UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 4

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

TÍTULO: ¨Propuesta de una herramienta de apoyo a la enseñanza de la asignatura de Introducción a la programación en lenguaje C#."

AUTORES: Efren Olamendiz Miqueli.

Jorge Manuel Ramy Fleitas.

TUTORES: Luis Felipe Díaz Barrios.

CO-TUTOR: Leyanis Pompa Arcía.

Yudanis Gago Martínez.

DECLARACIÓN DE AUTORÍA

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

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

Jorge Manuel Ramy. Efren Olamendiz Miqueli.

_____ _____

Firma del autor. Firma del autor.

Luis Felipe Díaz Barrios. Lesyanis Pompa Arias. Yudanis Gago Martínez.

_____ _____ _____

Firma del tutor. Firma del co-tutor. Firma del co-tutor.

Dedicatoria.

¨A nuestra familia por el apoyo incondicional y el cariño brindado, a la Universidad de Ciencias Informáticas por formarnos como profesionales así como a los profesores que lo hicieron posible.¨

Agradecimientos.

A la profesora Lesyanis Pompa y a Alejandro León por su apoyo incondicional en el desarrollo de la investigación, a mis padres por estar siempre presente, a mi hermana, a mis primos Laíz, Ayola, Lázaro por ayudarme cada vez que lo he necesitado, a Jorge, Omar, Medardo, Dunia, Tatá, Ifrain, Fernando, Roberto, a los profes: Sasha, Lara, Greisy, Surelis, Luis Felipe, Thaymí, Raul, Lisandra, Yanko, Jhommy, Jose por su paciencia y dedicación, a mis compañeros de clase y amigos por estar cerca durante este período y a todas aquellas personas que aportaron un granito de arena en esta investigación.

Efren.

Dedicado a mi familia, de modo muy particular a mi madre Carmen, mi padre Jorge Esteban, mi hermana Yamilet, mi cuñado Miguel, a mis dos sobrinos lindos Angelica Maria y Miguel Angel. A mis grandes amigos que son como mi 2da familia. Guillermo Jose (el Metra), Javier Garcia (El Afgano), Yoanny Quintero (El Solapa) y en general a mis amistades y personas que me quieren.

Ramy.

Resumen.

El trabajo de investigación, muestra a través de una herramienta visual, la ejecución en modo gráfico de un programa en lenguaje C#, con la finalidad de apoyar el proceso de aprendizaje de una parte de los contenidos impartidos en la asignatura de IP (Introducción a la Programación), lo que sin dudas, es un tema muy importante para los estudiantes que cursan la carrera de informática en su primer año. La misma permite observar cómo funcionan los métodos, variables, arreglos y operaciones que se definan en el programa, facilitando que el estudiante siga las instrucciones del mismo paso a paso.

Para el desarrollo de la herramienta se siguieron los pasos que propone la metodología **XP** en sus distintas fases, obteniéndose como resultado final un producto funcional que satisface las necesidades del cliente, además de la documentación obtenida a lo largo de la investigación.

A partir de las entrevistas realizadas a especialistas se considera que la propuesta muestra amplias posibilidades para la formación de los estudiantes en la asignatura de IP (Introducción a la Programación), así como su contribución al quehacer científico de la **UCI** (Universidad de las Ciencias Informáticas).

PALABRAS CLAVES

Compilador, Fases de Compilación, Lexer, Parser, Análisis Lexicográfico, Análisis Sintáctico, Análisis Semántico, Árbol de Sintaxis Abstracta.

Índice

Índice de Figuras.

Índice de tablas.

Introducción.

Las últimas décadas del siglo pasado y los inicios del siglo XXI han sido testigos de una nueva era: la Era de la Informática. La misma hoy en día desempeña un papel fundamental a nivel mundial, por lo que su impacto ha propiciado cambios significativos en todos los ámbitos de la sociedad.

Cada nueva tecnología en la actualidad está estrechamente relacionada con los sistemas informáticos, estos sistemas presentan tres características que garantizan el uso y la eficacia de dichos sistemas, las mismas son: el ahorro de mano de obra, la eficiencia y la simplicidad. El enorme auge en la informática y de los lenguajes de programación exige, cada vez más, de mayores conocimientos y de profesionales más capacitados, de ahí la importancia de la enseñanza de la programación.

