra como el usuario puede analizar el código que se encuentra en la parte superior y mostrar el resultado de ese análisis en la parte inferior. El resultado del análisis serán los incumplimientos detectados en el código durante la comprobación de estándares. Estos incumplimientos se muestran organizados en forma de árbol en dependencia de su tipo.

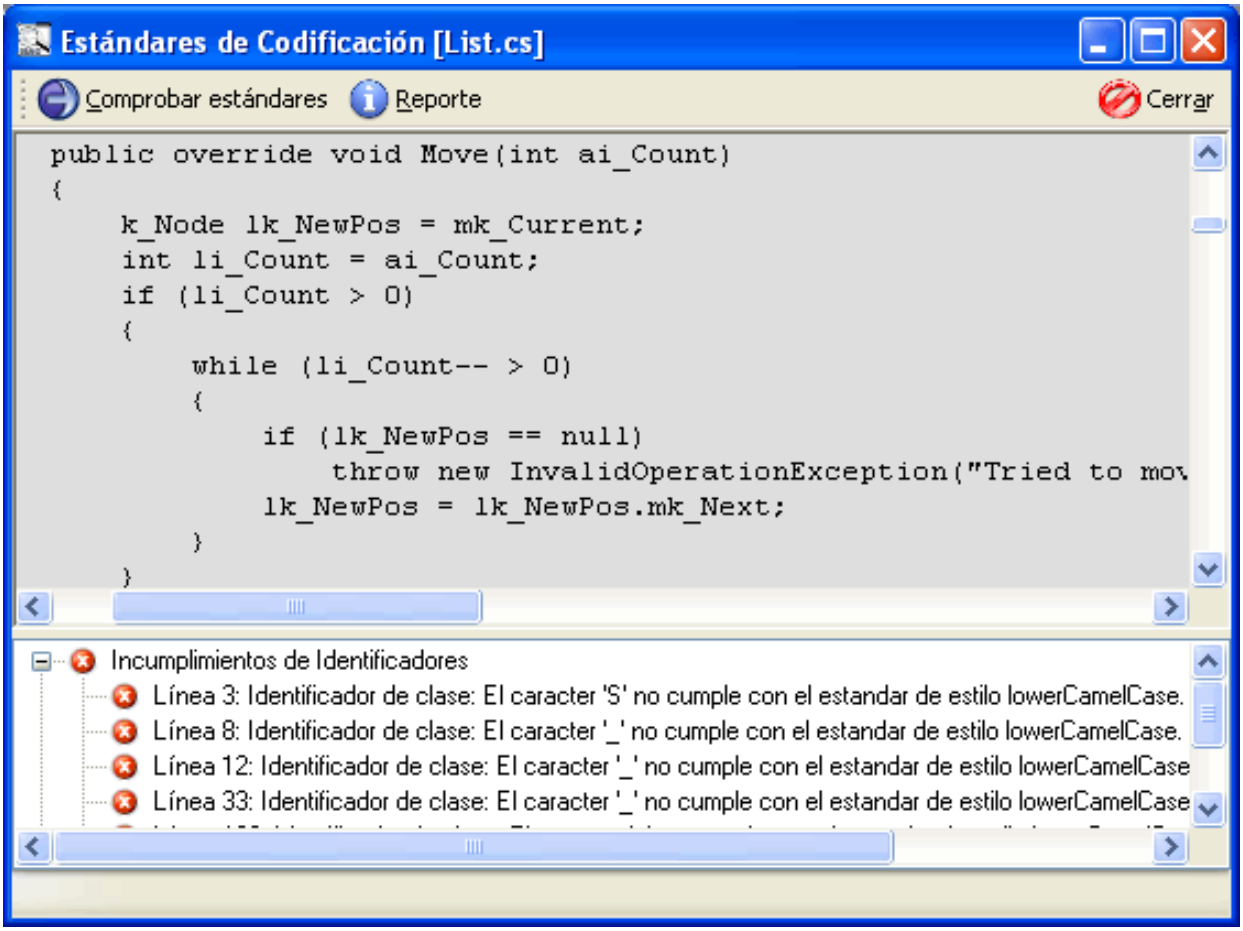
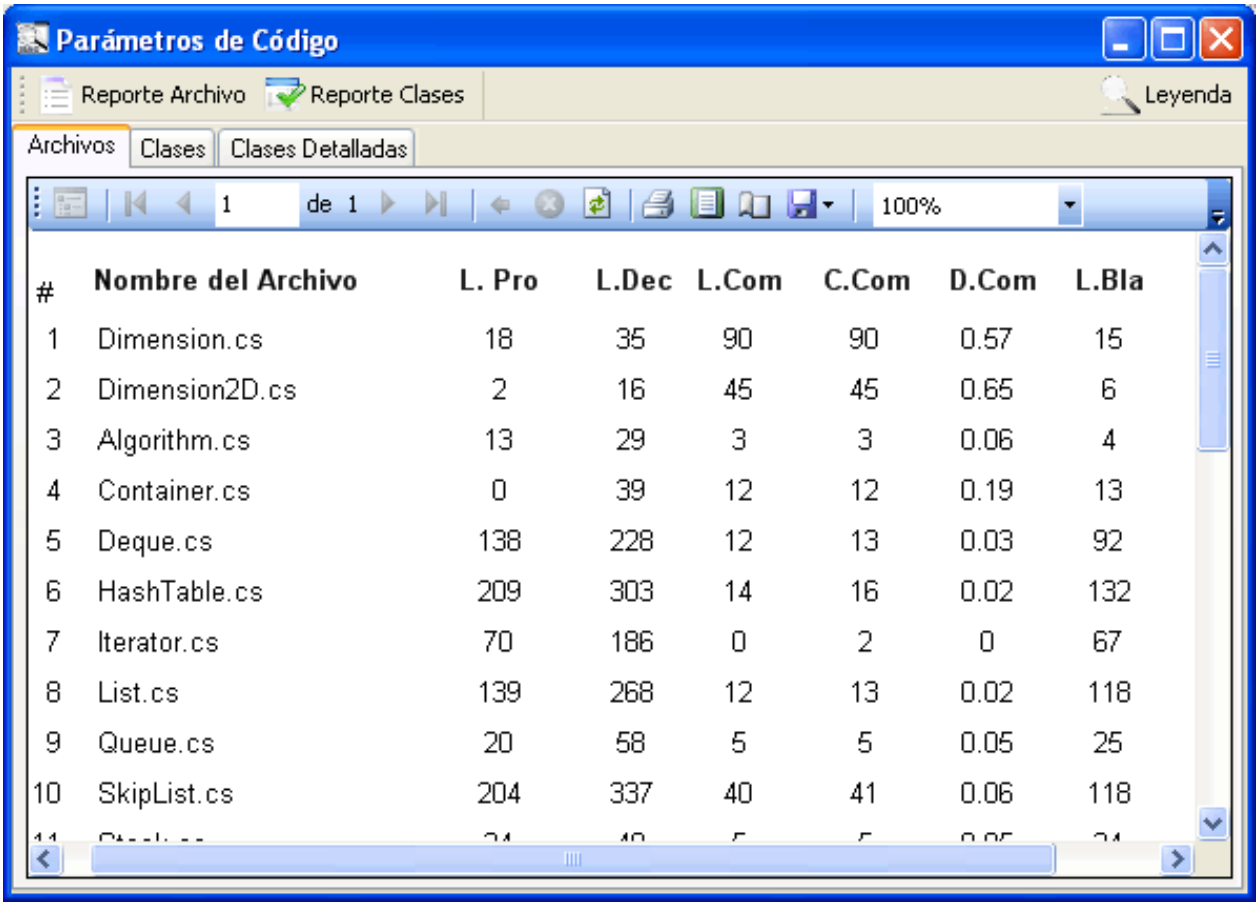


Figura 7 Pantalla principal del Módulo Estándares de Codificación.

La Figura 8 muestra como el usuario puede procesar varios archivos para determinar los parámetros de calidad del código fuente. Esto se puede hacer para todas las clases que se encuentren en los archivos o para todos los archivos seleccionados. Los parámetros se mostrarán como un reporte con todas las posibilidades que brindan los reportes generados en la plataforma .NET. Estos pueden ser impresos y para ello se pueden ajustar al formato de la página deseado en cuanto a tipo de hoja, tamaño del texto, márgenes y bordes.



#	Nombre del Archivo	L. Pro	L.Dec	L.Com	C.Com	D.Com	L.Bla
1	Dimension.cs	18	35	90	90	0.57	15
2	Dimension2D.cs	2	16	45	45	0.65	6
3	Algorithm.cs	13	29	3	3	0.06	4
4	Container.cs	0	39	12	12	0.19	13
5	Deque.cs	138	228	12	13	0.03	92
6	HashTable.cs	209	303	14	16	0.02	132
7	Iterator.cs	70	186	0	2	0	67
8	List.cs	139	268	12	13	0.02	118
9	Queue.cs	20	58	5	5	0.05	25
10	SkipList.cs	204	337	40	41	0.06	118

Figura 8 Pantalla principal del Módulo Parámetros de Calidad del Código Fuente.

La Figura 9 muestra un aspecto general de la ayuda de la aplicación. Este componente consiste en un sitio web estático insertado en HAC aprovechando las posibilidades que brindan estas aplicaciones. Este sitio presenta una interfaz sencilla y amigable. Durante la navegación el usuario puede obtener la información que busca de forma rápida. Puede usarse de conjunto con otras secciones de la herramienta.

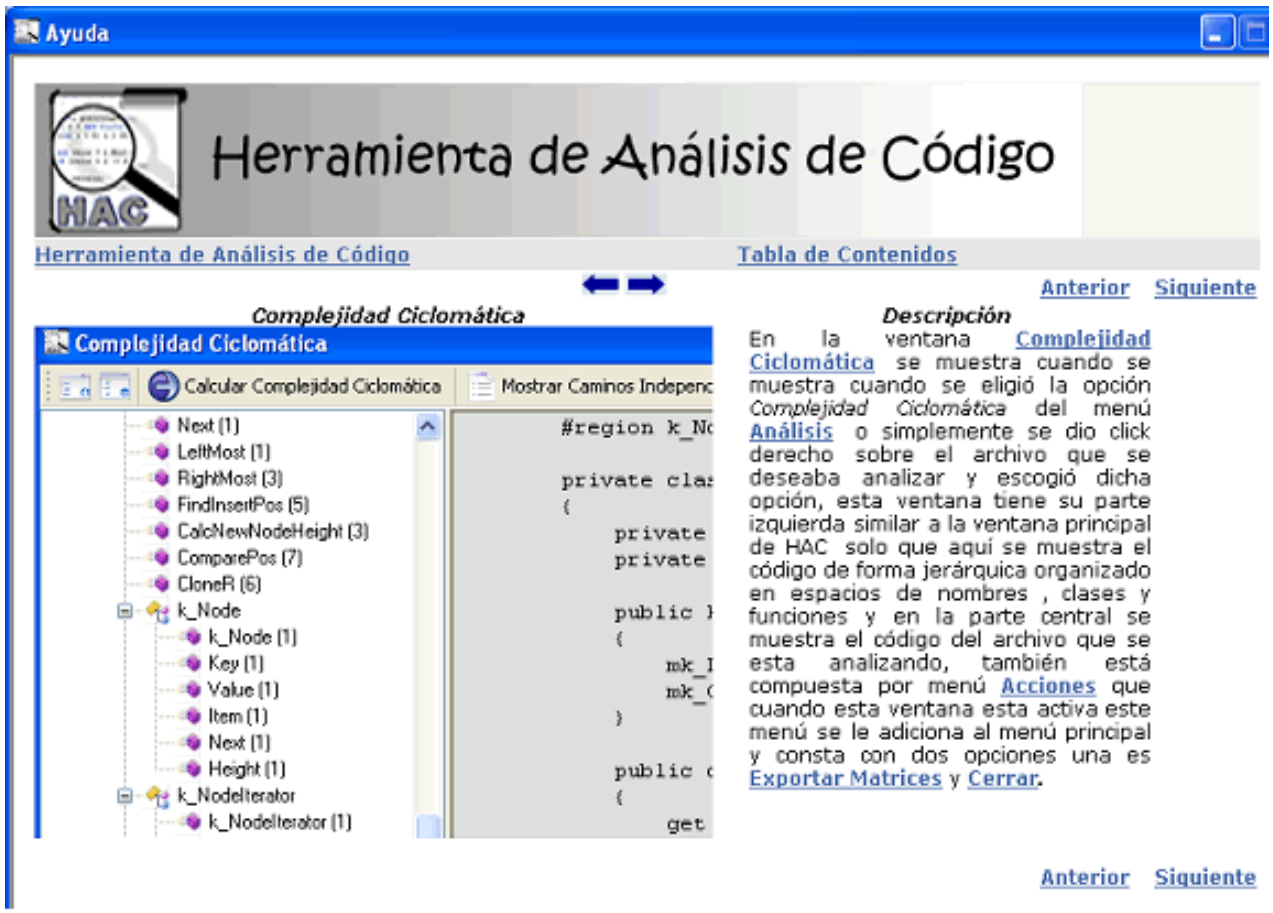


Figura 9 Pantalla de la ayuda de HAC.

La Figura 10 muestra la sección de configuración general de la herramienta. Desde aquí se escoge la configuración de estándar a utilizar durante la comprobación de estándares. Se agregan nuevos analizadores para nuevos lenguajes de programación con la posibilidad de establecer un filtro para los archivos que soporta dicho lenguaje. Se puede guardar la configuración general como un archivo en la computadora para ser aplicado cada vez que se ejecute HAC. Si se agrega un nuevo analizador, sería posible utilizarlo solo en el módulo Camino Básico ya que la comprobación de estándares y la determinación de parámetros dependen en gran medida de las especificidades sintácticas del lenguaje y es imposible hacer un analizador estándar para todos los lenguajes.

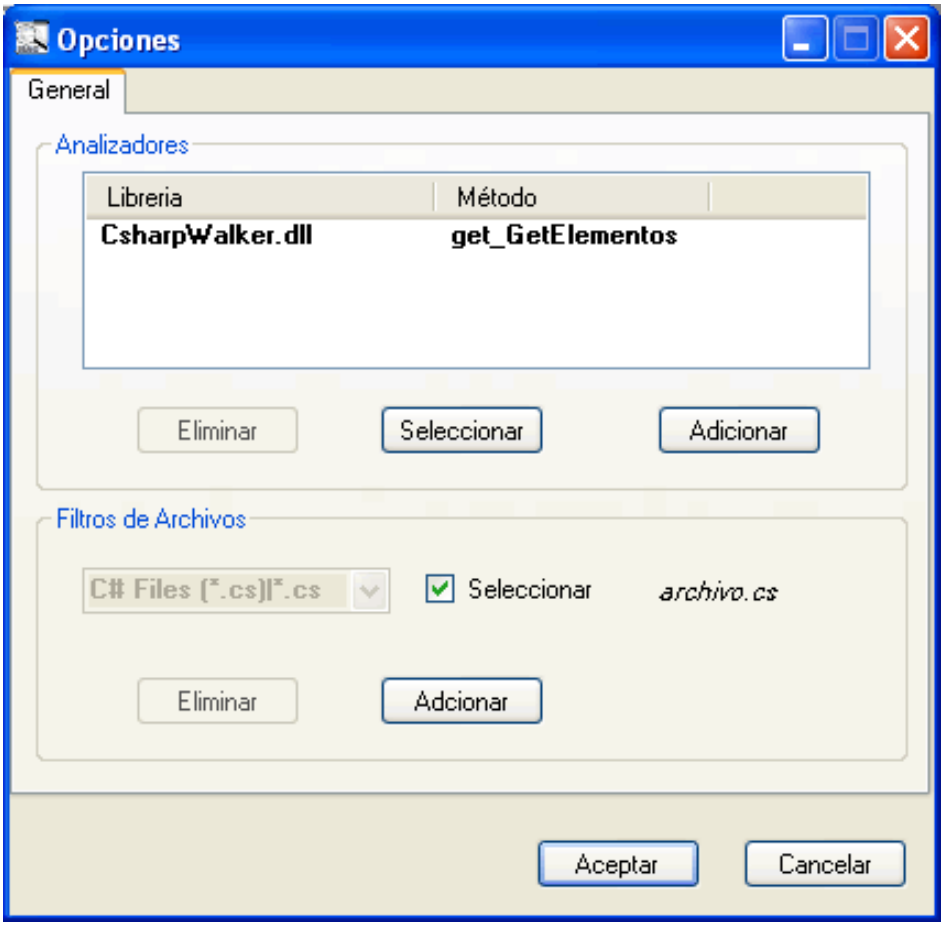


Figura 10 Pantalla Configuración de analizadores de lenguajes y filtros de archivos.

2.2 Reutilización de implementaciones, componentes y módulos

La reutilización es un fenómeno importante y decisivo dentro del desarrollo de la informática, multiplicando las capacidades de programación. Se plantea que cada segmento de programa debe pensarse como si fuera a ser usado por muchas personas aunque no se encuentre un motivo en ese momento para hacerlo. La gran mayoría de los sistemas utilizan bibliotecas de clases que tienen casi todo lo básico implementado y solo habría que invocar esas funcionalidades.

Por ejemplo, durante la programación de un editor de texto se implementa una clase que tiene un determinado número de funciones para todos los procesos que se llevan a cabo durante la impresión de documentos. Al cabo de dos años, el mismo programador decide realizar un programa cliente de correo electrónico que ocasionalmente puede imprimir mensajes de correo. Es entonces cuando entra a jugar su papel la reutilización y se inserta la clase de impresión que se había utilizado para el editor de texto.

A veces, es imprescindible realizar ajustes a los componentes para adaptarlos al medio donde se utilizarán. Otras veces el componente es compatible tal como fue implementado la primera vez. En cualquiera de los dos casos se estaría haciendo uso de la reutilización.

2.2.1 Utilización de la librería CSharpCodeLexer

Como se escribió en el Capítulo 1 de este trabajo, para cumplir con el análisis léxico de código C# en HAC, se utilizó una librería llamada *CSharpCodeLexer* desarrollada por Omer Van Kloeten (28). La misma permite obtener las palabras o *token* que se encuentren en el código basándose en el estándar "ECMA-334 Especificación para el Lenguaje C#" (29).

De este componente, la clase fundamental que se utiliza es *LexicalWalkerBase* pues es la clase que implementa de forma general el paso por cada palabra del código convirtiéndola y almacenándola como *token* y da la posibilidad que pueda heredarse de ella para decidir que hacer con estos elementos.

Se conformó el analizador del lenguaje C# para ser utilizado por HAC. Se empaquetó en la librería *CsharpWalker* que hace uso de la librería *CSharpCodeLexer* e incluye dos clases que se describirán a continuación. La clase *InterfazCsharp* es la mediadora entre la librería utilizada para el análisis léxico y el núcleo de HAC. A continuación se relacionan sus principales métodos.

Tabla 1 Métodos más importantes de la clase InterfazCsharp.