Estudios realizados por investigadores del **Instituto Científico Weizmann** en el año 2002, acerca de la enseñanza temprana de la programación, han demostrado que entre las principales causas que dificultan la comprensión de la misma se encuentra el trabajo con estructuras repetitivas y condicionales, conceptos fundamentales de la programación a los cuales los estudiantes no se han enfrentado con anterioridad. (1)

Existe preocupación dentro de la comunidad docente a nivel internacional sobre la forma en que se enseña la programación en los primeros momentos de las carreras relacionadas con la informática. Constantemente aparecen discusiones sobre si uno u otro lenguaje es el más apropiado o sobre cuál es la alternativa pedagógica que brinda mejores resultados. Pero, entre todas las opciones, existen dos ideas fundamentales que son universalmente compartidas a la hora de enseñar: es necesaria la utilización de ejemplos complejos donde se adquieran habilidades, pues la complejidad en éstos ayuda a los alumnos a comprender el verdadero valor del código dado que llevan la impronta del desarrollador que los ha creado, así como la experiencia de diseño de esta persona. (2) Por lo tanto, el estudio de éstos constituye la primera fuente de experiencia de programación de la que los alumnos se nutren. Pero no es suficiente, al igual que solo se aprende a jugar fútbol practicando, las habilidades de un programador solo se adquieren programando. En tal sentido es necesario fomentar actividades en la que los alumnos tengan la oportunidad de poner en práctica sus habilidades como programadores.

Por la complejidad del tema, a nivel internacional se cuenta con vasta información destinada a facilitar la comprensión de la programación. Sin embargo, a pesar de la gran

cantidad de manuales y herramientas con las que se dispone, no hay certeza de la existencia de un método pedagógico que sea totalmente eficaz, aunque una buena práctica consiste en visualizar mediante escenarios algunos conceptos de la programación. (1)

Cuba, país bloqueado con escasas posibilidades dentro del amplio desarrollo de las tecnologías, no está ajena al desarrollo tecnológico existente pues, a pesar de todas las dificultades que afronta para adquirir medios tecnológicos y avanzar en este mundo, ha reconocido la necesidad de lograr un desarrollo en este campo; es por eso que se hace necesario la utilización de herramientas de soporte a la asignatura de introducción a la programación.

La Universidad de Ciencias Informáticas (**UCI**) no está al margen de esta problemática. Los estudiantes recién incorporados al centro presentan las dificultades expuestas anteriormente. De acuerdo a los resultados de la encuesta **(Anexo 1),** aplicada a estudiantes de primer año de la carrera, uno de los mejores ejercicios en la etapa en que se está cursando la asignatura de Introducción a la Programación consiste en tracear el código fuente, ejercicio en el cual a medida que se avanza en los diferentes temas de programación se vuelve mucho más complejo y en ocasiones es tan trabajoso que tiende a provocar errores en la respuesta. Cabe destacar que dentro de la asignatura de Programación se imparten varios lenguajes, en dependencia del plan de estudio y la Facultad a la que pertenece el estudiante. Los mismos son: **C++**, **Java** y **C#**. Terminada la primera etapa del nuevo modelo de formación los estudiantes reciben la asignatura de Programación 5 donde se aprecia una migración al lenguaje **C#** y comienzan a utilizar el **Microsoft Visual Studio** con el objetivo de ahorrar tiempo y evitar posibles errores en la prueba de nivel de programación, razón por la cual se le ofrece mayor importancia a este lenguaje.

Teniendo en cuenta lo antes expuesto y luego de analizado el diagnóstico de las diferentes técnicas de investigación aplicadas se puede formular como **problema científico**: ¿Cómo contribuir al proceso de comprensión de los conceptos básicos de la asignatura de Introducción a la Programación (IP).?

Objeto de estudio: Técnicas para la enseñanza de la programación.

Campo de acción: Representación visual del traceo en la asignatura de IP utilizando como lenguaje C#.

Objetivo General: Desarrollar una herramienta de apoyo a la enseñanza de la asignatura de Introducción a la programación en lenguaje C# que permita la representación visual de las operaciones aritméticas, condicionales y estructuras repetitivas.

De lo antes expuesto se derivan los **objetivos específicos** expresados de la siguiente forma:

- Investigar el estudio de las principales técnicas y herramientas de la programación en el mundo y en Cuba.
- Definir las funcionalidades a implementar.
- Desarrollar el análisis, diseño, implementación.
- Validar la propuesta de solución.

Para dar solución al problema se plantea la siguiente **idea a defender**: La aplicación correcta de la herramienta de soporte a la asignatura de Introducción a la Programación (IP), posibilitará el mejoramiento de la enseñanza de esta disciplina en la universidad.

Tareas investigativas a desarrollar:

- Entrevista con profesores del Departamento Central de programación para definir las principales dificultades que presentan los estudiantes en la asignatura de IP.
- Estudio de las referencias en cuanto al desarrollo de la enseñanza de la programación en el mundo en general y Cuba en particular.
- Sistematización de las principales tendencias que existen actualmente en Cuba y el mundo, acerca de la enseñanza de la programación y la utilización de herramientas de soporte.
- Análisis del estado actual de la preparación de los estudiantes de la UCI en la asignatura de programación.
- Selección de las herramientas y tecnologías adecuadas para el desarrollo de la aplicación.
- Estudio y selección de las metodologías de desarrollo a utilizar.
- Realización del análisis y diseño de la aplicación.
- Implementación de las funcionalidades de la aplicación.
- Evaluación de la solución obtenida mediante pruebas unitarias y de aceptación.

Población y muestra: La población y muestra fue seleccionada de la Universidad de las Ciencias Informáticas.

Población: Estudiantes de la carrera de ingeniería informática de la Universidad de las Ciencias Informáticas, procedentes de todas las provincias y municipios de Cuba conformando un total de diez mil estudiantes.

Muestra: La muestra intencional no probabilística la constituyen 200 estudiantes del primer año, tomado de la facultad 3 y 4. Dichos estudiantes pertenecen a la carrera de informática y proceden de todas las provincias de Cuba, de ellos 150 varones y 50 hembras.

Para darle cumplimiento a las tareas anteriormente expuestas es necesario utilizar los siguientes métodos de investigación:

Métodos teóricos:

- **Analítico – sintético**: Dada la necesidad humana de descomponer e integrar el conocimiento mentalmente, permitió en esta investigación, el estudio de los diferentes factores en su relativa independencia, así como descubrir las relaciones existentes entre un factor y otro y la interacción que se establece entre ellos, constituyendo este método una unidad dialéctica en la actividad científica.
- **Histórico – lógico:** Se utiliza con el fin de establecer las tendencias, concepciones y teorías que permitan conocer el objeto de estudio en su devenir histórico.
- **El análisis documental:** Facilitó el análisis y la sistematización del tema a partir de los documentos de autores nacionales y extranjeros.
- **Estudio documental:** Se emplea para la revisión y estudio de documentos normativos oficiales y de política educacional con respecto a la enseñanza de la programación en centros de educación superior.

Métodos empíricos:

- **Entrevista y encuesta:** Permitieron la exploración y constatación del estado actual de la enseñanza de la programación en estudiantes de primer año de la **UCI**.
- **Criterios de especialista:** Se utiliza con el fin de realizar una valoración parcial de la propuesta.

Significación práctica: Se pone a disposición de la universidad una herramienta de representación de conceptos básicos de programación en lenguaje **C#**; en tanto la misma es flexible y contribuye al perfeccionamiento del proceso de enseñanza-aprendizaje en la esfera de la programación, teniendo en cuenta las actuales exigencias del modelo educativo de las universidades.

Estructura por capítulos:

La tesis está estructurada en Introducción, tres capítulos de desarrollo, conclusiones y recomendaciones.

Capítulo 1: Fundamentación Teórica: En este capítulo se analiza el estado del arte, se abordan elementos teóricos que sirven de base para la realización de esta investigación, como son los conceptos básicos más importantes, se describen las fases del proceso de compilación así como algunos sistemas similares existentes vinculados al campo de acción, descripción y selección de las herramientas a utilizar y las metodologías utilizadas.

Capítulo 2: Características del sistema: Contiene un estudio sobre los patrones de diseño usados y los estándares de codificación. Se realiza una descripción detallada del flujo de procesos presentes en el negocio y de los procesos que serán objeto de automatización, concluyendo con la propuesta del sistema a desarrollar.

Capítulo3. Descripción de la propuesta de solución: En este capítulo se describen y detallan las historias de usuario que representan las funciones y propiedades que el sistema debe cumplir, así como el tiempo estimado por iteraciones para dar cumplimiento a cada una de ellas.

Capítulo 4. Construcción y validación de la propuesta de solución. Se representan los artefactos que sirven de fundamento al proceso de desarrollo, se construyen las tarjetas CRC (Clase, Responsabilidad y Colaboración) que forman parte de la fase de diseño de la metodología seleccionada y se recopilan los resultados de las pruebas aplicadas a la solución final, con el fin de validar y comprobar que la herramienta obtenida funciona y cumple con las expectativas del usuario.

CAPÍTULO 1 ¨ FUNDAMENTACIÓN TEÓRICA¨.