Método	Parámetros	Descripción
InterfazCsharp		Crea una instancia de InterfazCsharp
InterfazCsharp	string filename	Crea una instancia de InterfazCsharp con los parámetros especificados
Iniciar	string filename	Método público que dirige el proceso de análisis que termina guardando en el atributo elementos el ArrayList de elementos o token extraídos del código
Analizar	string code	Invoca la funcionalidad de extraer la matriz de elementos de la librería e invoca la extracción de elementos de la clase MyWalker
ReadCode	string filename	Lee el texto correspondiente al código desde un archivo

CurrentFile		Obtiene o establece el nombre del archivo donde se encuentra el código a analizar
GetElementos		Obtiene la matriz de elementos extraídos del código

La clase *MyWalker* se declaró hija de la clase *LexicalWalkerBase* de la librería utilizada que es la encargada de extraer los elementos durante el recorrido por el código. A continuación sus métodos más importantes.

Tabla 2 Métodos más importantes de la clase MyWalker.

Método	Parámetros	Descripción
MyWalker		Crea una instancia de MyWalker
MyWalk	ILExicalElement element	Invoca el método Walk de la clase padre que comienza el recorrido por el código identificando los elementos
BeginWalkStringValue	StringValue element	Identifica un literal de cadena de caracteres
BeginWalkOperatorOrPunctuator	OperatorOrPunctuator element	Comienza la identificación de un operador o signo de puntuación
EndWalkOperatorOrPunctuator	OperatorOrPunctuator element	Culmina la identificación de un operador o signo de puntuación
BeginWalkNewLine	NewLine element	Comienza la identificación de un literal especial de nueva línea
EndWalkNewLine	NewLine element	Culmina la identificación de un literal especial de nueva línea
BeginWalkIdentifier	Identifier element	Identifica un identificador
BeginWalkAvailableIdentifier	AvailableIdentifier element	Identifica un identificador declarado de tipo nuevo
BeginWalkKeyword	Keyword element	Identifica una palabra reservada
BeginWalkWhiteSpace	WhiteSpace element	Identifica un espacio en blanco
BeginWalkPPAndExpression	PPAndExpression element	Identifica una expresión de pre procesamiento
BeginWalkCharacterLiteral	CharacterLiteral element	Identifica un literal de carácter

BeginWalkSingleLineComment	SingleLineComment element	Identifica un comentario de una sola línea
BeginWalkDelimitedComment	DelimitedComment element	Identifica un comentario de bloque o de varias líneas

De esta forma se hace la utilización de una librería existente y compartida para extraer los diferentes elementos de un fragmento de código escrito en C# y se basa en el estándar para ese lenguaje. Esta librería es uno de los muchos componentes que se han desarrollado para procesar código C# luego de ser estandarizado el lenguaje y la plataforma .NET. Cada vez se incrementa la cantidad de editores de código, compiladores y herramientas de prueba sobre el mencionado lenguaje tanto para entornos Windows como Linux.

2.3 Estrategias de integración

La informática en su desarrollo se ha ampliado para abarcar todas las esferas de la sociedad. Actualmente existen programas para casi todos los propósitos donde se necesite procesar información. Es por ello que para resolver un mismo problema se pueden encontrar muchos programas. Ejemplo de esto es la cantidad de procesadores de texto disponibles.

Además, se ha hecho necesario que programas que trabajen con datos similares o se usen en las mismas áreas se integren entre sí intercambiando datos mediante diferentes protocolos de comunicación. Por ejemplo, las aplicaciones web son programas que necesitan de un navegador para ser ejecutados en las estaciones clientes.

Esto ha facilitado el trabajo de los usuarios y por eso cada vez se exige más que los sistemas sean capaces de integrarse con sus semejantes.

Se ha concebido que la herramienta de este trabajo sea capaz de integrarse o intercambiar información con otros sistemas para ampliar sus utilidades. ¿Cómo se logra la integración? Esta interrogante encuentra su respuesta en las secciones siguientes.

2.3.1 Ejecución de programas con parámetros

Una de las capacidades de los programas que son útiles en la integración de sistemas, es la ejecución recibiendo parámetros. Para esto es necesario declarar la función *Main* de las aplicaciones de forma que contenga argumentos que serán tomados para realizar acciones automáticas durante la ejecución.

La función *Main* es lo primero que se ejecuta en cualquier programa y a partir de ahí se invocan las instancias y sus métodos necesarios para resolver los problemas. Generalmente esta función no

implementa nada, solo llama a las primeras instancias centrales que se encargan de todo lo demás y espera que terminen su ejecución para entonces terminar la ejecución de la función *Main* y de esa forma culmina el programa. Normalmente, en aplicaciones de escritorio para entornos de Windows con interfaces visuales de formularios, se invoca desde el *Main* al constructor de la clase de la ventana principal de la aplicación y es este quien realiza entonces todas las demás acciones hasta que es cerrado devolviendo el control al *Main*.

La función puede o no retornar valores. Es usual, por convención, retornar un entero que si es cero representa una operación exitosa y si es distinto de cero significa que existieron errores en la ejecución (30), como toda función admite parámetros que en el caso de C# puede ser una matriz de cadenas de caracteres o ninguno.

Si se quiere, por ejemplo, desde un editor de textos invocar a un programa que se usa para imprimir documentos, se pudiera pasar en una posición de la matriz, una cadena que contenga el texto a imprimir o una cadena que representa la dirección en el directorio que tiene el archivo del documento a imprimir. Para esto es necesario que el programa de impresión declare que en su función *Main* reciba estos parámetros. En el ejemplo de la Figura 11 se presenta la declaración de la clase *ProgramaImpresion* que contiene la función *Main* del programa que recibe como parámetros la dirección del archivo del documento a imprimir. Es entonces cuando se instancia a la clase Impresora y se le entrega el control del flujo del programa ordenándole la impresión del documento. Cuando termina la impresión, se le devuelve el control al *Main* y este culmina la ejecución general.

```
class ProgramaImpresion
{
    //Función Main recibiendo parámetros
    static void Main(string[] args)
    {
        //Comprobando existencia de parámetros
        if (args[0] != null && args[0] != "")
        {
            //Instanciando impresora para realizar acción con parámetros
            new Impresora().ImprimirDocumento(args[0]);
        }
    }
}
public class Impresora
{
    public Impresora() { }
    public void ImprimirDocumento(string nombreArchivoDocumento) { }
}
```

Figura 11 Programa para imprimir con función Main recibiendo la dirección del documento como parámetros.

Esta posibilidad es útil y muchos sistemas la utilizan. Además es sencillo de implementar. En la próxima sección se explica como se utiliza esta opción para la integración de HAC con otros sistemas.

2.3.2 Integración de HAC con editores de código fuente

Para lograr que las aplicaciones se comuniquen entre ellas automáticamente se usan los parámetros de ejecución como se explicó en la sección anterior.

La función de la herramienta de este trabajo es procesar código fuente en cualquier momento que el usuario lo requiera. Para los programadores, durante la implementación, es usual que se quiera realizar alguna operación con el código desde cualquier editor.

Para facilitar esta actividad y aprovechando las ventajas que brindan algunos programas se decide que la herramienta en su ejecución sea capaz de recibir como parámetro un fragmento de código o el directorio de los archivos de código a analizar.

Se implementó la función *Main* para que reciba el código en forma de cadena de caracteres (*string* en C#). Luego, en la construcción del formulario principal se verifica si se obtuvieron parámetros y en caso de que fuera así se muestra el código correspondiente listo para procesar.

A partir de esto, cualquier programa que lo permita podrá invocar la ejecución de la aplicación asignándole un fragmento de programa o los archivos donde se encuentra el mismo.

El Visual Studio por ejemplo, permite agregar herramientas externas para que enriquezcan sus funcionalidades. Da la posibilidad de asignar un juego de teclas de acceso rápido escogido por el usuario para realizar la ejecución fácilmente durante la codificación.

Los argumentos que da a escoger este IDE para pasarle a la nueva herramienta son:

- Ruta de acceso del elemento.
- Directorio del elemento.
- Nombre de archivo del elemento.
- Extensión del elemento.
- Línea actual.
- Columna actual.
- Texto actual.
- Ruta de acceso de destino.
- Directorio de destino.
- Nombre de destino.
- Extensión de destino.
- Directorio del proyecto.

- Nombre de archivo del proyecto.
- Directorio de la solución.
- Nombre de archivo de la solución.(31)

El término “elemento” se refiere al archivo que corresponde al código fuente que se encuentra activo en el editor, línea actual, columna actual y texto actual se refieren: al número de la línea donde se encuentra el cursor, el número de la columna donde se encuentra el cursor y la palabra que se encuentra debajo del cursor si no existe una línea seleccionada respectivamente.

El SharpDevelop brinda también la posibilidad de vincular herramientas externas con la posibilidad de pasarle parámetros. Los parámetros que ofrece son los mismos que ofrece Visual Studio pero aportando uno: Carpeta de inicio de #Develop. Sin embargo SharpDevelop no brinda la posibilidad de ejecutar una herramienta mediante juegos de teclas de acceso rápido (32).

La herramienta que se propone, toma como parámetros el código a analizar en la primera posición y en la segunda posición la dirección de una carpeta de archivos o la dirección de un archivo específico con código. Si la posición cero contiene un fragmento de código este se carga para ser analizado. Si no, se verifica la segunda posición. Si en esta aparece la dirección de un archivo, se carga y si es una carpeta se cargan los archivos de código que contenga esa carpeta.
Si se desea integrar con otros sistemas para que sea ejecutada automáticamente con los parámetros antes especificados, se puede hacer siguiendo el mismo procedimiento descrito anteriormente.

2.3.3 Integración de HAC con herramientas para el dibujo de grafos

Durante la aplicación de la técnica del camino básico se obtiene la matriz del grafo de flujo de la sección de programa que se está procesando. Con herramientas que estén diseñadas e implementadas para el dibujo de grafos, se puede visualizar el grafo correspondiente a esta matriz y de esta forma el usuario puede ver los nodos y sus aristas gráficamente y corroborar el flujo del programa así como realizar un análisis visual de las particularidades del algoritmo.
Es necesario que se comuniquen ambas aplicaciones y para ello resulta intuitivo poder guardar un archivo que represente la matriz para ser abierto en la herramienta que sea capaz de dibujar el grafo de esa matriz.
Existen algunas herramientas que se dedican al dibujo y procesamiento de grafos sin embargo, es importante destacar una de ellas que ha ganado gran aceptación por las funcionalidades que brinda y la amigabilidad de su diseño visual.

Se trata de Grafos, programa desarrollado por Alejandro Rodríguez Villalobos, Profesor Titular de la Universidad Politécnica de Valencia. Está desarrollado en .NET en lenguaje Visual Basic (33). Está compartido bajo la licencia Reconocimiento-NoComercial-CompartirIgual 3.0 (en Inglés Creative Commons License) que establece la copia y distribución libre y gratuita en formas no comerciales y que cualquier referencia debe contener la cita al autor. Esta herramienta brinda gran cantidad de posibilidades en el trabajo con grafos entre las que se encuentran:

- Representación de grafos en forma gráfica y tabular.
- Almacenamiento de grafos en varios formatos ya sea estándares para grafos como imágenes.
- Algoritmos esenciales de la teoría de grafos como Camino mínimo, Camino máximo, Coste mínimo, Flujo máximo y Distancia total mínima.

Permite además dibujar grafos a partir de matrices que pueden ser cargadas desde archivos de texto que deben tener formatos específicos pero a la vez flexibles y fácilmente configurables por el usuario. Es precisamente esta funcionalidad la que se aprovecha para integrar HAC con Grafos.

Grafos permite configurar el formato de la matriz según los siguientes criterios:

- Matriz binaria.
- Matriz mínimo.
- Matriz máximo.
- Matriz coste.
- Matriz etiqueta.
- Matriz valor.
- Matriz etiqueta y valor.
- Matriz etiqueta y coordenadas XY.

Se pueden especificar además el carácter que se usará para indicar el fin de línea, el carácter que se usará como separador entre los valores de la matriz y el carácter que se usará para el caso que no exista valor en una posición.

Por convención y buscando legibilidad se decidió que el formato en que HAC guardará sus matrices en forma de texto es:

- Matriz binaria.
- Sin carácter de fin de línea.
- El carácter “|” (barra vertical) como separador entre los valores.
- El carácter “0” (cero) para el caso que no existan valores.

La matriz binaria es cuadrada y contiene tantas filas y columnas como nodos tenga el grafo que ella representa. En la intersección de un nodo *A* con un nodo *B* se ubicará un “1” (uno) si existe una arista

que une *A* con *B*. Si entre *A* y *B* no existe arista alguna, en la intersección [*A*, *B*] se ubicará un “0” (cero).

La matriz en el archivo de texto tiene el aspecto de la Figura 12. La matriz de la figura es binaria y representa un grafo de 8 nodos y 8 aristas. Posee el formato que se usará en HAC para generar las matrices.

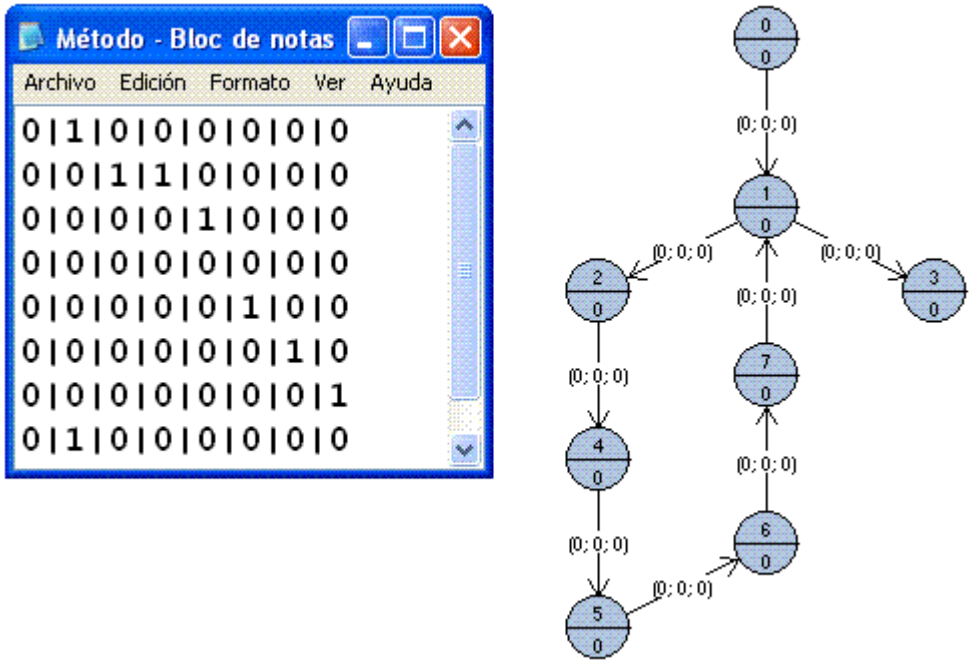


Figura 12 Dibujo mediante Grafos del grafo correspondiente a una matriz de ocho nodos y ocho aristas generada por HAC.

Al poder visualizarse el grafo a partir de la matriz, el usuario puede hacer análisis más profundos de forma visual y entonces la herramienta toma utilidad en otras áreas, no solo durante el desarrollo. En la docencia, durante la impartición de temas de pruebas de Caja Blanca y especialmente el Camino Básico pudiera aprovecharse sus ventajas.

Se recomienda a aquellos que necesiten trabajar con grafos para cualquier propósito que estudien y exploten las funcionalidades que Grafos brinda.

2.3.4 Ampliación con nuevos analizadores para otros lenguajes de programación

Inicialmente HAC fue implementado para procesar código C# realizando sobre el mismo las funcionalidades establecidas para la comprobación de estándares de codificación, la determinación de parámetros de calidad y la aplicación de la técnica del camino básico. Potencialmente, el usuario

pudiera necesitar el análisis de otros lenguajes. Para esto se diseñó la posibilidad de agregar nuevos analizadores para otros lenguajes y de esta forma enriquecer la utilidad de la herramienta.

Es entonces cuando se presenta la incógnita: ¿Cómo agregar los nuevos componentes de forma fácil sin tener que recompilar la aplicación?

Comúnmente, las aplicaciones toman las clases y funcionalidades de Librerías de Acceso Dinámico (archivos con extensión .dll). Es común también que los creadores de esos componentes los mejoren en algún momento manteniendo su estructura principal. Para actualizar las aplicaciones que hacen uso de dichos componentes sería necesario recompilarlas y así cambiar la referencia a la librería en cuestión.

El espacio de nombres *System.Reflection* de la biblioteca de clases de Microsoft .NET Framework brinda la posibilidad de crear aplicaciones que acepten la redirección de ensamblados (34). Esto consiste en enmendar el problema de tener que compilar aplicaciones para incorporar o actualizar librerías. El usuario puede agregar los nuevos elementos en tiempo de ejecución y la aplicación administra estos cambios mediante archivos de configuración.

Durante el desarrollo de HAC se hace uso del espacio de nombres antes mencionado para que se tomen dinámicamente las librerías con los nuevos analizadores para otros lenguajes de programación. La compatibilidad de estos analizadores vendría dada por la inclusión de una clase que serviría de intermediaria entre el analizador y la aplicación. El usuario pudiera configurar la herramienta con los aspectos necesarios para la admisión.

Durante la configuración se especifican elementos como el filtro para las extensiones de los archivos que manipula el nuevo lenguaje como se aprecia en la Figura 13.

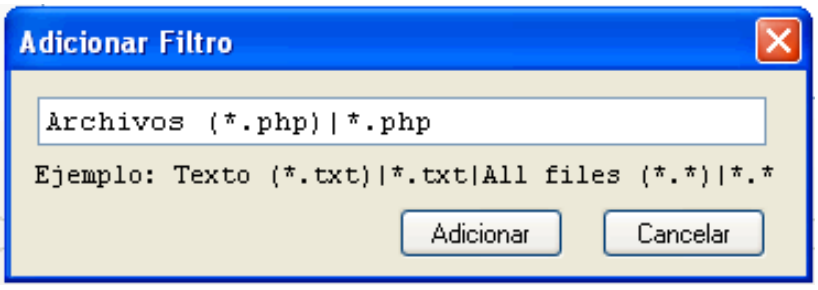


Figura13 Adicionar filtro para las extensiones de archivos del nuevo lenguaje.

Es necesario entonces seleccionar la librería donde se encuentra el analizador como se muestra en la Figura 14.