1.1 Introducción.

En este capítulo se hace un análisis sobre el proceso de compilación, se abordan temas relacionados con el problema científico, así como algunos conceptos que ayuden a su mejor comprensión. Se enfatiza lo respectivo a las soluciones existentes que brindan una respuesta al problema científico así como el análisis de las herramientas existentes vinculadas al campo de acción de mayor relevancia y se definen las tecnologías y herramientas que se van a utilizar.

1.2 Proceso de compilación.

La forma de comunicación de los seres humanos es muy diferente a la forma en que dos computadores intercambian información entre ellas, esto sucede porque los seres humanos utilizan lenguajes basados en múltiples símbolos y bastante complejos en el cual intervienen disímiles componentes de la lengua materna (emisor-receptor-codificador-decodificador-ruidos-canales y otros que influyen en la comunicación, su cantidad y su calidad). La computadora, aunque utiliza la mayoría de estos componentes del lenguaje, solamente es un medio tecnológico que utiliza en esencia el lenguaje binario: ceros y unos, dejando fuera componentes de carácter subjetivo que tienen que ver con el hombre: sus intereses, motivaciones, necesidades, sentimientos que influyen en la comunicación y en la gama de su interpretación de los objetos y fenómenos. (3)

1.2.1 Conceptos asociados al campo de acción.

Se denomina alfabeto (∑) a un conjunto arbitrario, finito y no vacío, de símbolos. Entre los alfabetos más comunes podemos mencionar el alfabeto binario {0,1}, el alfabeto latino {a, b, c, d, e, f, g}, etc.

Una cadena vacía no es más que una cadena sin elementos y se denota ε.

Una **gramática** es un **cuádruplo** G= {N, ∑, P, S} donde:

N: Conjunto finito no vacío de símbolos no terminales.

∑: Conjunto finito no vacío, disjunto de N, de símbolos terminales.

P: Conjunto de reglas de producción de la forma:

(N U T)*N (NUT)*→(NUT)*

S: Axioma o símbolo distinguido (SϵN) (3)

Léxico (Vocabulario) a un conjunto de palabras que forman parte de un lenguaje específico. (3)

Sintaxis a un conjunto de reglas necesarias para construir frases correctas en un lenguaje. (3)

Semántica al significado de frases generadas por la sintaxis y el léxico. (3)

Se denomina **Token** o **Lexema** a la unidad básica de un compilador, que puede representar un símbolo o conjunto de símbolos con un significado semántico. (3)

1.3 Fases del proceso de compilación.

El proceso de compilación está dividido en 6 fases, en la figura representada a continuación se delimitan y detallan cada una de ellas.

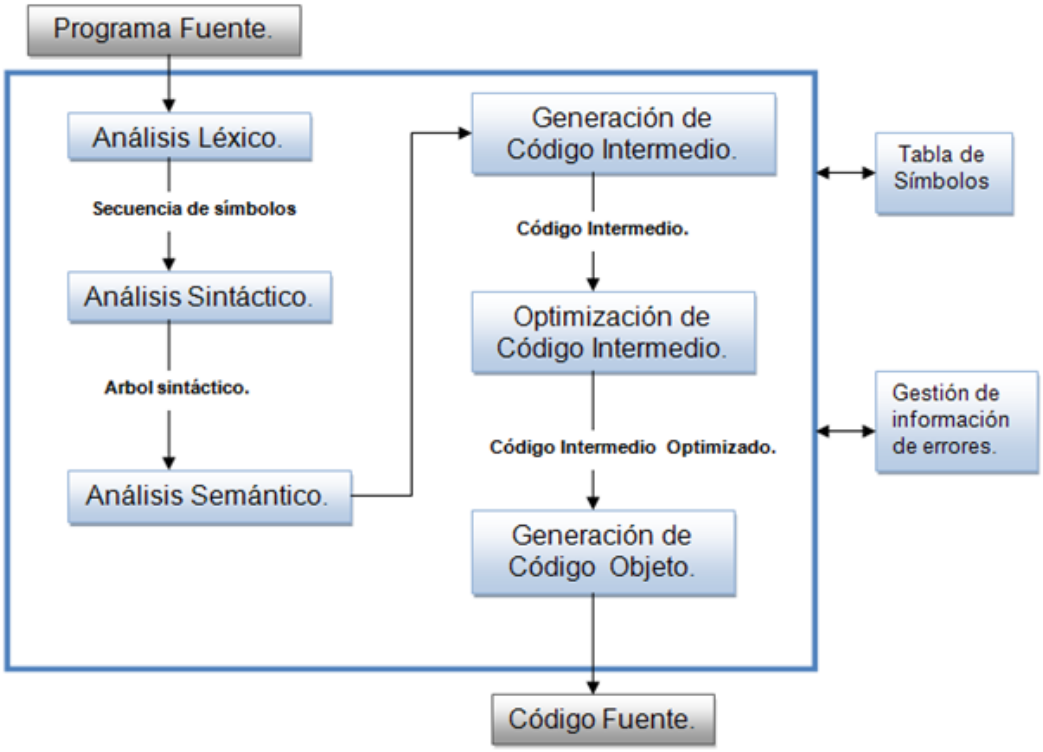

Figura 1 Fases del proceso de compilación.

1.3.1 Análisis Lexicológico.

En la fase de análisis léxico se leen los caracteres del programa fuente y se agrupan en cadenas que representan los componentes léxicos. Cada componente léxico es una secuencia de caracteres lógicamente coherente relativa a un identificador, estos pueden ser una palabra reservada, un operador o un carácter de puntuación. A la secuencia de caracteres que representa un componente léxico se le llama lexema (o con su nombre en inglés token). En el caso de los identificadores creados por el programador no solo se genera un componente léxico, sino que se genera otro lexema en la tabla de símbolos. (3)

1.3.2 Análisis sintáctico.

El análisis sintáctico es un proceso en el cual se examina la secuencia de tokens para determinar si el orden de la secuencia es correcto o no de acuerdo con ciertas reglas de la definición sintáctica del lenguaje.

La entrada al analizador sintáctico o parser es la secuencia de los tokens generados por el scanner. El parser analiza solamente el primer componente de cada token Si se logra reconocer la cadena completa según las reglas gramaticales se puede afirmar que la entrada está sintácticamente correcta. (3)

1.3.3 Análisis semántico.

En la fase de análisis semántico el compilador adiciona información al árbol de sintaxis abstracta generado en la fase de análisis sintáctico. Esta operación está relacionada con el segundo componente de la tupla devuelta en el análisis léxico (<tipo del token, info>). Al comprobar la validez semántica de un programa se pudiera decir que se realiza un reconocimiento sintáctico-semántico, pero los errores semánticos no pueden ser detectados por el analizador sintáctico, puesto que se relacionan con interdependencias entre las diferentes partes de un programa que no son reflejadas en un análisis gramatical. Es por ello que el análisis semántico se centra en encontrar instrucciones que tengan la estructura sintáctica correcta, pero que no tengan significado para la operación implicada. (3)

1.3.4 Generación de código intermedio.

Un compilador puede generar una o varias representaciones explícitas intermedias del código fuente. Dicha representación puede servir para la realización de un análisis semántico del código fuente o para la optimización del código. Las formas intermedias

deben tener dos características muy importantes: Deben ser fácil de producir y fácil de traducir al programa objeto. (3)

1.3.5 Optimización del código intermedio.

La fase de optimización se encarga de transformar el código intermedio en un nuevo código de función equivalente pero de menor tamaño o de menor tiempo de ejecución. Algunas de las transformaciones que puede llevar a cabo la fase de optimización son:

- Eliminar el cálculo de expresiones cuyo valor no se usa.
- Fundir en uno el cálculo repetido de la misma expresión.
- Sacar de los lazos las expresiones cuyo valor no cambia en ellos.
- Reducir el uso de memoria local reutilizando el espacio de una variable muerta. (3)

1.3.6 Generación del código objeto.

La fase de generación de código objeto se encarga de generar el programa nativo usando el juego de instrucciones específico de la máquina o CPU objeto, y el formato para archivos ejecutables del sistema operativo. Entre otras cosas, también se le asignan direcciones definitivas a las rutinas y variables que componen el programa. (3)

1.4 Herramientas de soporte a la enseñanza y aprendizaje de la programación.

Actualmente a nivel mundial se intensifica el uso de herramientas que brinden soporte al proceso docente educativo en la rama de la programación. Algunos autores argumentan que la programación no es difícil ni misteriosa, ya que si se pueden escribir instrucciones para conducir a alguna persona a su casa, entonces también se podrá programar. Que lo difícil de la programación es identificar todos los pequeños problemas que forman el problema mayor que se intenta resolver, y que para lograr que una computadora entienda el algoritmo a realizar, se debe especificar paso a paso lo que debe hacer. (4)

Los entornos de desarrollo para el aprendizaje son un concepto relativamente nuevo, sin embargo, es una práctica que se ha desarrollado desde hace varios años. Con base en un estudio realizado a las características de distintas herramientas de apoyo a la enseñanza de programación por la Universidad Complutense de Madrid, se pueden agrupar estas herramientas de la siguiente forma:

1) Sistemas con entorno de desarrollo incorporado: Son sistemas de enseñanza que incluyen un pequeño entorno de desarrollo que son menos complejos que los comerciales. En este grupo se encuentra **BlueJ**, el mismo proporciona un entorno de desarrollo altamente interactivo que promueve la exploración y la experimentación.