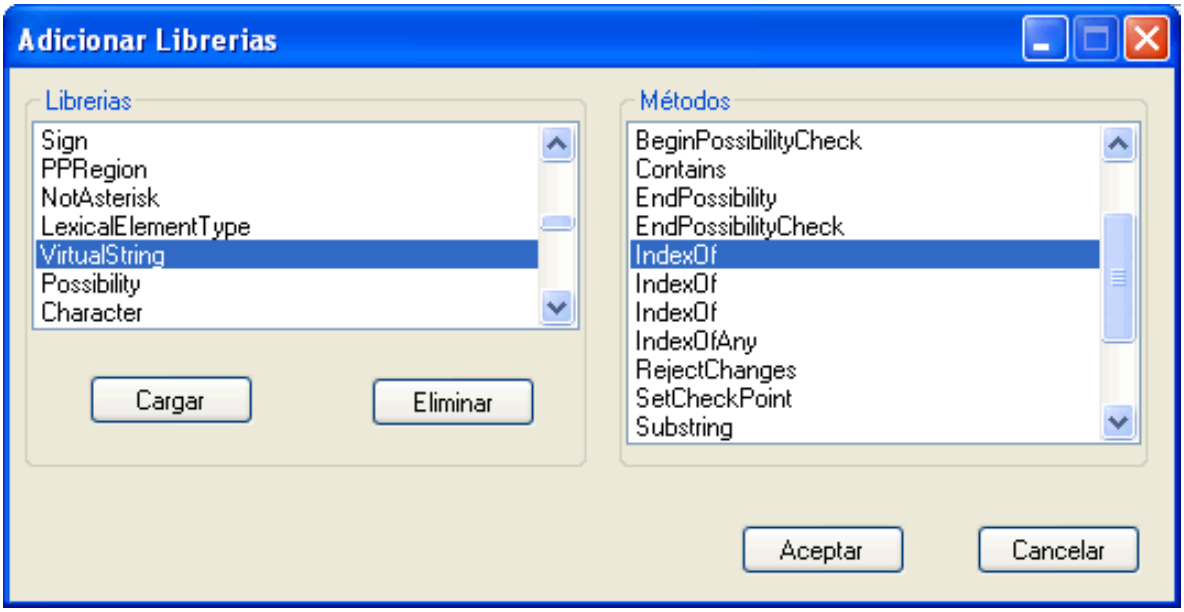


Figura 14 Adicionar librería con el analizador para el nuevo lenguaje.

Unido a esto se realizan ajustes que serían: escoger dentro de la librería la clase mediadora y los métodos de esta que son necesarios.

En esa forma se puede enriquecer la herramienta y procesar código fuente de varios lenguajes de programación.

2.4 Estándares de codificación

Es prudente establecer estándares de codificación para todos los programadores de cada equipo de desarrollo. Estos estándares consisten en estilos de codificación a la hora de escribir el código. Los aspectos para los que generalmente se establecen estándares son los siguientes (35):

- Identificadores.
- Indentación.
- Líneas y espacios en blanco.
- Comentarios.
- Declaraciones.
- Algoritmización.

Cada equipo define cuales de estos aspectos estandarizar y que estilo aplicar a cada aspecto. Existen innumerables combinaciones y muchos más aspectos que los especificados anteriormente.

Con el cumplimiento de estándares se persigue que parezca que todo el código haya sido implementado por una misma persona. De esta manera se consigue mayor legibilidad y facilidad de

mantenimiento. Los estándares deben responder además a acciones prácticas que acomoden al programador.
Un ejemplo de estándar puede ser escribir el identificador de cada función con letra inicial mayúscula. Otro puede ser ubicar la llave de apertura de bloques solitaria en la siguiente línea de la declaración del bloque.

A continuación se definirá qué estándares usar para la actividad de implementación correspondiente a este trabajo. Cabe destacar que estos estándares se definen teniendo en cuenta el estilo personal de cada programador, las características propias del lenguaje de programación, los recursos del lenguaje que se utilizarán y el tipo de programa que se debe implementar.

2.4.1 Identificadores

Para el caso de los identificadores existen estilos definidos mundialmente como el *lowerCamelCase* y el *UperCamelCase*. De la escritura del nombre de los mismos se puede inferir en qué consisten. Cada palabra interna en identificadores compuestos comienza con mayúsculas para ambos estilos. Para ambos estilos además ocurre que no se colocan caracteres de separación entre las palabras que conforman un identificador compuesto. Para el primero, el identificador comienza con minúscula y para el segundo, el identificador comienza con mayúscula.

- Interfaz: Para los identificadores de interfaces se escogió el *UperCamelCase* pero antecedido de la letra “I” en mayúscula. Ejemplo: “IEjemploInterfaz”.
- Espacio de nombre: Para los espacios de nombres se escogió el *UperCamelCase*.
- Clase: Para las clases se escogió el *UperCamelCase*.
- Estructura: Para las estructuras se escogió el *UperCamelCase*.
- Variable: Para las variables se establece el *lowerCamelCase*.
- Función: Para las funciones se establece el *UperCamelCase*.
- Enumerador: Para los enumeradores se establece el *UperCamelCase*.

Cada identificador, sin importar su tipo, debe ser lo suficientemente descriptivo. Se debe evitar escribir abreviaturas que pueden confundir a otros programadores por desconocer su significado. Se deben utilizar palabras que no den un significado ambiguo. A la vez, se deben buscar los sinónimos más cortos para que el identificador sea sencillo y a la vez expresivo.

2.4.2 Indentación

La indentación en una buena práctica de programación que consiste en comenzar a escribir cada línea de código a diferentes distancias desde el borde izquierdo del área de texto del editor. Esta distancia está determinada por la jerarquía que se forma al introducir sentencias dentro de bloques de estructuras. Esto persigue mayor legibilidad y entendimiento para el programador e igualmente depende del propio estilo de cada persona y del lenguaje y tipo de programa que se implementa.

Se definió que la indentación se hará agregando un *tab* al inicio de la línea que se desee escribir por cada unidad del grado de nivel que tenga la línea comenzando desde cero. La cantidad de espacios que incluya el *tab* dependerá del editor de texto y será, preferentemente, cuatro.
Se escribirá solo una sentencia por línea de código y en el caso de cortar las líneas, se hará luego de una coma o antes de un operador. La sección de la derecha de la línea que se corte se ubicará en la línea siguiente indentada al nivel de la expresión correspondiente en la línea superior.

2.4.3 Llaves

Existen diferentes criterios en cuanto a la ubicación de las llaves que delimitan el cuerpo de los bloques de código en los lenguajes que contienen este tipo de estructuras. Algunos programadores prefieren hacerlo ubicando la llave de apertura inmediatamente detrás de la línea cabecera del bloque mientras otros apuestan por ubicarlas de forma solitaria en la línea siguiente a la línea cabecera. Para este último estilo existen además diferencias en cuanto al nivel de indentación de las mismas. Algunos lo hacen al nivel de la línea cabecera y otros al nivel de las líneas del cuerpo del bloque. De igual forma algunos prefieren usar siempre las llaves mientras otros prefieren aprovechar las ventajas del lenguaje obviándolas en los casos que el cuerpo del bloque contenga solo una sentencia.
Para este trabajo:
Las llaves de apertura se colocarán solitarias en la línea siguiente e indentadas al nivel de la línea cabecera del bloque. Las llaves de cierre se colocarán solitarias en la línea que sigue a la última línea dentro del bloque e indentadas al nivel de la línea cabecera del bloque. En el caso de cuerpos de bloque con una sola sentencia se podrá o no usar las llaves a gusto del programador.
Es prudente señalar que este estilo agrega más líneas de código al programa al ubicar las llaves solitarias en una línea pero a su vez se gana en legibilidad del código. Es por eso que no existe una definición exacta de cual sería una buena práctica de programación en el uso de llaves debido a que cada método tiene ventajas y desventajas.

2.4.4 Líneas y espacios en blanco

Para mejorar la legibilidad del código muchas veces se utilizan líneas en blanco para separar segmentos de código que pueden corresponder a clases, funciones, declaraciones, implementaciones, comentarios, bloques o sencillamente secciones críticas que se deseen despejar. Así mismo sucede con los espacios en blanco cuando se utilizan para separar elementos dentro de las sentencias de código. A veces se separan con espacios cada operador de su respectivo operando, paréntesis, identificadores, símbolos y algunos lenguajes exigen que se separen las palabras propias del vocabulario de las adyacentes para ser comprendidas por los compiladores.

En este trabajo se colocarán líneas en blanco:

- Entre funciones.
- Antes de estructuras de control.
- Entre declaraciones de variables e implementaciones dentro del cuerpo de las funciones.
- Entre las declaraciones de interfaces, espacios de nombre, estructuras y clases.

Se colocarán espacios en blanco:

- Entre las palabras reservadas y los elementos adyacentes a las mismas.
- Después de las comas en la lista de argumentos de las funciones.
- Entre los operadores binarios y los elementos adyacentes a los mismos.
- Después de cada *punto-y-coma* (“;”) en las estructuras *for*.

Se puede observar que el uso de líneas en blanco incrementa la longitud en líneas de código del programa y el uso de espacios en blanco incrementa la longitud de cada línea de código del programa. Esto, sin embargo, mejora la legibilidad del código.
Es por esto que se deben establecer los estándares de conjunta aprobación entre los miembros del equipo de programadores.

2.4.5 Comentarios

El uso de comentarios durante la codificación puede parecer innecesario y que atrasa el proceso. Se ha demostrado, sin embargo, que esta práctica es beneficiosa por varias razones:

- Ayuda al programador a entender cada elemento o sección de código.
- Ayuda a adaptar el código durante su reutilización.
- Sirve de guía en los casos en que varios programadores trabajen sobre las mismas secciones del código.

- Disminuye el esfuerzo de análisis ya que el lenguaje natural es más legible que cualquier lenguaje de programación.
- Su importancia se aprecia al trabajar con código entre grandes intervalos de tiempo donde generalmente se olvida lo que se pensó en un momento.

Los comentarios se pueden utilizar para varios fines:

- Se utilizan para explicar el propósito de las funciones.
- Para explicar las características fundamentales de las clases.
- Para sintetizar las acciones de los algoritmos complejos.
- Para aclarar los datos que representan las variables.
- Para dividir secciones de código en dependencia de los diferentes contextos.
- Para dejar constancia del autor y algunas condiciones en que se generó el código.
- A veces, se usan comentarios temporales para recordar cosas que faltan, se deben modificar o analizar en otro momento.

Es necesario tener en cuenta algunos detalles al escribir comentarios:

- La capacidad de síntesis.
- El uso de lenguaje técnico.
- No repetir exactamente paso por paso lo que hace el algoritmo sino expresar un resumen de su propósito.
- Usar un estilo uniforme de comentario definido en estándares para todo el equipo.
- Escribir el comentario solo donde sea necesario.
- No usar expresiones obscenas e informales que dañen el ambiente de respeto entre los desarrolladores.

Es necesario aclarar que los propósitos, la importancia y las buenas prácticas en el uso de comentarios varían en dependencia de los programadores, el lenguaje de programación y el tipo de programa que se realice. Solo se presentaron los aspectos más usuales.

Existen dos tipos de comentarios:

- Comentarios de bloque o de varias líneas cuya sintaxis para el C# es *"/* Éste es un comentarios de bloque */*".
- Comentarios lineales o de una sola línea cuya sintaxis para el C# es *"// Éste es un comentario lineal*".

Identificada la necesidad de incluir comentarios en el código se definen estándares para homogeneizar los mismos.

En este trabajo se estableció la utilización de ambos tipos (comentarios de bloque y comentarios lineales). Se colocarán encima de la línea a la que se le quiera aplicar o encima de la línea cabecera del bloque al que se le desee aplicar. La indentación se hará al nivel de la línea en cuestión.
Se utilizarán en funciones y clases. Se puede utilizar en algoritmos no triviales y secciones de diferentes contextos dentro de los métodos. Se deben realizar revisiones para eliminar los comentarios temporales cuyos segmentos de código ya se hayan tratado.

2.4.6 Otros estándares

Se deben asignar los tipos adecuados a las variables para no hacer uso excesivo de recursos. Por ejemplo: Programando en C#; para una variable que almacena la cantidad de ocurrencias de un objeto en una colección cuya cantidad no supera las 100 unidades, es lógico asignar un tipo *int* a esa variable en lugar de un tipo *long* para ahorrar memoria y aumentar la eficiencia en tiempo de ejecución.
Cuando se precise realizar recorridos a través de colecciones, se deben utilizar las estructuras de control adecuadas. Por ejemplo: Programando en C#; al intentar recorrer una colección y en cuyo recorrido solo se tomarán valores de los objetos en cada posición, es oportuno hacerlo mediante un *foreach*. Sin embargo si se modificará la información en alguno de estos objetos, se puede utilizar un *for* en lugar del *foreach* porque este último no permite modificaciones.
Si se produce un entramado complejo de expresiones *if*, es prudente emplear el *switch*. Por ejemplo: Programando en C#; se pretende realizar acciones diferentes para cada valor que puede tener una variable cuyos valores será un número que representa un mes del año. Existirían doce casos que sería más fácil tratar con un *switch*.
Todos los atributos de las clases serán privados[18] y deben tener sus correspondientes propiedades[19]. El identificador de éstas, como todo método, debe responder al estilo *UperCamelCase*. Para los atributos estrictamente internos de las clases, que no tengan intercambio de información con el exterior, no es necesario implementar sus propiedades. Al referirse a atributos dentro de los métodos de una clase se hará invocando a su propiedad.
En las clases, se declararán primeramente los atributos, seguidamente los constructores, luego las propiedades y por último el resto de los métodos. Dentro de las funciones se declaran primero todas las variables locales y luego el resto de las instrucciones. Debe tenerse en cuenta que el identificador

[18] Condición establecida por el modificador de acceso “private” que mantiene al miembro accesible solo dentro de la clase.

[19] Métodos especiales llamados descriptores de acceso que controlan la asignación o lectura de información de los atributos de una clase.

de cada parámetro que reciba la función será leído por todo aquel que invoque a ese método dentro y fuera de la clase, por tanto debe tener una palabra razonable y descriptiva.

El nombre de cada componente que intervenga en el diseño de la interfaz visual estará compuesto por un sufijo seguido de una palabra que exprese su acción, dato que contenga o el contexto en que se ubica. El sufijo corresponderá al tipo de componente según la Tabla 3.

Tabla 3 Sufijo para el nombre de cada componente en dependencia de su tipo.

Tipo de componente	Sufijo
Button	bt
CheckBox	ch
ComboBox	cb
FolderBrowserDialog	fbd
GroupBox	gb
Label	lb
ListBox	lbx
ListView	lv
MenuStrip	ms
NumericUpDown	nud
OpenFileDialog	ofd
Panel	pnl
ProgressBar	pb
RadioButton	rb
ReportViewer	rv
RichTextBox	rtb
SaveFileDialog	sfd
SplitContainer	spc
StatuStrip	ss
TabControl	tc
TableLayoutPanel	tlp
TabPage	tp
TextBox	tb
ToolStrip	ts
ToolStripContainer	tsc

TreeView	tv
WebBrowser	wb

En la tabla anterior se relacionaron los componentes más usados en los diseños visuales de .NET. Si se utiliza otro que no aparezca en la tabla se establecerá un sufijo para el mismo.

Por ejemplo: Al usar un *TextBox* para escribir el nombre de una persona se puede nombrar como "tbNombrePersona". Obsérvese que los sufijos están en minúsculas, así el identificador de cada componente cumplirá con el *lowerCamelCase* exigido para los atributos.

Los elementos en los formularios se organizarán respetando las distancias recomendadas por el diseñador de formularios.

Las etiquetas correspondientes a las directivas de preprocesador[20] se escribirán indentadas al nivel del código que modifican.

Estas prácticas mencionadas anteriormente y otras que contribuyan a mejorar la legibilidad, organización y eficiencia del código no se deben olvidar.

Conclusiones

En este capítulo se realizó un análisis del diseño entregado por el analista donde se hicieron algunas modificaciones y aportes necesarios en el proceso de implementación para obtener los resultados deseados. Se hizo un esbozo de los componentes utilizados en la solución de las funcionalidades principales.

Se logró establecer una línea base sencilla que permita la programación eficiente por parte de ambos desarrolladores. Se establecieron los estándares de codificación pertinentes para que el código generado tenga la legibilidad necesaria al trabajar con algoritmos críticos y grandes volúmenes de código. La estructura de empaquetamiento de la aplicación permite futuras modificaciones puntuales de forma fácil sin afectar los demás elementos. La estructura que se diseñó para el flujo de trabajo con la herramienta simplifica la labor del usuario.

Se trataron algunos conceptos fundamentales para la comprensión de algunos temas. Se propusieron ejemplos sencillos en los momentos que se creyó necesario incluyendo figuras para su comprensión. Se explicó la implementación de aquellas partes que pudieran resultar confusas, ambiguas o complicadas.

[20] Directivas que se utilizan para facilitar la compilación condicional.

CAPÍTULO 3: Implementación del sistema

Introducción

En este capítulo se explicarán las estructuras de datos utilizadas para manejar los datos en el programa. Se hace un análisis de la complejidad y funcionamiento de los algoritmos más importantes para la implementación de la herramienta. Finalmente, se presenta la descripción de las clases fundamentales que definen el comportamiento del sistema con sus atributos y métodos esenciales.

3.1 Estructuras de datos utilizadas

Los datos semejantes y relacionados se pueden agrupar y tratarlos a la vez en lugar de escribir segmentos de código para trabajar con cada elemento. Resulta útil formar colecciones como las que contiene el espacio de nombres *System.Collections* de la biblioteca de clases de Microsoft .NET Framework.

System.Collections contiene interfaces y clases como listas, pilas y colas que especializan las funciones generales de una colección común para ajustarse a problemas más prácticos. Estas están administradas por la clase *Array* que contiene métodos para crear, manipular, buscar y ordenar matrices.

En las siguientes secciones se detallan algunas de las más usadas.

3.1.1 La clase ArrayList

El *ArrayList* de *System.Collections* es una colección que acepta objetos de cualquier tipo a la vez. Permite el acceso mediante índices de tipo entero. La capacidad de almacenamiento aumenta automáticamente a medida que se agregan elementos.

En la Tabla 4 se describen algunos métodos de esta clase.

Tabla 4 Métodos más usados de la clase ArrayList.

Método	Parámetros	Descripción
ArrayList	int capacity, ICollection c	Crea una instancia de ArrayList
Count		Obtiene la cantidad de elementos de la lista
Add	Object value	Agrega un elemento al final de la lista
BinarySearch	Object value	Verifica si un objeto se encuentra en la lista

Insert	Object value, int index	Inserta un elemento en la posición indicada
Remove	Object value	Elimina un elemento de la lista
Sort	IComparer comparer	Ordena los elementos según criterio especificado

3.1.2 La clase List

La clase *List* de *System.Collections.Generics* representa una lista genérica como colección de elementos. Es el equivalente genérico de *ArrayList*. Al declarar una instancia de esta clase se debe especificar el tipo de dato de los elementos que contendrá la colección. Es útil al manejar grupos de objetos del mismo tipo porque no se tiene que hacer la conversión explícita al referirse a uno de ellos. Permite acceder a los elementos mediante índices. Su capacidad aumenta al agregar nuevos objetos. Organiza la lista y elimina objetos.

En la Tabla 5 se muestran sus métodos más importantes.

Tabla 5 Métodos más usados de la clase List.

Método	Parámetros	Descripción
List	int capacity, IEnumerable<Collection> collection	Crea una instancia de List para almacenar objetos del tipo especificado
Count		Obtiene la cantidad de elementos de la lista
Add	Type item	Agrega un elemento al final de la lista
BinarySearch	Type item	Verifica si un objeto se encuentra en la lista
Insert	Type item, int index	Inserta un elemento en la posición indicada
Remove	Type item	Elimina un elemento de la lista
Sort	IComparer <Type> comparer	Ordena los elementos según criterio especificado
ForEach	Action<Type> action	Realiza la acción especificada para cada elemento de la lista
IndexOf	Type item	Devuelve la posición del elemento

		especificado

3.1.3 La clase Queue

La clase *Queue* de *System.Collections* representa una cola de elementos. Las colas establecen el procedimiento de "*primero en entrar, primero en salir*" (en inglés "*first in, first out*", FIFO). De esta forma, al agregar elementos a una cola se ubican al final y al extraerlos se hace desde el principio. No permite acceder a los elementos mediante índices. Su capacidad aumenta al agregar nuevos objetos.
En la Tabla 6 se muestran sus métodos más importantes.

Tabla 6 Métodos más usados de la clase Queue.

Método	Parámetros	Descripción
Queue	int capacity, ICollection col	Crea una instancia de Queue
Count		Obtiene la cantidad de elementos de la cola
Enqueue	object obj	Agrega un elemento al final de la cola
Dequeue		Elimina y devuelve el objeto situado al inicio de la cola
Peek		Devuelve el objeto situado al inicio de la cola
Contains	object obj	Verifica si un objeto se encuentra en la cola

3.1.4 La clase Stack

La clase *Stack* de *System.Collections* representa una pila no genérica de objetos. Las pilas administran colecciones bajo el procedimiento "*último en entrar, primero en salir*" (en inglés "*last in, first out*", LIFO). De esta forma, al agregar objetos a la pila se hace por el principio o tope y al extraerlos se hace de igual forma quitando el último que entró. No permite acceder a los elementos mediante índices. Su capacidad aumenta al agregar nuevos objetos.
En la Tabla 7 se muestran sus métodos más importantes.

Tabla 7 Métodos más usados de la clase Stack.

Método	Parámetros	Descripción
Stack	int initialCapacity, ICollection col	Crea una instancia de Stack
Count		Obtiene la cantidad de elementos de la pila
Push	object obj	Agrega un elemento en el tope o cima de la pila
Pop		Elimina y devuelve el objeto que se encentra en el tope de la pila
Peek		Devuelve el objeto que se encuentra en el tope de la pila

3.1.5 La clase Grafo

Otra importante estructura son los grafos que se utilizan para modelar problemas donde existan caminos, rutas, y recorridos. Un grafo se compone de nodos y sus relaciones. Un nodo *A* se relaciona con un nodo *B* mediante una arista o arco si desde *A* se puede llegar directamente a *B*. Un recorrido o secuencia de aristas se denomina “camino”.

Existen muchos tipos de grafos entre los que están los dirigidos, donde el desplazamiento de un nodo a otro se hace en una sola dirección o en ambas pero siempre se especifica. Además los arcos pueden tener características diferentes de los demás arcos que muchas veces suelen ser costos en tiempo, esfuerzo y recursos. Por ejemplo, si el grafo se utiliza para representar el trayecto de un ómnibus urbano dentro de una ciudad, los trayectos entre dos puntos pueden ser diferentes en cuanto a distancia, combustible necesario y tiempo del recorrido.

Existen diferencias además en cuanto a la topografía de los grafos donde se forman ciclos que determinan que desde un nodo *C* se puede llegar a él mismo recorriendo una secuencia de arcos determinada. En los grafos conexos siempre va a existir al menos un camino que recorra todos los nodos del grafo.

En este trabajo se representan computacionalmente los grafos mediante matrices de adyacencia que consiste en una matriz cuadrada de tantas filas y columnas como nodos tenga el grafo correspondiente. Para representar las aristas se coloca un “1” (uno) en la casilla que corresponde a la intersección de dos nodos y un “0” (cero) donde se crucen dos nodos que no se unen por ninguna arista.

En este trabajo se utilizó la implementación de una clase grafo genérico para representar el grafo de flujo correspondiente a los segmentos de código que se analicen en la aplicación de la técnica del Camino Básico.

Los métodos más importantes de la clase *Grafo* utilizada se presentan en la Tabla 8.

Tabla 8 Principales métodos de la clase grafo genérico.

Método	**Parámetros**	**Descripción**
Grafo		Crea una instancia de Grafo
Vertices		Obtiene o establece la lista de los vértices del grafo
MA		Obtiene o establece la matriz de adyacencia del grafo
CopiarEn	ref Grafo<T> g	Asigna una copia del grafo a uno dado
NumerodeVertices		Devuelve la cantidad de nodos
EstaVertice	T vert	Determina si un nodo se encuentra en el grafo
EstaArco	T vert1, T vert2	Determina si entre dos nodos dados existe alguna arista
AdicionarVertices	T v1	Adiciona un nodo dado al grafo
AdicionarArco	T v1, T v2	Adiciona una arista entre dos nodos dados
Adyacentes	T v1	Dado un nodo, devuelve una lista de sus nodos adyacentes
Existecamino	T v1, T v2	Determina si existe un camino que una a dos nodos dados
EsConexo		Determina si el grafo es conexo
BPP	T v1	Realiza el algoritmo de recorrido en profundidad
RecorridoProfundidad		Dirige el recorrido en profundidad
BPA	T v1	Realiza el algoritmo de recorrido a lo ancho
RecorridoaloAncho		Dirige el recorrido a lo ancho

3.1.6 La clase Arbol

En los problemas donde existen elementos que se relacionan de forma jerárquica el modelado se debe hacer en forma de árbol. Esto es un tipo de grafo dirigido porque desde el padre se puede llegar al hijo pero desde el hijo no se puede llegar al padre. Este diseño consiste en un entramado formado por nodos y sus relaciones. Existen varios modelos de árboles en la resolución de numerosos problemas de la programación. Todos los modelos se basan en que un nodo *A* referencia a un nodo *B* donde *A* se denomina "padre" de *B* e igualmente *B* sería "hijo" de *A*. El nodo que no tenga padre, o el nodo para el cual no exista un nodo que haga referencia a él, se denomina nodo "raíz" y el mismo es único para todo el árbol. Puede existir más de un nodo que no tenga hijos o que no haga referencia a ningún otro nodo, en tal caso se le denomina nodo "hoja". Un nodo puede tener o no hijos y si los tiene, pueden ser más de uno.

La mayor ventaja de los árboles está en el fenómeno de la recursividad. Esta es una alternativa a los tradicionales bucles y consiste esencialmente en una función que se llama a sí misma repetidas veces hasta que se cumpla una condición llamada "condición de parada".

Por ejemplo: Tanto pudiera obtenerse la sumatoria de los números desde el uno hasta el diez mediante un ciclo como con la llamada a un método recursivo. En la Figura 15 se muestran dos trozos de código que hacen esta función. El primero utiliza la recursividad y el segundo un ciclo *for*. El código está en C#.

```
//Sumatoria con recursividad
public int SumatoriaRecursiva(int contador)
{
    if (contador == 10)
    {
        return contador;
    }
    return contador + SumatoriaRecursiva(contador + 1);
}

//Sumatoria con bucle
public int SumatoriaConBucle()
{
    int sumatoria = 0;
    for (int i = 1; i <= 10; i++)
    {
        sumatoria += i;
    }
    return sumatoria;
}
```

Figura 15 Sumatoria con recursividad

La recursividad es posible por la forma en que se relacionan los nodos. Resumidamente si se quisiera obtener las hojas bastaría con preguntar recursivamente a cada hijo si es hoja comenzando desde la raíz.

La otra aplicación de los árboles consiste en la organización de datos para realizar algoritmos de búsqueda y ordenamiento más eficientes. Entre los modelos más usados para estos propósitos se encuentran los Árboles Binarios que tienen como característica distintiva que cada nodo puede tener como máximo dos hijos. Este tipo de árbol es usado para organizar datos y luego poder realizar búsquedas eficientes en esas colecciones.

Por ejemplo: El árbol binario de la Figura 16 permite organizar alfabéticamente las palabras mientras se van insertando. Formar una lista con las palabras ya ordenadas se hace más eficiente usando estos artefactos que con los métodos tradicionales de ordenamiento sobre arreglos.

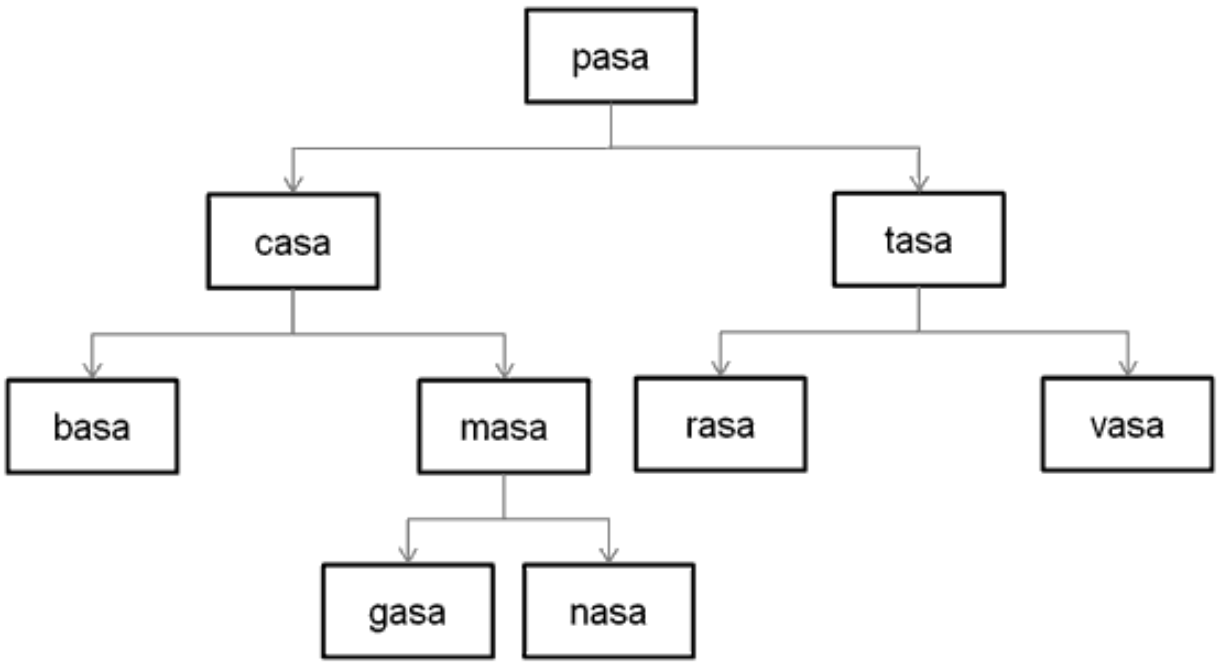


Figura 16 Árbol Binario para organizar palabras en orden alfabético.

En este caso, buscar la primera palabra por orden alfabético sería preguntar recursivamente a cada hijo izquierdo si es hoja comenzando desde la raíz. Por el contrario, si se tuvieran las palabras en un arreglo, habría que recorrerlo hasta el final para conocer la primera palabra.

Existen varios algoritmos fundamentales para aprovechar las ventajas de estas estructuras.

- *A lo ancho*: Recorre el árbol comenzando desde la raíz visitando para cada nodo, sus nodos adyacentes (analógicamente, sus nodos hermanos). Para el árbol de la Figura 16 el recorrido sería {pasa, casa, tasa, basa, masa, rasa, vasa, gasa, nasa}.
- *En pre orden*: Recorre el árbol comenzando desde la raíz visitando primero esta y después haciendo un recorrido *en pre orden* para cada hijo de izquierda a derecha. Para el árbol de la Figura 16 el recorrido sería {pasa, casa, basa, masa, gasa, nasa, tasa, rasa, vasa}.
- *En post orden*: Recorre el árbol comenzando desde la raíz. Hace primero un recorrido *en post orden* para cada hijo de izquierda a derecha y luego coloca la raíz. Para el árbol de la Figura 16 el recorrido sería {basa, gasa, nasa, masa, casa, rasa, vasa, tasa}.
- *En in orden*: Recorre el árbol comenzando desde la raíz. Hace primero un recorrido *en in orden* del hijo más a la izquierda, luego coloca la raíz y después hace un recorrido *en in orden* para los restantes hijos. Para el árbol de la Figura 16 el recorrido sería {basa, casa, gasa, masa, nasa, pasa, rasa, tasa, vasa}.

En este trabajo se utilizó la implementación de una clase árbol genérico cuyos principales métodos se relacionan en la Tabla 9.

Tabla 9 Principales métodos de la clase árbol genérico.

Método	**Parámetros**	**Descripción**
Arbol	T valor, List<Arbol<T>> hijos	Crea una instancia de Arbol
Hijos		Obtiene o establece la lista de los hijos
Raiz		Obtiene la raíz del árbol
EsHoja		Determina si el árbol tiene hijos o no
Grado		Devuelve la cantidad de hijos
Peso		Devuelve la distancia desde la raíz hasta la hoja más lejana
Clone		Devuelve una copia del árbol
Equal	Arbol<T> subArbol	Determina si un subárbol es igual al árbol
ObtenerSubArbol	int posicion	Devuelve el hijo que se encuentra en una posición dada de la lista de hijos
AddSubArbol	Arbol<T> subArbol	Agrega un subárbol a la lista de hijos
EliminarSubArbol	int posicion	Elimina el hijo que se encuentre en una posición dada de la lista de hijos

InsertarSubArbol	Arbol<T> subArbol, int posicion	Inserta un subárbol en una posición dada en la lista de hijos
ALoAncho	List<T> lista	Realiza un recorrido a lo ancho por el árbol
PreOrden	List<T> lista	Realiza un recorrido en pre orden por el árbol
InOrden	List<T> lista	Realiza un recorrido en in orden por el árbol
PostOrden	List<T> lista	Realiza un recorrido en post orden por el árbol
Buscar	Arbol<T> praiz	Devuelve si existe un subárbol igual a uno dado

Esta clase se utilizó para representar la jerarquía de clases contenida en los paquetes de archivos a analizar durante la determinación de los parámetros de calidad del código fuente.

3.1.7 La clase Nodo

Como tipo de dato para manejar con los grafos se utilizó una implementación de la clase nodo. Estos nodos contienen tres atributos de tipo entero:

- numero: que lo identifica de los demás nodos.
- línea: que representa la línea en el código donde fue identificado el nodo.
- pos: que representa la posición dentro de la lista de elementos que tiene el inicio del nodo.

Solo tiene un método que es el constructor con parámetros de la clase y se encarga de inicializar las variables que son atributos con los valores que recibe el constructor como parámetros.

Esta clase es contenedora de información por excelencia por lo que no implementa ninguna función sobre sus datos.

3.2 Análisis de los algoritmos más críticos

En esta sección se hará un análisis de aquellos algoritmos que resultan importantes para la implementación de la herramienta o que su codificación es compleja. Aparecerá una breve explicación del valor y el contexto de uso de los métodos que los contienen. Se mostrará el código fuente de los mismos y se describirán los pasos principales en que consiste el algoritmo.

3.2.1 Algoritmo para cargar carpetas y archivos específicos de un directorio

Este algoritmo está contenido en el método *Directorios* de la clase *frm.HAC* del módulo *HAC*. Su función consiste en cargar las carpetas y los archivos determinados por un filtro que se encuentren dentro de un directorio especificado. Se usa para abrir estos elementos y organizarlos recursivamente en una instancia de *TreeNode* que es una clase de componente visual para manejar datos estructurados en forma de árbol. Hace uso de otro método llamado *Archivos* de la misma clase y será explicado en la próxima sección. Su importancia está dada porque en casos en que el usuario especifique una carpeta con numerosos archivos y subcarpetas, la operación se pudiera tardar varios segundos.

Como se muestra en la Figura 17 el método *Directorios* presenta como único elemento de complejidad un ciclo cuyas iteraciones dependen de la cantidad de archivos contenidos en el directorio especificado. El método *Archivos* es similar en cuanto a estructura y complejidad.

```
private TreeNode Directorios(string direccion)
{
    string[] directorios = Directory.GetDirectories(direccion);
    TreeNode nodo = new TreeNode(direccion.Substring(
        direccion.LastIndexOf(@"\") + 1), 0, 0);
    Array.Sort(directorios);
    foreach (string archivo in directorios)
    {
        nodo.Nodes.Add(Directorios(archivo));
    }
    Archivos(direccion, nodo);
    return nodo;
}
private void Archivos(string directorio, TreeNode nodo)
{
    string filter = configuracion.Filtros[
        configuracion.FiltroSelecionado].ExT;
    string[] files = Directory.GetFiles(directorio, filter);
    Array.Sort(files);
    foreach (string var in files)
    {
        TreeNode nuevoNodo = new TreeNode(
            var.Substring(var.LastIndexOf(@"\") + 1), 1, 1);
        nuevoNodo.Tag = var;
        nuevoNodo.ContextMenuStrip = contextMenuArchivo;
        archivoscargados.Add(var);
        nodo.Nodes.Add(nuevoNodo);
    }
}
```

Figura 17 Método Directorios y método Archivos de la clase frm.HAC del módulo HAC.

Pasos del algoritmo del método *Directorios*:

1. Declarar un arreglo de cadenas y almacenar en el mismo los directorios contenidos en uno específico.
2. Crear una instancia de *TreeNode* pasándole como parámetros el nombre del directorio.
3. Organizar el arreglo de directorios.
4. Hacer un ciclo de tantas iteraciones como elementos en el arreglo de directorios donde se invoque recursivamente al mismo método para almacenar todos los elementos de la jerarquía.
5. Invocar al método Archivos para agregar los mismos según filtro.
6. Devolver el objeto *TreeNode* con la jerarquía organizada.

Pasos del algoritmo del método *Archivos*:

1. Guardar en una variable de cadena la extensión de archivo en el filtro de la configuración general.
2. Extraer los archivos que cumplan con el filtro desde el directorio indicado y guardarlos en un arreglo de cadenas.
3. Organizar el arreglo de cadenas.
4. Recorrer en ciclo en arreglo de cadenas creando en cada iteración una instancia de *TreeNode* y agregando la misma al *TreeNode* de los parámetros.

En el Capítulo 3 se analizará la complejidad de ambos métodos.

3.2.2 Algoritmo para extraer clases a partir del código fuente

El objetivo de este algoritmo es extraer las clases contenidas en un archivo con código fuente. La complicación de este algoritmo está dada por el hecho de que una clase puede estar declarada dentro de otra y a su vez declarar otras en su interior. Este tipo de anidamiento siempre induce a usar la recursividad. El algoritmo pertenece al método *ExtraerClasesRecursivo* de la clase *ExtraccionJerarquica* del módulo Camino Básico. La importancia de este método es que en los casos que existan en el código muchas clases y el nivel de anidamiento sea alto, el proceso puede demorar un poco más. En la Figura 18 se muestra el código del método.

```
private void ExtraerClasesRecursivo(Clase c)
{
    for (int i = c.inicio; i < c.fin; i++)
    {
        Elemento el = codigo[i];
        if (el.Tipo == TipoElemento.Keyword)
            if (el.Lexema == "class" || el.Lexema == "struct")
            {
                string nombre = GetNombreNamespace(i);
                int inicio = GetInicio(i);
                int fin = GetFinLlaves(inicio);
                Clase cl = new Clase(nombre, inicio, fin);
                cl.nombrePadre = GetNombrePadre(i);
                cl.nombreCompleto = c.nombreCompleto + "." + cl.nombre;
                ExtraerFunciones(cl);
                c.clases.Add(cl);
                i = fin;
            }
            else
                continue;
    }
    foreach (Clase cl in c.clases)
    {
        ExtraerClasesRecursivo(cl);
    }
}
```

Figura 18 Método ExtraerClasesRecursivo de la clase ExtraccionJerarquica del módulo Camino Básico.