2) Entornos basados en ejemplos: Incluyen aquellos sistemas de enseñanza que se apoyan en la construcción de bases de ejemplos de programación, desarrollo de mecanismos de acceso y selección. En este grupo se puede mencionar a **Jeliot** que presenta al estudiante diversos niveles de aprendizaje mediante ejemplos listos para ser utilizados.

3) Entornos basados en visualización y animación: Es uno de los métodos de enseñanza de programación más explorado. Fundamenta su método en el uso de visualización y animación de algoritmos, logrando representaciones esquemáticas de los programas, permitiendo al alumno una fácil comprensión. Aquí, puede ser mencionado **Jeliot.**

4) Entornos basados en simulación: Éstos son otro subconjunto, los cuales se centran en entornos de simulación. Este tipo de entornos no se detiene a visualizar o animar la ejecución de un programa, sino va más lejos, simulando un mundo virtual en el cual los habitantes se comportan de acuerdo a las instrucciones del programa. El entorno más conocido de este tipo es **Alice**. (5)

Basado en lo anteriormente expuesto el sistema a desarrollar se clasifica como entorno basado en visualización y animación.

1.5 Tendencias y tecnologías actuales.

Mientras un número de herramientas útiles han sido desarrolladas con el objetivo de apoyar el proceso de aprendizaje de la programación, existe un área considerable para la mejora, sobre todo en los temas iniciales que se imparten en la programación, ya que la mayoría de las aplicaciones hacen hincapié en temas más complejos. (6) El estudiante inexperto puede beneficiarse con el apoyo de herramientas que aborden los temas básicos de la programación de forma visual mediante escenarios. Estas herramientas están basadas en los conocimientos de los que se dispone, las mismas utilizan animaciones para ayudar al estudiante a visualizar los resultados del código de ejemplo.

En la actualidad las herramientas disponibles son diseñadas para mostrar visualmente cómo se lleva a cabo el flujo del proceso de programación y consolidar así los conocimientos en esta rama. Cada herramienta de enseñanza es diseñada para una audiencia específica.

A continuación se describen algunas de las tendencias respecto a las tecnologías actuales más usadas y que es posible utilizar para dar solución al problema planteado con anterioridad, teniendo en cuenta las necesidades existentes y el entorno donde se aplicará el sistema a desarrollar.

Dentro de las herramientas actuales se pueden encontrar: **Bluej**, **Jeliot3**, **Alice** y **Scratch.**

BlueJ

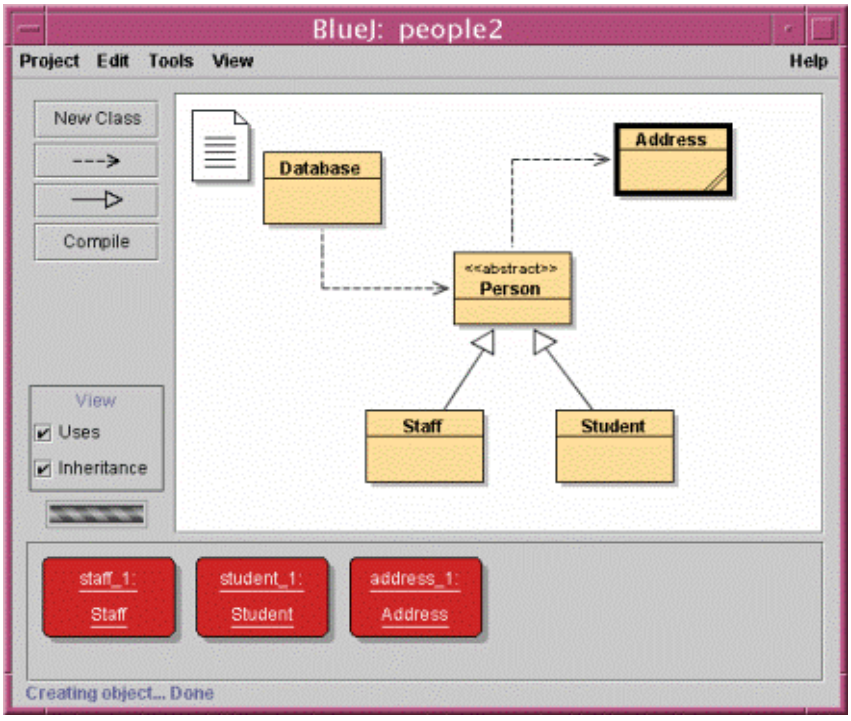

Figura 2 Interfaz gráfica de la herramienta Bluej.

Es un entorno de desarrollo integrado para el lenguaje de programación Java, creado principalmente para ser usado en sistemas educacionales. Fue creado con el objetivo de apoyar el aprendizaje y la enseñanza de la programación, su diseño se diferencia de otros entornos de desarrollo como consecuencia de ello. Los conceptos orientados a objetos (clases, objetos, la comunicación a través de llamadas a los métodos) están representados visualmente. (7)

Esta herramienta proporciona un entorno de desarrollo altamente interactivo que promueve la exploración y la experimentación. No ayuda al estudiante sobre la interpretación de los errores cometidos, ni brinda posibles soluciones.

Jeliot3

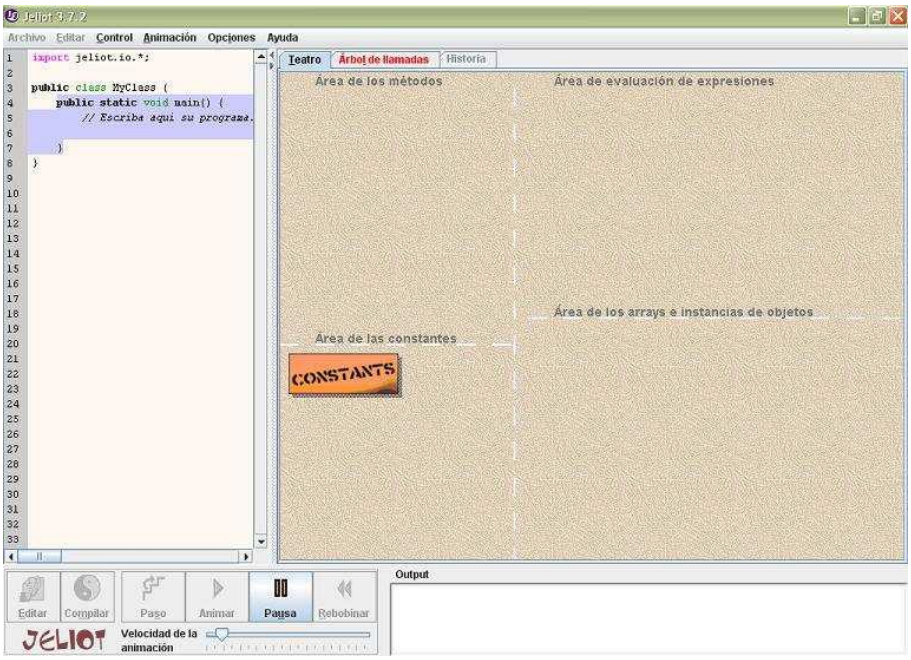

Figura 3 Interfaz gráfica de la herramienta Jelio3.

Es una herramienta que permite visualizar cómo un programa en Java es interpretado. Muestra el funcionamiento en una pantalla como una animación continua, que permite al estudiante seguir paso a paso la creación de variables, las llamadas a los métodos. Además, se ejecuta en prácticamente cualquier plataforma, incluyendo Windows, Linux, Mac, el único requisito es tener **Java Runtime Environment (JRE)** instalado en su sistema. (8)

Jeliot3 se puede utilizar para enseñar a estudiantes que se estén iniciando en la asignatura de programación ya que permite visualizar los conceptos de la misma, proporcionando a los estudiantes un modelo concreto de la ejecución del programa.