Pasos del algoritmo del método *ExtraerClasesRecursivo*:

1. Recorrer en ciclo la los elementos del código desde el inicio hasta el fin de la clase pasada como parámetro. Si el elemento en la posición actual es una clase o estructura, crearla, extraer sus funciones y agregarla a la lista.
2. Recorrer la lista de clases y para cada clase extraer las clases en su interior invocando recursivamente al propio método.

3.2.3 Algoritmo para organizar la jerarquía de clases

Este algoritmo se basa en el típico algoritmo "*Burbuja*" de ordenamiento de colecciones. Su objetivo es organizar la jerarquía de clases poniendo delante en la lista las clases que no tienen padre. Pertenece al método Ordenar de la clase *JerarquiaCodigo* del módulo Parámetros de Código. Se centra en el recorrido de la lista mediante dos ciclos anidados. La Figura 19 muestra el código del método Ordenar.

```
private void Ordenar()
{
    for (int i = 0; i < clases.Count; i++)
    {
        for (int j = i; j < clases.Count; j++)
        {
            if (clases[i].nombrePadre != string.Empty)
            {
                if (clases[j].nombrePadre == string.Empty)
                {
                    Clase c = clases[i];
                    clases[i] = clases[j];
                    clases[j] = c;
                }
            }
        }
    }
}
```

Figura 2 Método Ordenar de la clase JerarquiaCodigo del módulo Parámetros del Código.

Pasos del algoritmo burbuja del método *Ordenar*:

1. Recorrer en ciclo la lista de clases.
2. En cada iteración recorrer la lista de clases a partir de la posición actual del ciclo más afuera.
3. En cada iteración del ciclo interior si la clase en la posición actual del ciclo externo tiene padre y la clase en la posición actual del ciclo interno no tiene padre intercambiar las posiciones entre dichas clases.

3.2.4 Algoritmo para exportar la matriz del grafo de flujo

El objetivo es convertir la matriz del grafo de flujo en texto con un formato específico y guardarlo en un archivo en la computadora. La importancia de este algoritmo radica en que si la matriz es muy grande la operación puede tardar varios segundos. Pertenece al método *ExportarMatrizParaGrafos* de la clase *McCabe* del módulo Camino Básico. La Figura 20 presenta el código del método *ExportarMatrizParaGrafos*.

```
public string ExportarMatrizParaGrafos(string direccion)
{
    bool[,] matriz = grafo.MA;
    int dimensionMatriz = int.Parse(
        Math.Sqrt(double.Parse(matriz.Length.ToString(),
        System.Globalization.NumberStyles.Integer)).ToString());
    //Texto que se guardara en el bloc de notas
    string textoMatriz = "";
    //Caracteres que llevara la matriz en forma de texto
    string[,] matrizTexto = new string[dimensionMatriz, dimensionMatriz];
    //Llenar la matriz de textos
    for (int i = 0; i < dimensionMatriz; i++)
    {
        for (int j = 0; j < dimensionMatriz; j++)
        {
            if (matriz[i, j] == true)
            {
                matrizTexto[i, j] = "1";
            }
            else
            {
                matrizTexto[i, j] = "0";
            }
        }
    }
    //Llenar el texto que contendra el bloc de notas
    int dimensionMatrizTexto = int.Parse(
        Math.Sqrt(double.Parse(matrizTexto.Length.ToString(),
        System.Globalization.NumberStyles.Integer)).ToString());
    for (int i = 0; i < dimensionMatrizTexto; i++)
    {
        for (int j = 0; j < dimensionMatrizTexto; j++)
        {
            textoMatriz += matrizTexto[i, j];
            if (j < dimensionMatrizTexto - 1)
            {
                textoMatriz += "|";
            }
        }
        textoMatriz += System.Environment.NewLine;
    }
    File.WriteAllText(direccion, textoMatriz);
    return textoMatriz;
}
```

Figura 20 Método ExportarMatrizParaGrafos de la clase McCabe del módulo Camino Básico.

Pasos del algoritmo del método *ExportarMatrizParaGrafos*:

1. Guardar la matriz en un arreglo bidimensional *booleano*.
2. Guardar las dimensiones en una variable de tipo entero.
3. Crear una variable de cadena de caracteres para guardar el texto que se pondrá en el archivo.
4. Crear una matriz bidimensional de cadenas de caracteres con el mismo tamaño que la matriz booleana.

5. Llenar la matriz de cadenas a partir de la matriz booleana haciendo dos ciclos anidados.
6. En el recorrido se coloca un cero (“0”) en la matriz de cadenas en las posiciones donde exista un “false” en la matriz booleana y se coloca un uno (“1”) en la matriz de cadenas en las posiciones donde exista un “true” en la matriz booleana.
7. Conformar el texto a guardar en la variable de cadena recorriendo la matriz de cadenas en dos ciclos anidados.
8. Se coloca una barra vertical (“|”) a continuación de cada uno y cada cero excepto después del último de cada fila, en cuyo caso se agrega una cadena especial de fin de línea.
9. Se hace uso del método *WriteAllText* de la clase *File* del espacio de nombres *System.IO* de la biblioteca de clases de Microsoft.NET Framework para crear un archivo con el texto formado y guardarlo en el directorio pasado como parámetros.

3.2.5 Algoritmo para encontrar los nodos siguientes de un nodo

El objetivo es encontrar los nodos con los que un nodo dado comparte una arista siendo este último el origen. Los nodos se guardarán en una lista organizados ascendentemente según su grado. El grado está determinado por la cantidad de aristas que parten de un nodo. Este algoritmo pertenece al método *BuscarSiguientes* de la clase *CIndependientes* del módulo Camino Básico.

Este método se usa durante la generación de un conjunto básico de caminos independientes. Es importante debido a que su ejecución debe ser rápida, ya que esta operación hay que realizarla para encontrar cada nodo que conformará el camino y al ser largo el camino o al existir gran cantidad de nodos en la matriz, el proceso puede tornarse complejo. La Figura 21 muestra el código del algoritmo del método *BuscarSiguientes*:

```
private List<int> BuscarSiguientes(int nodoActual)
{
    List<int> nodosSiguientes = new List<int>();
    for (int i = 0; i < aristas.Count; i++)
    {
        if (aristas[i].DesdeNodo == nodoActual)
        {
            bool insertado = false;
            for (int j = 0; j < nodosSiguientes.Count; j++)
            {
                if (CantidadAristasNodo(nodosSiguientes[j]) >
                    CantidadAristasNodo(aristas[i].HastaNodo))
                {
                    nodosSiguientes.Insert(j, aristas[i].HastaNodo);
                    insertado = true;
                    break;
                }
            }
            if (!insertado)
            {
                nodosSiguientes.Add(aristas[i].HastaNodo);
            }
        }
    }
    return nodosSiguientes;
}
```

Figura 21 Método BuscarSiguientes de la clase CIndependientes del módulo Camino Básico.

Pasos del algoritmo del método *BuscarSiguientes*:

1. Crear una lista de enteros que representa los nodos siguientes.
2. Hacer un ciclo para recorrer la lista de aristas.
3. En cada iteración si se cumple que la arista sale del nodo dado insertar el nodo destino de la arista en cuestión en la lista de nodos siguientes de forma organizada en dependencia de la cantidad de aristas que salgan del nodo destino.
4. Devolver la lista de nodos siguientes.

Es necesario aclarar que se aseguró la ejecución de estos algoritmos de forma que no se consuman muchos recursos si la tarea se torna compleja. Para ello se utilizó la programación en varios hilos de ejecución que consiste en ordenar el desarrollo de un proceso independiente del flujo normal de la aplicación.

3.3 Descripción de las clases utilizadas

A continuación se hará una descripción de las clases que se utilizaron para modelar el problema.
Las clases se describirán en una tabla que consta de los siguientes campos:

- Nombre: En este campo se escribirá el nombre o identificador de la clase.
- Tipo de clase: En este campo se especificará el tipo de clase que representa la misma para el diseño de la solución.
- Atributo: En este campo se escribirá el nombre o identificador para cada atributo que contenga la clase en cuestión. Las líneas de este campo se incrementan para cada atributo de la clase.
- Tipo: En este campo se especificará el tipo de dato al que pertenece el atributo cuyo identificador aparece a la izquierda. Las líneas de este campo son, de igual forma, incrementables para cada atributo de la clase.
- Nombre (Para cada responsabilidad): En este campo se escribirá el nombre o identificador de cada función o método de la clase seguido de los parámetros que la misma debe recibir (con su tipo de dato e identificador encerrados entre paréntesis). Las líneas de este campo se incrementarán para cada función de la clase.
- Descripción (Para cada responsabilidad): En este campo se escribirá una breve descripción del objetivo o acción principal de la función correspondiente unida a alguna observación oportuna que se requiera de la misma. Las líneas de este campo son, de igual forma, incrementables para cada función de la clase.

Tabla 9 Descripción de la clase *Controladora* del módulo *App.HAC*.

Nombre: Controladora	
Tipo de clase: Controladora	
Atributo:	**Tipo:**
archivo	string
config	Config
Para cada responsabilidad:	
Nombre:	Controladora(Config config)
Descripción:	Constructor de la clase.
Nombre:	Ejecutar()
Descripción:	Ejecuta el análisis léxico del fichero que se encuentre en el atributo “archivo”.
Nombre:	SetFile(string file)
Descripción:	Cambia el archivo que se este analizando.

Tabla 10 Descripción de la clase *HiloExtraccionJerarquica* del módulo *App.HAC*.

Nombre: HiloExtraccionJerarquica	
Tipo de clase: Controladora	
Atributo:	**Tipo:**
carpeta	string
matrizCodigo	ArrayList
espacioNombre	List<EspacioNombre>
callback	CalculoCallback
Para cada responsabilidad:	
Nombre:	HiloExtraccionJerarquica(ArrayList p, CalculoCallback callback)
Descripción:	Constructor de la clase.
Nombre:	HiloExtraccionJerarquica(string folder, List<EspacioNombre> espacioNombre)
Descripción:	Constructor de la clase.
Nombre:	ExportarMatriz()
Descripción:	Inicia la acción de exportar matrices.
Nombre:	CalculoComplejoArchivo()
Descripción:	Inicia un hilo de ejecución para realizar el análisis a un archivo.
Nombre:	CalculoComplejoCodigo()
Descripción:	Inicia un hilo de ejecución para realizar el análisis a un fragmento de código.

Tabla 11 Descripción de la clase *Manejador* del módulo *Camino Básico*.

Nombre: Manejador	
Tipo de clase: Controladora	
Atributo:	**Tipo:**
espacioNombre	List<EspacioNombre>
codigo	List<Elemento>

Para cada responsabilidad:	
Nombre:	Manejador(ArrayList matrizCodigo,bool archivo)
Descripción:	Constructor de la clase.
Nombre:	AnalizarArchivo(ArrayList matrizCodigo)
Descripción:	Realiza el análisis de un archivo a partir de su matriz de tokens.
Nombre:	AnalizarCodigo(ArrayList matrizCodigo)
Descripción:	Realiza el análisis de un fragmento de código a partir de su matriz de tokens.
Nombre:	ExportarMatriz(string folder)
Descripción:	Exporta la matriz para una carpeta especificada
Nombre:	Codigo
Descripción:	Devuelve el código como una lista de elementos.
Nombre:	EspacioNombres
Descripción:	Devuelve los espacios de nombre encontrados en el código.

Tabla 12 Descripción de la clase *EstandaresCodificacion* del módulo *Estandares de codificacion*.

Nombre: EstandaresCodificacion		
Tipo de clase: Controladora		
Atributo:		**Tipo:**
Para cada responsabilidad:		
Nombre:	ConvertirCodigoElementos(ArrayList codigo)	
Descripción:	Extrae los elementos de la matriz en el ArrayList y los ubica en una lista insertando un elemento de fin de línea donde corresponda	
Nombre:	QuitarEspacios(List<Elemento> codigo)	
Descripción:	Elimina los elementos de tipo espacios en blanco	
Nombre:	AnalizarCodigo(ArrayList tokens, ConfiguracionEstandares configuracion)	
Descripción:	Es el método principal que ordena la comprobación de estándares de todo el	

	código haciendo uso de los restantes métodos
Nombre:	ComentarioPrevio(int linea, List<Elemento> elementos)
Descripción:	Determina si inmediatamente antes de una línea dada existen comentarios
Nombre:	ComentarioPosterior(int linea, List<Elemento> elementos)
Descripción:	Determina si inmediatamente después de una línea dada existen comentarios
Nombre:	GradoIndentacionLinea(int posLinea, List<Elemento> elementos)
Descripción:	Devuelve un número que representa el nivel de indentación de la línea dada
Nombre:	AnalizarIndentacionAlineacion(List<Elemento> codigo, int espacios, ConfiguracionEstandares configuracion)
Descripción:	Comprueba el grupo alineación de la indentación en el código
Nombre:	QuitarComentarios(List<Elemento> elementos)
Descripción:	Elimina los comentarios de una lista de elementos
Nombre:	AnalizarIdentificadores(List<Elemento> codigo, ConfiguracionEstandares configuracion)
Descripción:	Comprueba los estándares de los identificadores en el código
Nombre:	AnalizarVariable(string variable, ConfiguracionEstandares configuracion, int linea)
Descripción:	Comprueba los estándares de las variables en el código
Nombre:	AnalizarFuncion(string funcion, ConfiguracionEstandares configuracion, int linea)
Descripción:	Comprueba los estándares de las funciones en el código
Nombre:	AnalizarClase(string clase, ConfiguracionEstandares configuracion, int linea)
Descripción:	Comprueba los estándares de las clases en el código
Nombre:	AnalizarComentarios(List<Elemento> codigo, ConfiguracionEstandares configuracion)
Descripción:	Comprueba los estándares de las comentarios en el código
Nombre:	AnalizarIndentacion(List<Elemento> codigo, ConfiguracionEstandares configuracion)
Descripción:	Comprueba los estándares de la indentación en el código

Nombre:	AnalizarIndentacionLongitudDeLinea(List<Elemento> codigo, ConfiguracionEstandares configuracion)
Descripción:	Comprueba el grupo de la longitud de línea de la indentación en el código
Nombre:	AnalizarIndentacionCorteLinea(List<Elemento> codigo, ConfiguracionEstandares configuracion)
Descripción:	Comprueba el grupo del corte de línea de la indentación en el código
Nombre:	AnalizarIndentacionAlineacionLineaCortada(List<Elemento> codigo, ConfiguracionEstandares configuracion)
Descripción:	Comprueba el grupo la alineación la línea cortada de la indentación en el código
Nombre:	AnalizarLlaves(List<Elemento> codigo, ConfiguracionEstandares configuracion)
Descripción:	Comprueba los estándares de las llaves en el código
Nombre:	AnalizarLineasEspaciosBlanco(List<Elemento> codigo, ConfiguracionEstandares configuracion)
Descripción:	Comprueba los estándares de las líneas y los espacios en blanco en el código
Nombre:	AnalizarLineasBlanco(List<Elemento> codigo, ConfiguracionEstandares configuracion)
Descripción:	Comprueba los estándares de las líneas en blanco en el código

Tabla 13 Descripción de la clase *Busquedas* del módulo *Parametros del codigo.*

Nombre: Busquedas	
Tipo de clase: Controladora	
Atributo:	**Tipo:**
arbolClases	JerarquiaCodigo
halsteadLoc	List<HalsteadLoc>
reporteclases	List<ReporteClase>
archivos	List<string>
list	List<ArrayList>
Para cada responsabilidad:	

Nombre:	Busquedas(List<ArrayList>list, List<string> archivos)
Descripción:	Constructor de la clase.
Nombre:	ReportByFile()
Descripción:	Genera un reporte teniendo en cuenta cada archivo.
Nombre:	GenerarParametrosArchivo()
Descripción:	Genera todos los parámetros a mostrar en el reporte por archivo.
Nombre:	GenerarParametrosClases()
Descripción:	Genera todos los parámetros a mostrar en el reporte por clases.
Nombre:	ReporteClases()
Descripción:	Genera un reporte teniendo en cuenta cada clase

Conclusiones

En la implementación de la herramienta se utilizaron técnicas avanzadas de programación y ventajas que ofrece el lenguaje C# y la tecnología .NET. Entre los elementos más importantes empleados están la programación en hilos de ejecución, el uso oportuno de la Herencia y el Polimorfismo. El uso de reportes y componentes visuales novedosos que permiten la personalización y la flexibilidad. La utilización de la recursividad en varios algoritmos. La referencia dinámica de librerías. Las implementaciones de componentes a nivel de interfaz visual.

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

Introducción

En este capítulo se hace una búsqueda de las técnicas de prueba de software que permitan comprobar la validez de la solución obtenida. Se detallan los casos de prueba con sus objetivos. Se explican los juegos de datos utilizados para las pruebas y se analizan los resultados obtenidos.

4.1 Pruebas de software

La calidad de un sistema está determinada, entre otras cosas, por la coincidencia entre lo que se programó y los requisitos establecidos en la primera fase. Para comprobar el grado de cumplimiento de estos requisitos se usan las pruebas del sistema. Estas definen un conjunto amplio de acciones de comprobación que abarcan todas las características que determinan la calidad de un software.

Se comprueban las funcionalidades diseñando casos de prueba que definen cómo proceder. Estos casos de prueba incluyen los juegos de datos a usar que son los válidos o esperados y los no válidos o no esperados por el programa. Además establecen los resultados a alcanzar en correspondencia de la lógica del programa y los datos ingresados. Describen las condiciones generales en las que se debe aplicar las pruebas para obtener los objetivos propuestos. El objetivo de los casos de prueba es forzar al máximo el sistema en los puntos críticos para encontrar fallos y detectar defectos.

Las pruebas se deben aplicar durante todo el ciclo de vida del software e invariablemente se le debe dedicar una gran parte del esfuerzo total del desarrollo. Se deben planificar correctamente desde el inicio y establecer qué hacer, cómo hacer, quién va a hacer y en qué condiciones hacer las comprobaciones. Es beneficioso que los desarrolladores prueben su producto pero que no falte la mano de terceras personas que no intervinieron en el proyecto directamente ya que así se detecta mayor cantidad de fallas (36).

Se debe escoger los tipos de prueba que se adapten mejor al sistema que se va a probar. Para esto se debe tener en cuenta el lenguaje de programación, el proceso de desarrollo, las características de los desarrolladores, el tipo de funcionalidad que se implementa, la plataforma en que se ejecutan los procesos, los errores más importantes, si la aplicación es de escritorio o web, si realiza conexiones a bases de datos entre otras observaciones.

Los dos grandes grupos de pruebas unitarias existentes son las pruebas de Caja Negra y las pruebas de Caja Blanca.

4.1.1 Pruebas unitarias

Las pruebas unitarias permiten probar, como su nombre lo indica, cada unidad independiente del software. Actúan esencialmente sobre el código fuente y sobre los elementos básicos de la interfaz del módulo.

Según Pressman (37) los casos de prueba que se generan durante las pruebas de unidad deben estar encaminados a verificar los siguientes elementos:

- Interfaz: “Se prueba la interfaz del módulo para asegurar que la información fluye de forma adecuada hacia y desde la unidad de programa que está siendo probada”.
- Estructuras de datos locales: “Se examinan las estructuras de datos locales para asegurar que los datos que se mantienen temporalmente conservan su integridad durante todos los pasos de ejecución del algoritmo”.
- Condiciones límites: “Se prueban las condiciones límites para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento”.
- Caminos independientes: “Se ejercitan todos los caminos independientes (caminos básicos) de la estructura de control con el fin de asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez”.
- Caminos de manejo de errores: “… se prueban todos los caminos de manejo de errores”.

4.1.2 Pruebas de Caja Blanca

Las pruebas de Caja Blanca se nombran de esta forma porque a diferencia de las pruebas de Caja Negra que actúan sobre la interfaz, estas revisan la parte interna del software, específicamente sobre el código fuente. Se basan en el examen minucioso de los detalles procedimentales. Se comprueban los caminos lógicos del sistema generando casos de prueba que ejerciten las estructuras condicionales y los bucles. Es por esto que las Pruebas Unitarias se basan en las Técnicas de Pruebas de Caja Blanca.

Existen varios métodos que analizan diferentes partes del programa y se complementan entre sí para garantizar la calidad del sistema. Como se explicó en el Capítulo 2, la técnica del camino básico se utiliza para comprobar la complejidad lógica de un diseño procedimental. Permite diseñar casos de prueba para cubrir todas las sentencias de un programa a partir de la obtención de un conjunto de caminos independientes. La complejidad ciclomática, como resultado fundamental de estas pruebas, acota la cantidad mínima de casos de prueba que se deben ejecutar.

La Prueba de las Condiciones Es un método que se encamina hacia la ejercitación de las condiciones. Se basa en el principio de que si un conjunto de casos de prueba es capaz de ejercitar todas las

condiciones contenidas en un bloque de código, este mismo conjunto serviría para encontrar más errores en el programa que no tengan que ver directamente con las condiciones.

La Prueba del Flujo de Datos verifica la validez en el uso de las variables para manipular los datos de la aplicación. Selecciona los casos de prueba atendiendo a las definiciones y los usos de las variables. El procedimiento indica que se debe encontrar las sentencias donde se define cada variable y las sentencias donde se hace uso de las mismas. Luego se encuentran las "cadenas de definición - uso" que representan el ciclo de vida de las variables y se diseñan casos de prueba que las ejecuten en su totalidad.

La Prueba de los Bucles se centra en la validez de las estructuras cíclicas o bucles. El objetivo es probar el comportamiento de estas estructuras en sus valores límites de iteración. Los bucles se clasifican en cuatro tipos: Bucles simples, Bucles anidados, Bucles concatenados y Bucles no estructurados. Aunque la esencia es la misma, cada tipo se prueba de forma diferente.

De lo anterior pudiera pensarse que las pruebas de Caja Blanca logran enmendar todos los errores del programa. La desventaja de estas es entonces de tipo logística ya que resulta imposible abarcar todo el código fuente de un sistema medianamente grande. El tiempo necesario para realizarlas sería considerable y se torna compleja su aplicación sobre algoritmos críticos.

4.1.3 Pruebas de Caja Negra

Las Pruebas de Caja Negra deben su nombre a los elementos que estas revisan y las condiciones en que se hace la revisión. Estas se basan en los requerimientos funcionales del sistema y se llevan a cabo desde el exterior de la aplicación.

Técnica de la Partición Equivalente:

Es un método de prueba de caja negra que consiste en analizar los valores de entrada de un programa para diseñar casos de prueba con datos que representen clases válidas e inválidas. Los datos tienen tres tipos de condiciones de entrada:

- Condición de entrada lógica (en lo adelante CEL).
- Condición de entrada de Rango (en lo adelante CER).
- Condición de entrada de Valor (en lo adelante CEV).

Por ejemplo, un dato de entrada en un formulario correspondiente al nombre de un medicamento que sea dañino para una persona, tiene como condiciones de entrada:

- CEL: El nombre del medicamento puede o no aparecer.
- CEV: El nombre del medicamento si aparece es un alfanumérico sin signos.

Entonces, para probar la validez de asimilación de la entrada de dicho campo en el programa se pueden diseñar casos de prueba donde las entradas para dicho campo serían:

- Dejar el campo vacío.
- Ingresar “Dipirona”.
- Ingresar “Dipirona@”.

Análisis de Valores Límites:

Este método complementa al de Partición Equivalente. La diferencia está en que este selecciona los casos de prueba con valores que se encuentran en los límites de los rangos de validez. Por ejemplo, si un campo de entrada debe poseer valores comprendidos entre *A* y *B*. Se deben diseñar casos de prueba como sigue:

- Que el valor sea mayor que *A* y lo más próximo posible a este.
- Que el valor sea menor que *B* y lo más próximo posible a este.
- Que el valor sea menor que *A* y lo más próximo posible a este.
- Que el valor sea mayor que *B* y lo más próximo posible a este.
- Que el valor sea igual que *A*.
- Que el valor sea igual que *B*.

Las pruebas de caja negra son importantes a la hora de medir el grado de cumplimiento de los requerimientos solicitados por el cliente y se aplican sobre la interfaz de la aplicación observando las respuestas del sistema antes determinadas acciones y los datos de salida para determinados datos de entrada (38).

Al analizar los métodos de prueba de Caja Blanca y Caja Negra se llega a la conclusión de que es más factible aplicar las pruebas de Caja Negra para comprobar la validez en las respuestas del programa ante las acciones del usuario y la calidad de las salidas en dependencia de las entradas. Para comprobar la eficiencia de la codificación es más adecuado emplear las pruebas de Caja Blanca, más específicamente, la Técnica del Camino Básico.

Para probar la validez de la solución propuesta en la implementación de HAC se decidió probar los elementos fundamentales de la interfaz de usuario mediante casos de prueba de Caja Negra así como la validez de las salidas del sistema en dependencia de las entradas del usuario.

Los algoritmos más importantes o más complejos se comprobarán mediante la técnica del Camino Básico.

4.2 Diseño y ejecución de los Casos de Prueba de Caja Blanca

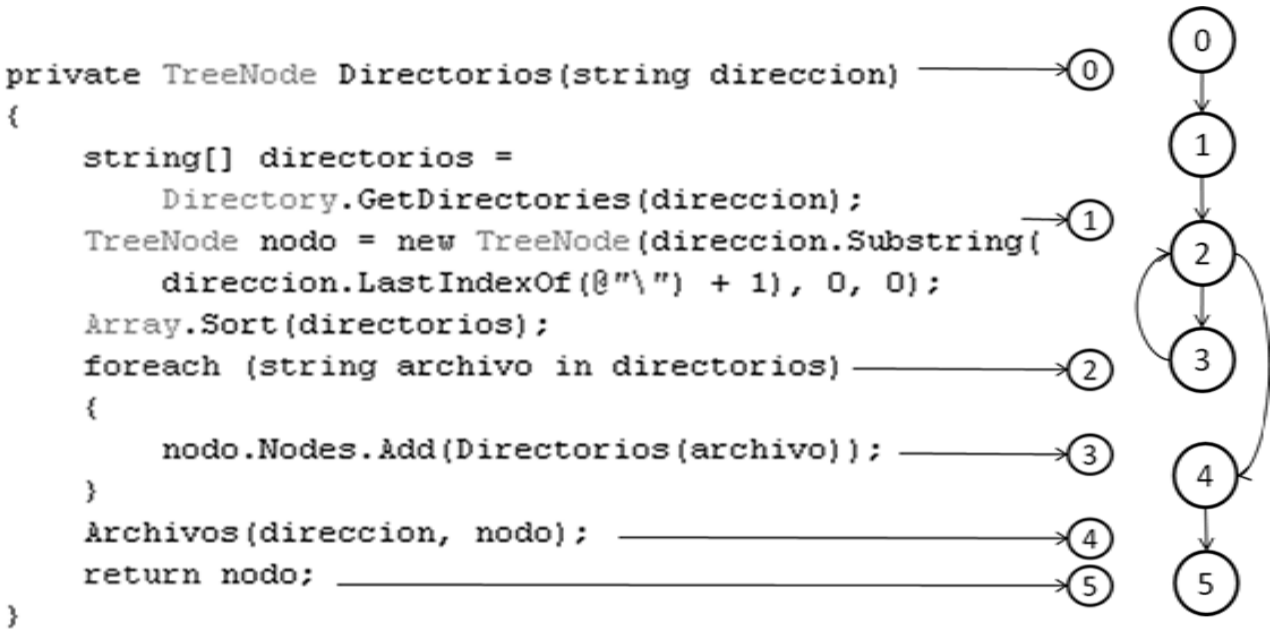
En esta sección se escogerán algunos algoritmos o funciones importantes en la implementación de HAC para diseñar los casos de prueba mediante la Técnica del Camino Básico. El proceso se realizará de forma manual y automatizada.

Primeramente se aplicará la técnica de forma manual obteniendo el grafo de flujo correspondiente al código. Luego se calculará la complejidad ciclomática mediante las tres variantes explicadas. Por último se obtiene un conjunto de caminos independientes para el grafo obtenido.

A continuación de esto se aplicará la técnica de forma automatizada utilizando HAC y Grafos. Mediante HAC se calcula la complejidad ciclomática y se genera la matriz del grafo. Luego se importa la matriz desde Grafos y se visualiza el grafo correspondiente. Por último se calcula la complejidad ciclomática por las tres vías ya explicadas y se muestra un conjunto de caminos independientes.

4.2.1 Método Directorios de la clase frm.HAC del módulo HAC

Como se explicó en el Capítulo 3, el objetivo de este método es cargar todas las subcarpetas y archivos C# contenidos en un directorio especificado por el usuario. El método *Directorios* hace uso del método *Archivos* y sus implementaciones son semejantes. A continuación, en la Figura 22 y en la Figura 23 se aplica la Técnica del Camino Básico de forma manual para ambas funciones respectivamente.



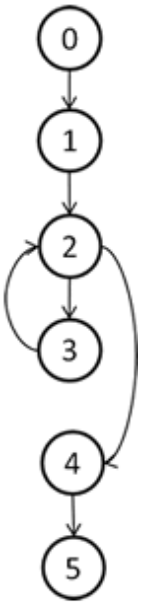
A = 6 (aristas)	P = 1 (nodos predicado)	R = 2 (regiones del grafo)
N = 6 (nodos)	V(G) = P + 1	V(G) = R
V(G) = A – N + 2	V(G) = 1 + 1	V(G) = 2
V(G) = 6 – 6 + 2	V(G) = 2	
V(G) = 2		

Conjunto de caminos independientes:

- 0 – 1 – 2 – 4 – 5
- 0 – 1 – 2 – 3 – 2 – 4 – 5

Figura 22 Aplicación de la Técnica del Camino Básico de forma manual al método Directorios de la clase frm.HAC del módulo HAC.

```
private void Archivos(string directorio, TreeNode nodo)
{
    string filter = configuracion.Filtros[
        configuracion.FiltroSelecionado].ExT;
    string[] files = Directory.GetFiles(directorio, filter);
    Array.Sort(files);
    foreach (string var in files)
    {
        TreeNode nuevoNodo = new TreeNode(
            var.Substring(var.LastIndexOf(@"\") + 1), 1, 1);
        nuevoNodo.Tag = var;
        nuevoNodo.ContextMenuStrip = contextMenuArchivo;
        archivoscargados.Add(var);
        nodo.Nodes.Add(nuevoNodo);
    }
}
```



A = 6 (aristas)
N = 6 (nodos)
V(G) = A – N + 2
V(G) = 6 – 6 + 2
V(G) = 2

P = 1 (nodos predicado)
V(G) = P + 1
V(G) = 1 + 1
V(G) = 2

R = 2 (regiones del grafo)
V(G) = R
V(G) = 2

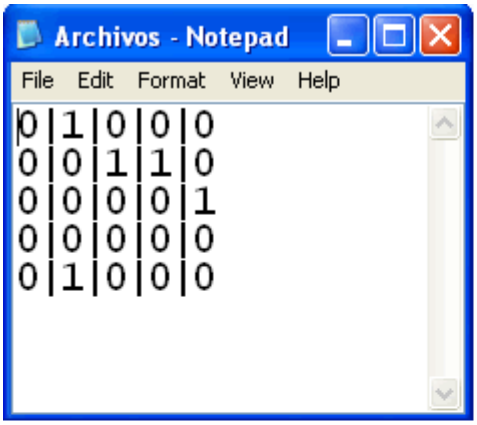
Conjunto de caminos independientes:

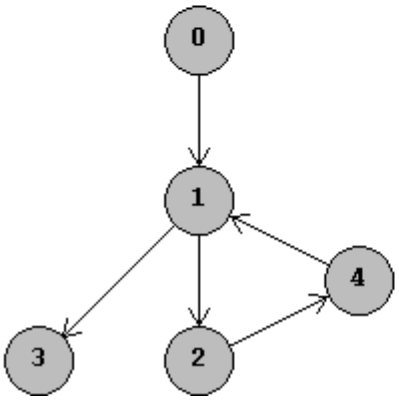
- 0 – 1 – 2 – 4 – 5
- 0 – 1 – 2 – 3 – 2 – 4 – 5

Figura 23 Aplicación de la Técnica del Camino Básico de forma manual en el método Archivos de la clase frm.HAC del módulo HAC.

Una vez realizada el procedimiento de forma manual se procede a utilizar las herramientas informáticas para comparar ambos resultados. La matriz en forma de texto que se presenta es la que se genera en HAC y el grafo presentado es el que dibuja Grafos al importar dicha matriz.

En la Figura 24 se realiza el proceso automatizado para ambos métodos. Se utiliza solo una figura ya que para ambos métodos HAC genera matrices idénticas y por tanto, los grafos dibujados serían iguales también. Esto ocurre por la semejanza en el flujo de control de ambos algoritmos.





A = 5 (aristas)
N = 5 (nodos)
V(G) = A – N + 2
V(G) = 5 – 5 + 2
V(G) = 2

P = 1 (nodos predicado)
V(G) = P + 1
V(G) = 1 + 1
V(G) = 2

R = 2 (regiones del grafo)
V(G) = R
V(G) = 2

Conjunto de caminos independientes:
- 0 – 1 – 3
- 0 – 1 – 2 – 4 – 1 …

Figura 24 Aplicación de la Técnica del Camino Básico de forma automática en el método Directorios y en el método Archivos de la clase frm.HAC del módulo HAC.

Como se puede apreciar los dos métodos analizados son muy sencillos y únicamente poseen un ciclo cuyas iteraciones están determinadas por la cantidad de archivos y subcarpetas del directorio seleccionado. La complejidad ciclomática es 2, por lo que se considera óptima.
Se aprecia que entre el grafo del procedimiento manual y el grafo del procedimiento automático existen diferencias. Sin embargo, si se analiza con detenimiento sus estructuras, se puede observar que haciendo ajustes sobre la base de la eliminación de secuencias lineales de nodos, se pueden obtener grafos muy parecidos. Lo que determina el cálculo de la complejidad ciclomática es la cantidad de nodos predicados presentes, la proporción entre nodos y aristas y la cantidad de regiones del grafo. De esta manera, pueden existir grafos diferentes pero que mantienen iguales estos aspectos. En esos casos se puede decir que los grafos son equivalentes en términos de la complejidad ciclomática.

A continuación los casos de prueba que forzarán los caminos del método Directorios.
Caso de prueba del camino 1:

- Pasarle como parámetro la siguiente dirección "D:\\CarpetaVacia" que es una carpeta que no contiene archivos para que no pase el bucle.
  Se retorna una instancia de *TreeNode tn* tal que se cumpla que *tn.Nodes.Count* sea cero.

Caso de prueba del camino 2:

- Pasarle como parámetro la siguiente dirección "D:\\CarpetaConUnArchivo" que es una carpeta con un archivo C# para que haga una iteración en el bucle.
  Se retorna una instancia de *TreeNode tn* tal que se cumpla que *tn.Nodes.Count* sea uno.

A continuación los casos de prueba que forzarán los caminos del método Archivo.

Caso de prueba del camino 1:

- Pasarle como parámetro la siguiente dirección "D:\\CarpetaVacia" que es una carpeta que no contiene archivos para que no pase el bucle. Y un objeto *TreNode tn*.
  Finalmente se modifica *tn* tal que se cumpla que *tn.Nodes.Count* sea cero.

Caso de prueba del camino 2:

- Pasarle como parámetro la siguiente dirección "D:\\CarpetaConUnArchivo" que es una carpeta con un archivo C# para que haga una iteración en el bucle.
  Se retorna una instancia de *TreeNode tn* tal que se cumpla que *tn.Nodes.Count* sea uno.

4.2.2 Método ExtraerClasesRecursivo de la clase ExtraccionJerarquica del módulo Camino Básico

Como aparece en el Capítulo 3, este método extrae las clases contenidas en un archivo con código fuente. En la Figura 25 se muestra la aplicación de la Técnica del Camino Básico de forma manual para el mismo.

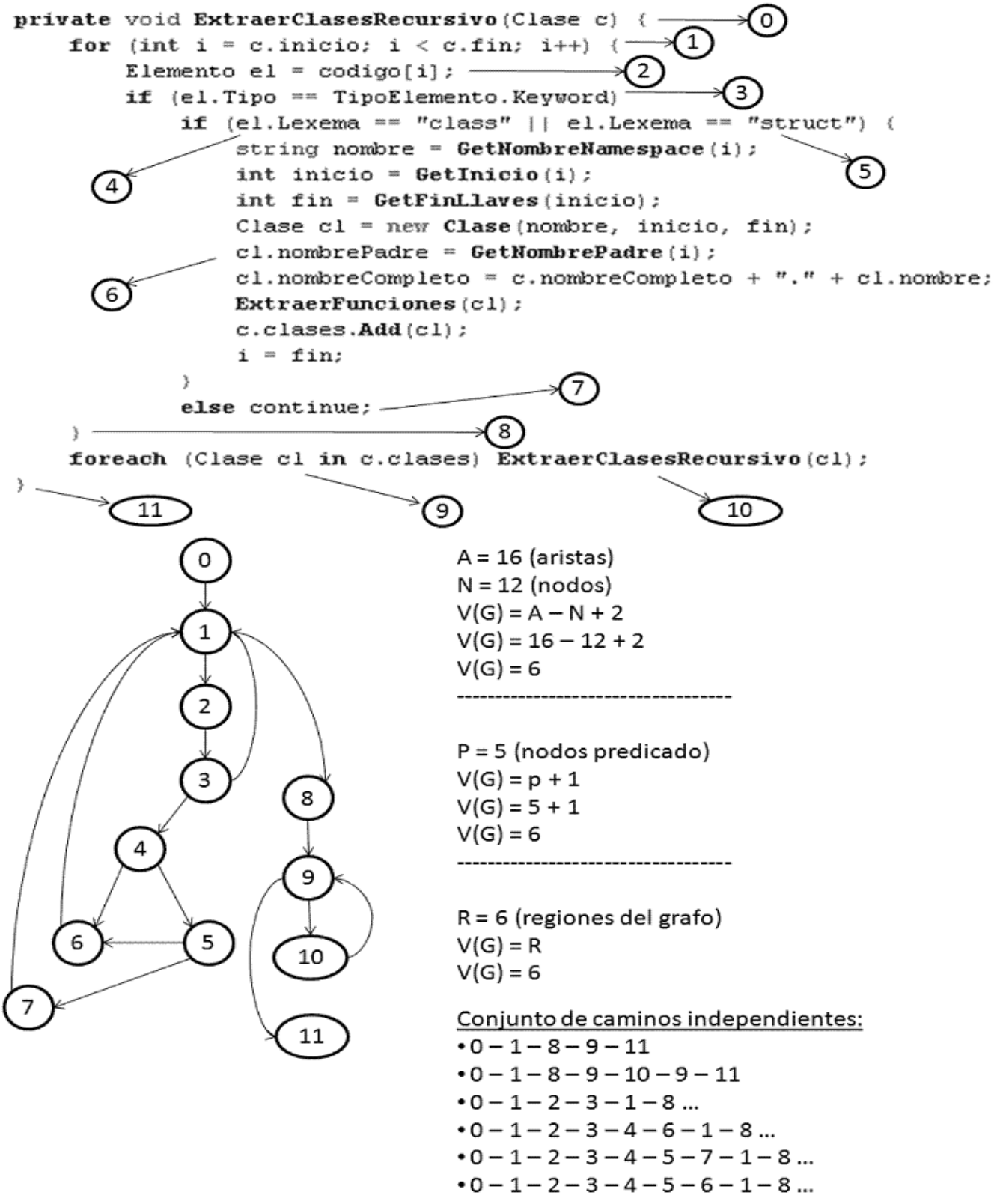
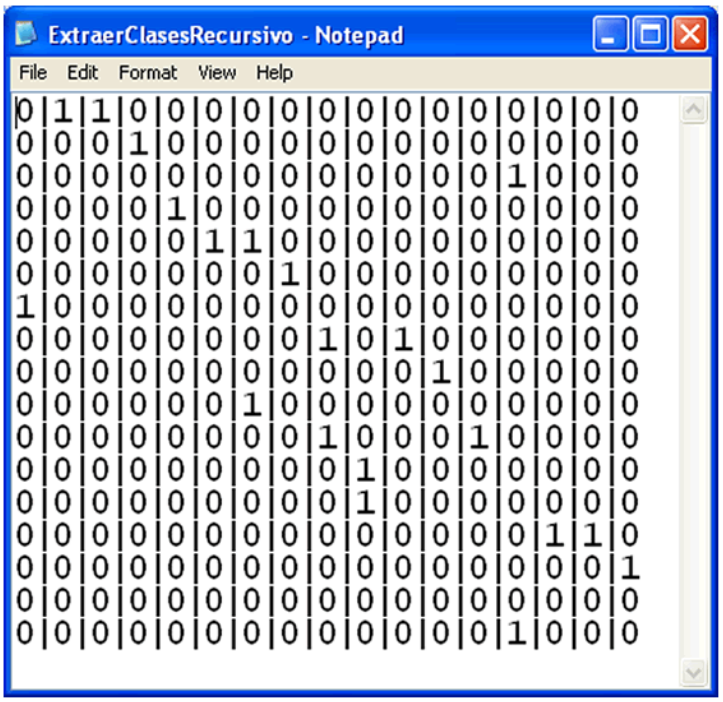
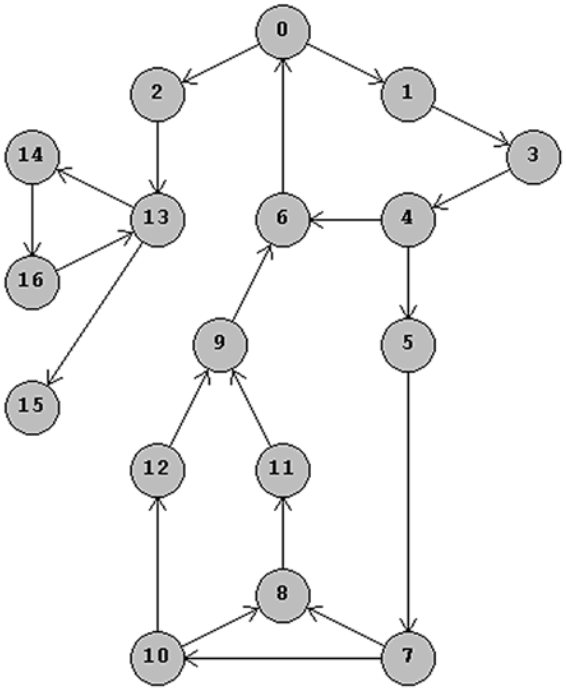


Figura 25 Aplicación de la Técnica del Camino Básico en el método ExtraerClasesRecursivo de la clase ExtraccionJerarquica del módulo Camino Básico.

Como se realizó con los dos métodos anteriores, luego de realizar el procedimiento manual, se procede a realizarlo de forma automática. En la Figura 26 se presenta el proceso como se explicó anteriormente utilizando la matriz generada por HAC y el grafo de dicha matriz dibujado por Grafos.





A = 20 (aristas)
N = 16 (nodos)
V(G) = A – N + 2
V(G) = 20 – 16 + 2
V(G) = 6

P = 5 (nodos predicado)
V(G) = P + 1
V(G) = 5 + 1
V(G) = 6

R = 6 (regiones del grafo)
V(G) = R
V(G) = 6

Conjunto de caminos independientes:

- 0 – 2 – 13 – 15
- 0 – 2 – 13 – 14 – 16 – 13 …
- 0 – 1 – 3 – 4 – 6 – 0 …
- 0 – 1 – 3 – 4 – 5 – 7 – 8 – 11 – 9 – 6 …
- 0 – 1 – 3 – 4 – 5 – 7 – 10 – 8 …
- 0 – 1 – 3 – 4 – 5 – 7 – 10 – 12 – 9 – 6 …

Figura 26 Aplicación de la Técnica del Camino Básico de forma automática en el método ExtraerClasesRecursivo de la clase ExtraccionJerarquica del módulo Camino Básico.

Como se pudo observar, nuevamente coincidió el resultado principal del cálculo de la complejidad ciclomática aunque se pueden observar diferencias en cuanto a la cantidad de nodos y aristas en los grafos usados en ambos procedimientos.

A continuación se diseñan los casos de prueba que cubran los caminos independientes presentados.
Caso de prueba del Camino 1:

- Se pasa como parámetro una instancia de Clase donde *c.fin* sea igual a *c.inicio* para que salte el primer bucle. Que la lista de clases de c esté vacía para que salte el segundo bucle.

Caso de prueba del Camino 2:

- Se pasa como parámetro una instancia de Clase donde *c.fin* sea cero para que salte el primer bucle. Que la lista de clases de c tenga una sola clase para que pase el segundo bucle en una iteración.

Caso de prueba del Camino 3:

- Se pasa como parámetro una instancia de Clase donde *c.fin* sea mayor que *c.inicio* en una unidad para que pase el primer bucle en una sola iteración. Que el elemento en la posición *c.inicio* no sea de tipo “Keyword” para que no se cumpla la primera condición. A partir de este momento, en todos los caminos restantes, se procede de la misma manera que en el caso de prueba del camino 1 para que salte el segundo bucle.

Caso de prueba del Camino 4:

- Se pasa como parámetro una instancia de Clase donde *c.fin* sea mayor que *c.inicio* en una unidad para que pase el primer bucle en una sola iteración. Que el elemento en la posición *c.inicio* sea de tipo “Keyword” y su lexema sea “class” para que se cumpla la condición compuesta.

Caso de prueba del Camino 5:

- Se pasa como parámetro una instancia de Clase donde *c.fin* sea mayor que *c.inicio* en una unidad para que pase el primer bucle en una sola iteración. Que el elemento en la posición *c.inicio* sea de tipo “Keyword” y su lexema sea distinto de “class” y sea distinto de “struct” para que no se cumpla la condición compuesta.

Caso de prueba del Camino 6:

- Se pasa como parámetro una instancia de Clase donde *c.fin* sea mayor que *c.inicio* en una unidad para que pase el primer bucle en una sola iteración. Que el elemento en la posición *c.inicio* sea de tipo “Keyword” y su lexema sea “struct” para que se cumpla la condición compuesta.

La ejecución de los Casos de Prueba de Caja Blanca de los métodos seleccionados para probar, fue satisfactoria en todos los casos. Se logró cubrir todos los caminos y con ellos todas las aristas del grafo. La complejidad ciclomática es baja lo que permite un mayor entendimiento, facilidad de prueba, facilidad de modificación y de reutilización.

4.3 Diseño y ejecución de los Casos de Prueba de Caja Negra

Para diseñar los casos de prueba de Caja Negra se utilizarán las plantillas que actualmente se utiliza en el Grupo de Calidad de la Facultad 7 durante las pruebas a los proyectos de dicha facultad antes de ser liberados.

En la Tabla 14 se agrupan las diferentes secciones del flujo de trabajo de HAC y sus escenarios.

- Nombre de la sección: es el nombre que describe a qué sección se hace referencia.
- Escenarios de la sección: son los nombres que describen a qué escenarios de la sección se hace referencia.
- Descripción de la funcionalidad: Es una breve descripción de lo que se hace en el escenario.
- Flujo Central: Son las acciones generales a ejecutar para poder llevar a cabo las pruebas en el escenario indicado.

Tabla 10 Secciones y escenarios a probar.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: Pantalla principal.	EC 1.1: Abrir carpeta de archivos	Abrir una carpeta que contenga archivos C# para su posterior análisis	Desplegar el menú Archivo y en el submenú Abrir escoger Carpeta
	EC 1.2: Abrir archivo	Abrir un archivo C# para su posterior análisis	Desplegar el menú Archivo y en el submenú Abrir escoger Archivo
	EC 1.3: Abrir Introducir código	Abrir la ventana para introducir código	Desplegar el menú Archivo y escoger Introducir Código
	EC 1.4: Cerrar todos	Cerrar todos los archivos y carpetas de archivos abiertos	Desplegar el menú Archivo accionar la opción Cerrar todos o hacer clic en el botón Cerrar todos de la barra de Herramientas
	EC 1.5: Salir	Salir de la aplicación	Desplegar el menú Archivo y escoger Salir

	EC 1.6: Abrir Complejidad ciclomática	Mostrar la ventana Complejidad Ciclomática	Abrir archivo o carpeta de archivos y seleccionar uno. Desplegar el menú Análisis y escoger Complejidad Ciclomática o Accionar el botón de herramienta Complejidad Ciclomática
	EC 1.7: Abrir Estándares de Codificación	Mostrar la ventana estándares de codificación	Abrir archivo o carpeta de archivos y seleccionar uno. Desplegar el menú Análisis y escoger Estándares de Codificación o Accionar el botón de herramienta Estándares de Codificación
	EC 1.8: Abrir Parámetros del Código	Mostrar la ventana parámetros del código	Abrir archivo o carpeta de archivos. Desplegar el menú Análisis y escoger Parámetros del Código
	EC 1.9: Pegar	Abrir ventana de introducir código con el texto del portapapeles de Windows	Desplegar el menú Edición y escoger Pegar
	EC 1.10: Abrir Configuración de Estándares de Codificación	Abrir ventana de Configuración de estándares de codificación	Desplegar el menú Configuración y escoger Estándares de Codificación
	EC 1.11: Abrir Opciones	Abrir ventana para configurar los analizadores y los filtros de archivos	Desplegar el menú Configuración y escoger Opciones
	EC 1.12: Ayuda. Acerca de HAC	Mostrar la información general acerca de HAC	Desplegar el menú Ayuda y escoger Acerca de HAC
	EC 1.13: Ayuda. Pruebas de Caja Blanca	Mostrar la información general acerca de las pruebas de Caja Blanca	Desplegar el menú Ayuda y escoger Pruebas de Caja Blanca
SC 2: Estándares de Codificación	EC 2.1: Analizar los Estándares.	Analizar el código cargado según la configuración especificada	Hacer clic en la opción Analizar del menú Acciones o hacer clic en el botón Realizar Análisis de la barra de herramientas

	EC 2.2: Cerrar	Cerrar la ventana donde se esta realizando el análisis	Hacer clic en la opción Cerrar del menú Acciones o en el botón cerrar de la barra de titulo
SC 3: Complejidad Ciclomática	EC 3.1: Analizar Todos	Analizar todas las funciones que se encuentran cargadas, calculando la complejidad ciclomática	Hacer clic en el botón Analizar Todos o hacer clic en la opción Analizar Todos del menú Acciones
	EC 3.2: Exportar Matriz	Exportar la matriz que representa el grafo de flujo de una función seleccionada	Hacer clic derecho en una función y escoger Exportar Matriz
	EC 3.3 : Exportar Matrices	Exportar todas las matrices	Hacer escoger la opción Exportar Matrices del menú Acciones
SC 4: Parámetros del Código	EC 4.1 Reporte Archivo	Extraer parámetros de Archivos en forma de reporte	Hacer clic en el botón Reporte Archivo
	EC 4.2 Exportar Reporte	Exportar el reporte de archivos	Hacer clic en exportar y escoger la opción PDF, luego escoger la dirección y hacer clic en guardar
	EC 4.3 Reporte Clases	Extraer los parámetros de Clases en forma de reporte	Hacer clic en el botón Reporte Clases
SC 5 : Configuración de estándares de Codificación	EC 5.1 Abrir Configuración	Abrir la Configuración desde archivo	Hacer clic en el botón Abrir configuración de estándares existente y sobrescribir la actual
	EC 5.2 Guardar Configuración	Guardar la Configuración hacia un archivo	Hacer clic en el botón Guardar Estándares Como… y escoger la dirección donde se desea guardar el estándar y dar guardar
	EC 5.3 Mostrar	Mostrar la configuración estándares	Hacer clic en el botón Mostrar Configuración de Estándares y se muestra una ventana con

	configuración		posibilidad de impresión

La Tabla 15 recoge el diseño de los casos de prueba necesarios para probar las funcionalidades de los escenarios de cada sección, descrita anteriormente. Seguidamente aparece una breve descripción de cada campo de la tabla.

- Id del escenario: Identificador del escenario que coincide con el identificador que aparece en la tabla de definición de escenarios.
- Escenario: Nombre del escenario que coincide con el nombre que aparece en la tabla de definición de escenarios.
- Acción del usuario: Acción que realiza el usuario sobre la interfaz visual que provoca una respuesta del sistema.
- Respuesta del sistema: Acción o grupo de acciones que debe realizar el sistema como respuesta a una acción del usuario.
- Resultado de la prueba: Si la acción del usuario y la respuesta del sistema coinciden con los esperados, el resultado es Satisfactorio. Si el sistema responde de forma inesperada, se detalla lo ocurrido.

Tabla 11 Diseño de los Casos de Prueba de Caja negra.

Id del escenario	Escenario	Acción del usuario	Respuesta del Sistema	Resultado de la Prueba
EC 1.1	Abrir carpeta de archivos	Seleccionar una carpeta y dar clic en Abrir	Cargar carpeta con archivos C# en la aplicación	Satisfactorio
EC 1.2	Abrir archivo	Seleccionar un archivo C# y dar clic en Abrir	Cargar el archivo C# en la aplicación	Satisfactorio
EC 1.3	Introducir código	Seleccionar Introducir Código del menú Archivo	Mostrar la ventana de introducción de código	Satisfactorio
EC 1.4	Cerrar todos	Accionar el botón Cerrar todos de la barra de herramientas	Cerrar todos los archivos y carpetas que se encuentren abiertos en la aplicación	Satisfactorio
EC 1.5	Salir	Escoger Salir del menú Archivo	Mostrar mensaje de confirmación	Satisfactorio

EC 1.6	Abrir Ventana Complejidad Ciclomática	Escoger un archivo cargado y hacer clic en el botón Complejidad Ciclomática de la barra de herramientas	Se muestra la ventana con el código y la jerarquía de clases de dicho código	Satisfactorio
EC 1.7:	Abrir Estándares de Codificación	Escoger un archivo cargado y hacer clic en el botón Estándares de Codificación de la barra de herramientas	Se muestra la ventana con el código	Satisfactorio
EC 1.8:	Abrir Parámetros del Código	Hacer clic en el botón Parámetros del Código	Se muestra la ventana	Satisfactorio
EC 1.9:	Pegar	Presionar las teclas Ctrl + V o hacer clic en Pegar del menú Edición	Se muestra la ventana Introducir Código con el texto del portapapeles de Windows	No se pega el código
EC 1.10:	Abrir Configuración de Estándares de Codificación	Hacer clic en la opción Estándares de Codificación del menú Configuración	Se muestra una ventana donde se puede configurar los estándares de codificación	Satisfactorio
EC 1.11:	Abrir ventana Opciones	Hacer clic en la opción Opciones del menú Configuración	Se muestra la ventana Opciones	Satisfactorio
EC 1.12:	Abrir la ventana Acerca de HAC	Hacer clic en el menú Ayuda y escoger Acerca de HAC	Se muestra la ventana con la información de HAC	Satisfactorio
EC 1.13:	Pruebas de Caja Blanca	Hacer clic en el menú Ayuda y escoger la opción Pruebas de Caja Blanca	Se muestra una ventana con información sobre las Pruebas de Caja Blanca	Satisfactorio
EC 2.1	Analizar los Estándares	Hacer clic en el botón Analizar	Se muestra un listado de incumplimientos agrupados por categoría	Satisfactorio
EC 2.2	Cerrar	Hacer clic en el botón	Se cierra la ventana	Satisfactorio.

		cerrar de la ventana	actual	
EC 3.1	Analizar Todos	Hacer clic en el botón analizar todos	Las funciones cargadas se les agrega un número que indica su complejidad ciclomática	Satisfactorio.
EC 3.2	Exportar Matriz	Hacer clic derecho en una función analizada y escoger la opción Exportar Matriz	Se muestra un cuadro de diálogo para escoger la carpeta donde se va a guardar el archivo de matriz y al dar aceptar se guarda correctamente	Satisfactorio.
EC 3.3	Exportar Matrices	Hacer clic en el botón Exportar Matrices del menú Acciones	Se muestra un cuadro de diálogo para escoger la carpeta donde se van a guardar los archivos de matrices y al dar aceptar se guardan correctamente	Se mostró el cuadro de diálogo pero no se exportaron las matrices
EC 4.1	Reporte Archivo	Hace clic en el botón Reporte Archivo	Se muestra un reporte con cada archivo y sus parámetros de código	Satisfactorio
EC 4.2	Exportar Reporte	Hacer clic en el botón Exportar	Se muestra un cuadro de diálogo para escoger la dirección donde se desea guardar, se da aceptar y se guarda correctamente	No se mostró el cuadro de diálogo
EC 4.3	Reporte Clases	Hacer clic en el botón Reporte Clases	Se muestra un reporte con cada clase y sus parámetros de código	No se mostró nada
EC 5.1	Abrir Configuración	Hacer clic en el botón Abrir Configuración	Muestra un cuadro de diálogo para seleccionar la configuración que se desea abrir	Satisfactorio
EC 5.2	Guardar Configuración	Hacer clic en el botón Guardar Configuración	Muestra un cuadro de diálogo para seleccionar la dirección donde guardar la configuración	Satisfactorio

			que se desea	
EC 5.3	Mostrar configuración	Hacer clic en el botón Mostrar configuración	Muestra una ventana con la configuración	Satisfactorio

La ejecución de los Casos de Prueba de Caja Negra develó algunos errores en la implementación de la interfaz visual principalmente. La mayoría debido a descuidos del programador o a errores mismos en la lógica de programación pero en general los resultados fueron satisfactorios. Finalmente se corrigieron dichos errores y se comprobó nuevamente el resultado.

Conclusiones

Durante el diseño y ejecución de las pruebas de Caja Blanca y Caja Negra a la aplicación, se encontraron algunos errores que a simple vista no hubieran sido fáciles identificar. Algunos se produjeron por las siguientes causas: descuido, falta de comprensión de la parte del problema que se soluciona, falta de revisión durante la codificación y falta de diseño de los algoritmos en pseudocódigo antes de la codificación.

Se recomienda limar algunos detalles en el diseño de la interfaz de usuario y en la implementación y validación de sus elementos. Mostrar mensajes de información, error o confirmación antes y después de cada acción importante, con vistas a facilitar el uso de la misma por parte del usuario. Además se deben revisar y perfeccionar algunos algoritmos que pudieran ser más sencillos y quizás más eficientes.

Los resultados de la aplicación de la Técnica del Camino Básico mediante el método manual y automatizado permiten llegar a la conclusión de que el uso de HAC es factible, por el tiempo que el usuario ahorra y por la confiabilidad de los resultados. Fue muy útil además, el uso de Grafos para visualizar los grafos, analizar los caminos de ejecución del programa, contar las regiones de los grafos y los nodos predicado, así como los nodos y aristas.

CONCLUSIONES

Durante el desarrollo del presente Trabajo de Diploma han sido cumplidos el objetivo y las tareas previstas, obteniéndose los siguientes resultados:

Antes de la implementación de la herramienta de este trabajo se hizo necesario realizar un estudio valorativo y crítico del diseño precedente y a partir del mismo hacer las modificaciones pertinentes en las clases con sus características y responsabilidades.

Para implementar las funcionalidades se estudiaron los contenidos teóricos referentes a la Técnica del Camino Básico, los Parámetros del código, los Estándares de Codificación y la sintaxis del lenguaje C#. Además, se escogieron la herramienta adecuada para el desarrollo, el lenguaje y la plataforma más ajustados.

Además se validaron las funcionalidades en el producto para comprobar que cumpliera con los requerimientos funcionales, lográndose una aplicación de fácil manejo, con una interfaz visual amigable y sencilla.

La ejecución de las tareas en el sistema obtenido ocurre de forma aceptable, incluso en las situaciones más críticas. Mediante el análisis de los resultados, se llega a la conclusión de que el uso de la herramienta permitirá facilitar el trabajo en los procesos de revisión de código a los productos software de la Facultad, aumentando la productividad y el ahorro de recursos.

RECOMENDACIONES

Los autores de este trabajo recomiendan las siguientes actividades para enriquecer la herramienta:

Implementar otros analizadores, para otros lenguajes de programación enriqueciendo la utilidad de la herramienta.

Aplicar tecnologías novedosas sobre la base de los controles personalizados en el perfeccionamiento de los elementos de la interfaz visual de la herramienta propuesta.

Una vez haya madurado el desarrollo de interfaces visuales sobre la plataforma Mono, comenzar una estrategia de migración que permita la portabilidad en sistemas operativos Linux.

Agregar nuevas funcionalidades que permitan completar las pruebas de Caja Blanca con la capacidad de diseñar y ejecutar casos de prueba.

Determinar nuevos parámetros del código fuente con el objetivo de que la herramienta alcance mayor aceptación y determinación e interpretación de nuevas métricas.

Integrar un componente para el dibujo de grafos, de forma que se integre a HAC y se puedan explotar aún más sus potencialidades.

Implementar las funcionalidades para la gestión de proyectos en el trabajo con la herramienta con la posibilidad de guardar configuraciones, últimos cambios, archivos y carpetas abiertos.

REFERENCIAS BIBLIOGRÁFICAS

1. McCabe, Thomas J. A Complexity Measure. s.l. : IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 1976.

2. *Student portfolios and software quality metrics in computer science education.* Patton, Arnold L. and McGill, Monica. 4, Peoria : Journal of Computing Science in College, 2006, Vol. 21. ISSN:1937-4771.

3. *A Metric Suite for Object Oriented Design.* Chidamber, Shyam R. and Kemerer, Chris F. 6, s.l. : IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 1994, Vol. 20.

4. Archer, Tom y Whitechapel, Andrew. *LA BIBLIA DE MICROSOFT VISUAL C++ .NET.* s.l. : ANAYA, 2003. 84-415-1487-9.

5. Mueller, John. *Visual C++ .Net.* s.l. : ANAYA MULTIMEDIA-ANAYA INTERACTIVA, 2002. ISBN: 978-84-415-1408-9.

6. Mitchell, Will David. *Java sin errores.* Madrid : McGraw-Hill, 2001. ISBN: 84-481-3107-1.

7. Zukowsky, John. *Java 2 J2SE 1.4.* Madrid : ANAYA, 2003. 84-415-1559-1.

8. Ecma International. *Standard ECMA-334 C# Language Specification.* s.l. : Ecma International, 2006.

9. *Programming for the New Plattform .* Richter, Jaffrey. s.l. : MSDN Magazine, 2000.

10. Microsoft Corporation . MSDN Microsoft Developer Network . *Visual Studio Developer Center.* [En línea] MIcrosoft Corporation. [Citado: 5 21, 2008.] http://msdn.microsoft.com/en-gb/vstudio/bb265237.aspx.

11. Archer, Tom. *A FONDO C#.* Redmond : McGraw-HILL/INTERAMERICANA DE ESPAÑA, S.A.U, 2001. ISBN: 84-481-3246-7.

12. Ver referencia 10.

13. Ver referencia 10.

14. Borland Software Corporation. C# Builder. *Borland C# Builder Developer Resources.* [En línea] Borland Software Corporation. [Citado: 5 21, 2008.] http://www.csharpbuilder.info/.

15. ic#code. ic#code. *SharpDevelop The Open Source Development Environment for .NET.* [En línea] ic#code. [Citado: 5 21, 2008.] http://www.icsharpcode.net/OpenSource/SD/.

16. MonoDevelop. MonoDevelop Free GNOME Development Environment. *Feature List.* [En línea] MediaWiki. [Citado: 5 21, 2008.] http://www.monodevelop.com/Feature_List.

17. Free Software Foundation. Free Software Foundation. *Licensing.* [En línea] Free Software Foundation. [Citado: 5 21, 2008.] http://www.fsf.org/licensing/.

18. Imsrika, Samuel. C# LEX Manual. [En línea] 5 12, 2005. [Citado: 5 21, 2008.] http://www.infosys.tuwien.ac.at/cuplex/lex.htm.

19. Per Cederberg. Grammatica Version Information. [En línea] 8 27, 2003. [Citado: 5 21, 2008.] http://grammatica.percederberg.net/doc/release/version.html.

20. metaspec. metaspec. *C# Parser.* [En línea] [Citado: 5 21, 2008.] http://www.csharpparser.com/csparser.php.

21. VIRT Laboratory. PAX script . *About paxScript.NET.* [En línea] 10 1, 2007. [Citado: 5 21, 2008.] http://www.paxscript.net/.

22. Kloeten, Omer Van. Omer Van Kloeten`s .NET Zen. *Designing The Lexical Analyzer.* [En línea] 7 13, 2004. [Citado: 5 21, 2008.] http://weblogs.asp.net/okloeten/pages/181974.aspx.

23. Ver referencia 1.

24. Ver referencia 3.

25. Ver referencia 2.

26. Ver referencia 3.

27. Arregui, Juan José Olmedilla. Revisión Sistemática de Métricas de Diseño Orientado a Objetos. Madrid : s.n., 2005.

28. Ver referencia 22.

29. Ver referencia 8.

30. Ferguson, Jeff, y otros. *La biblia de C#.* Madrid : GRUPO ANAYA S.A., 2003. ISBN: 84-415-1484-4.

31. Ver referencia 10.

32. Ver referencia 15.

33. Villalobos, Alejandro Rodríguez. Grafos - Manual de usuario. 2005.

34. Ver referencia 30.

35. Hommel, Scott. *Conveciones de código para el lenguaje de programación Java.* 2001.

36. Pressman, Roger S. *Ingeniería de Software. Un enfoque práctico.* s.l. : Mc Graw-Hill/Interamericana de España, S. A, 2002.

37. Ver referencia 36.

38. Ver referencia 36.

BIBLIOGRAFÍA

- *A Metric Suite for Object Oriented Design.* Chidamber, Shyam R. and Kemerer, Chris F. 6, s.l. : IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 1994, Vol. 20.
- Akif, Mohammad, y otros. *Java y XML.* Madrid : ANAYA, 2002. ISBN: 84-415-1364-5.
- *An Interview with Bjarne Stroustrup.* Buchanan, James. s.l. : Dr. Dobb´s Magazine, 3 27, 2008.
- ANSI. www.ansi.org. *Introducción.* [En línea] [Citado el: 19 de 5 de 2008.] http://www.ansi.org/about_ansi/introduction/introduction_sp.aspx?menuid=1.
- Archer, Tom y Whitechapel, Andrew. *LA BIBLIA DE MICROSOFT VISUAL C++ .NET.* s.l. : ANAYA, 2003. 84-415-1487-9.
- Archer, Tom. *A FONDO C#.* Redmond : McGraw-HILL/INTERAMERICANA DE ESPAÑA, S.A.U, 2001. ISBN: 84-481-3246-7.
- Arregui, Juan José Olmedilla. Revisión Sistemática de Métricas de Diseño Orientado a Objetos. Madrid : s.n., 2005.
- Ayala, Ramón Montero. *XML Iniciación y referencia.* s.l. : McGraw-Hill, 2001. 84-481-2894-1.
- *Bjarne Stroustrup on the Evolution of Languages (Interview).* Dierking, Howard. 5, s.l. : MSDN Magazine, 4 2008, Vol. 23.
- Borland Software Corporation. C# Builder. *Borland C# Builder Developer Resources.* [En línea] Borland Software Corporation. [Citado: 5 21, 2008.] http://www.csharpbuilder.info/.
- Borland Software Corporation. CodeGear from Borland. *Entornos Integrados de Desarrollo para Windows, Java y Web .* [En línea] Borland Software Corporation. [Citado: 5 21, 2008.] http://www.codegear.com/products/bds2006.
- *C++ Machinery.* Stepanov, Alexander and McJones, Paul. 2008. Elements of Programming.
- ECMA INTERNATIONAL. ecma INTERNATIONAL. *Standard ECMA-262.* [En línea] 12 1999. [Citado: 5 20, 2008.] http://www.ecma-international.org/publications/standards/Ecma-262.htm.
- Ecma International. *Standard ECMA-334 C# Language Specification.* s.l. : Ecma International, 2006.
- Ecma International. *Standard ECMA-335 Common Language Infrastructure (CLI).* s.l. : Ecma International, 2006.
- Ferguson, Jeff, y otros. *La biblia de C#.* Madrid : GRUPO ANAYA S.A., 2003. ISBN: 84-415-1484-4.
- Free Software Foundation. El Sistema Operativo GNU. [En línea] Free Software Foundation. [Citado el: 21 de 5 de 2008.] http://www.gnu.org/home.es.html.

- Free Software Foundation. Free Software Foundation. *Licensing.* [En línea] Free Software Foundation. [Citado: 5 21, 2008.] http://www.fsf.org/licensing/.
- Hommel, Scott. *Conveciones de código para el lenguaje de programación Java.* 2001.
- ic#code. ic#code. *SharpDevelop The Open Source Development Environment for .NET.* [En línea] ic#code. [Citado: 5 21, 2008.] http://www.icsharpcode.net/OpenSource/SD/.
- Imsrika, Samuel. C# LEX Manual. [En línea] 5 12, 2005. [Citado: 5 21, 2008.] http://www.infosys.tuwien.ac.at/cuplex/lex.htm.
- International Organization for Standarization. www.iso.org. *About ISO.* [En línea] [Citado: 5 19, 2008.] http://www.iso.org/iso/about.htm.
- Kloeten, Omer Van. Omer Van Kloeten`s .NET Zen. *Designing The Lexical Analyzer.* [En línea] 7 13, 2004. [Citado: 5 21, 2008.] http://weblogs.asp.net/okloeten/pages/181974.aspx.
- McCabe, Thomas J. A Complexity Measure. s.l. : IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 1976.
- metaspec. metaspec. *C# Parser.* [En línea] [Citado: 5 21, 2008.] http://www.csharpparser.com/csparser.php.
- Microsoft Corporation . MSDN Microsoft Developer Network . *Visual Studio Developer Center.* [En línea] MIcrosoft Corporation. [Citado: 5 21, 2008.] http://msdn.microsoft.com/en-gb/vstudio/bb265237.aspx.
- Microsoft Corporation. CodePlex. *SharpDevelop.* [En línea] Microsoft Corporation. [Citado: 5 21, 2008.] http://www.codeplex.com/SharpDevelop.
- Microsoft Corporation. MSDN Microsoft Developer Network. *.NET Framework Developer Center.* [En línea] Microsoft Corporation. [Citado: 5 21, 2008.] http://msdn.microsoft.com/en-gb/netframework/default.aspx.
- Mitchell, Will David. *Java sin errores.* Madrid : McGraw-Hill, 2001. ISBN: 84-481-3107-1.
- MonoDevelop. MonoDevelop Free GNOME Development Environment. *Documentation.* [En línea] MediaWiki. [Citado: 5 21, 2008.] http://www.monodevelop.com/Documentation.
- MonoDevelop. MonoDevelop Free GNOME Development Environment. *Feature List.* [En línea] MediaWiki. [Citado: 5 21, 2008.] http://www.monodevelop.com/Feature_List.
- Mueller, John. *Visual C++ .Net.* s.l. : ANAYA MULTIMEDIA-ANAYA INTERACTIVA, 2002. ISBN: 978-84-415-1408-9.
- Per Cederberg. Grammatica Version Information. [En línea] 8 27, 2003. [Citado: 5 21, 2008.] http://grammatica.percederberg.net/doc/release/version.html.

- Pressman, Roger S. *Ingeniería de Software. Un enfoque práctico.* s.l. : Mc Graw-Hill/Interamericana de España, S. A, 2002.
- *Programming for the New Plattform .* Richter, Jaffrey. s.l. : MSDN Magazine, 2000.
- Schildt, Herbert. *C++ Guía de autoenseñanza.* [trad.] CARLOS CERVIGON RÜCKAUER. MADRID : McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S.A.U, 1995. ISBN: 84-481-3203-3 .
- Stepanov, Alex. Interview of Alex Stepanov by Yuyong Zhao. [interv.] Yuyong Zhao. *Chinese Popular Computer Weekly.* 2 28, 2005. p. 3.
- Stroustrup, Bjarne. www.research.att.com. [En línea] 2 5, 2008. [Citado: 5 21, 2008.] http://www.research.att.com/~bs/bs_faq.html#oop.
- *Student portfolios and software quality metrics in computer science education.* Patton, Arnold L. and McGill, Monica. 4, Peoria : Journal of Computing Science in College, 2006, Vol. 21. ISSN:1937-4771.
- SUN Microsystems. The Java EE 5 Tutorial. *Java SUN Microsystems.* [En línea] [Citado: 5 20, 2008.] http://java.sun.com/javaee/5/docs/tutorial/doc/.
- Sun Microsystems. The Java Tutorials . *Learning the Java Language.* [En línea] [Citado: 5 20, 2008.] http://java.sun.com/docs/books/tutorial/java/nutsandbolts/index.html.
- *The Birth of Web Services.* Skonnard, Aaron. s.l. : MSDN Magazine, 2002.
- UP to DOWN. *Descargar SharpDevelop 2.2.1.2648.* [En línea] Media Ingea SL. [Citado el: 21 de 5 de 2008.] http://sharpdevelop.uptodown.com/.
- Villalobos, Alejandro Rodríguez. Grafos - Manual de usuario. 2005.
- VIRT Laboratory. PAX script . *About paxScript.NET.* [En línea] 10 1, 2007. [Citado: 5 21, 2008.] http://www.paxscript.net/.
- Zukowsky, John. *Java 2 J2SE 1.4.* Madrid : ANAYA, 2003. 84-415-1559-1.
- —. MSDN Microsoft Developer Network. *MSDN Library (Español).* [En línea] Microsoft Corporation. [Citado el: 21 de 5 de 2008.] http://msdn.microsoft.com/es-es/library/default.aspx.
- —. www.ansi.org. *ANSI: una visión histórica.* [En línea] [Citado el: 19 de 5 de 2008.] http://www.ansi.org/about_ansi/introduction/history_sp.aspx?menuid=1.

GLOSARIO DE TÉRMINOS

- Bytecode: Lenguaje en el cual toda la información es representada por secuencias de ceros y unos.
- CodeDOM: Proporciona un modelo estándar para conjuntar objetos XML, a la vez que un interfaz estándar para manipular y recorrer los objetos y sus correlaciones.
- Compilación condicional: Da la posibilidad de seleccionar las secciones concretas del código fuente del programa que se desee compilar.
- Delegados: Un delegado es un tipo de datos que hace referencia a un método y se comporta como tal.
- Ejecución multi-hilo: Ejecución simultánea de más de un flujo de programa.
- For: Bucle que ejecuta un grupo de instrucciones repetidamente hasta que no se cumpla una determinada condición.
- Foreach: Instrucción que se utiliza para recorrer en iteración una colección de elementos y obtener información sin modificarla.
- If: Estructura donde se ejecuta una instrucción u otra en dependencia del valor de una expresión condicional.
- Inline: Son funciones que se marcan con la palabra inline indicando al compilador que puede o no sustituir el código de su invocación por el cuerpo de su implementación en todo el programa logrando mayor velocidad de ejecución pero también mayor volumen de código fuente.
- Int: En C# representa un tipo entero de 32 bits con signo.
- IntelliSense: Novedosa funcionalidad que permite el completado automático de palabras en dependencia del contexto así como la guiar en el completamiento de la lista de parámetros de las funciones y brindar información completa de la declaración de los identificadores.
- Kernel: Parte principal de un sistema operativo, encargado del manejo de los dispositivos, la gestión de la memoria, del acceso a disco y en general de casi todas las operaciones del sistema que permanecen invisibles para el usuario. Pudiera considerarse además como “esqueleto principal de un programa”.
- Long: En C# representa un tipo entero de 64 bits con signo.
- Refactorización: Técnica de la Ingeniería de Software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.
- Reflexión: Al convertir el código de alto nivel en código de bajo nivel se conservan algunas características de la implementación inicial.

- Smart Device: Aparatos pequeños, con algunas capacidades de procesamiento y memoria, con conexión permanente o intermitente a una red como teléfonos celulares y PDAs.
- Subversion: Sistema de control de versiones libre.
- Switch: Estructura donde se ejecuta una instrucción dentro de varias posibles en dependencia del valor de una expresión integral.
- Tab: Espacio formado por varios espacios sencillos cuya cantidad está determinada en la configuración del sistema operativo o en la configuración del editor de texto y se introduce en el texto al presionar la tecla “Tab” del teclado.
- Token: Palabra correspondiente a una gramática que posee reglas en dependencia del tipo que define dentro de dicha gramática.