Alice

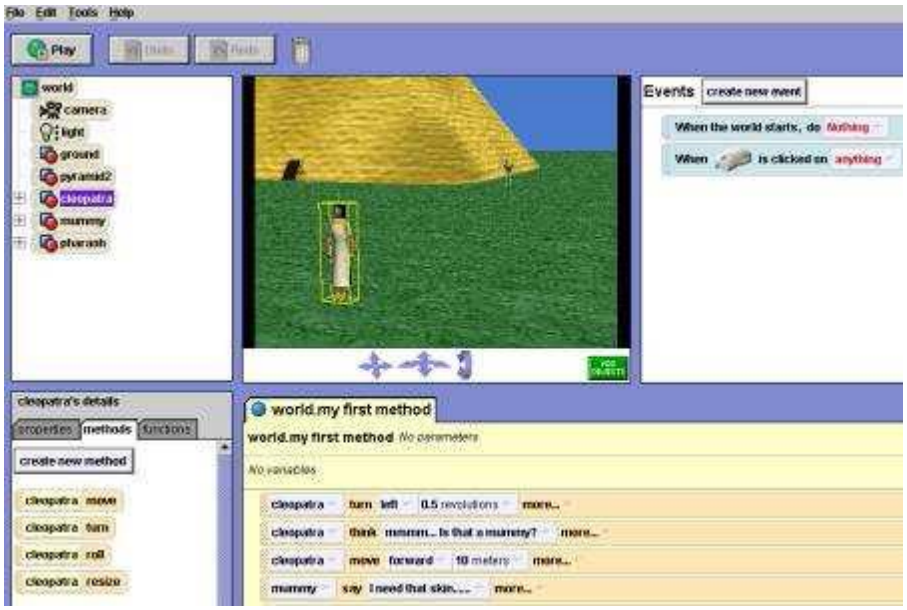

Figura 4 Interfaz gráfica de la herramienta Alice.

Alice fue desarrollado inicialmente como un sistema de prototipado rápido, después se propuso utilizar para enseñar gráficos **3D** y por último para enseñar programación. **Alice** consiste en un mundo virtual al que el alumno puede agregar objetos animados y de esta forma se va familiarizando con los conceptos principales de la programación orientada a objetos. (9)

Permite a los estudiantes ver cómo son ejecutados sus programas de forma gráfica, lo que les ayuda a comprender fácilmente la relación entre la programación de las declaraciones y el comportamiento de los objetos. Mediante la manipulación de los objetos en su mundo virtual los estudiantes adquieren experiencia con todas las estructuras de la programación.

Scratch

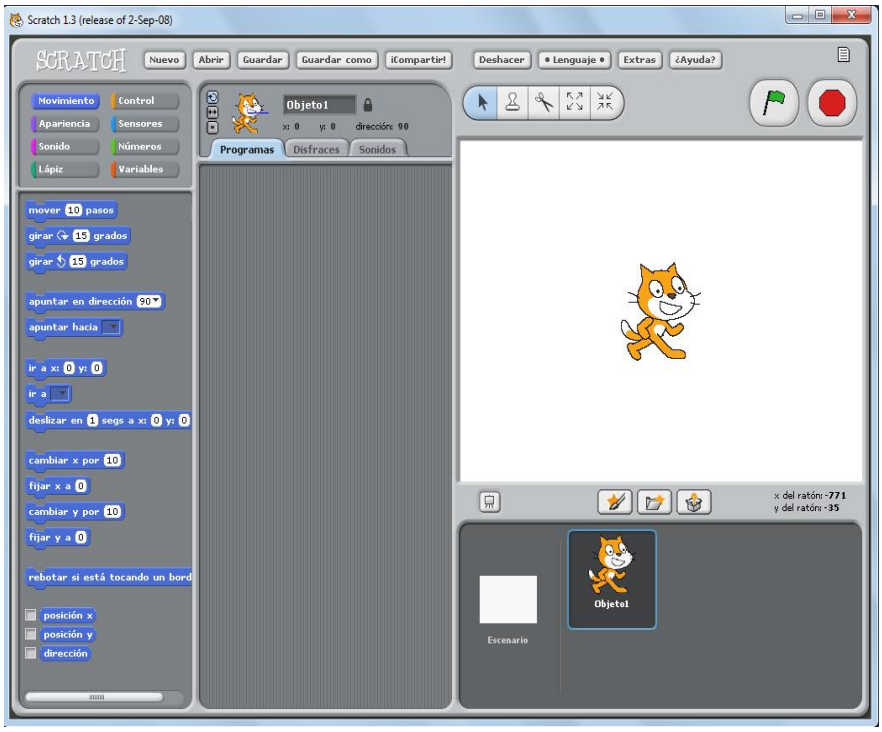

Figura 5 Interfaz gráfica del Scratch

Scratch es una herramienta gratuita para **Windows** en la que es posible generar animaciones gráficas de forma sencilla. Las animaciones se generan en base a objetos (fondos, imágenes, gráficos generados desde el programa), los cuales se tienen que programar para que realicen una función determinada.

En la **UCI** se utiliza el **Scratch** como punto de partida en el curso de Algoritmia lo que permite formar la base a la asignatura de Introducción a la Programación, pero no se utiliza ninguna herramienta que tenga como objetivo mostrar mediante escenarios la ejecución de un código determinado en dicha asignatura. Además del **Scratch** durante los primeros momentos en que se imparte programación se utiliza una herramienta desarrollada por el departamento de Ingeniería Telemática de la Universidad “Carlos III” de Madrid en **Adobe Flash** denominada Algoritmos de Ordenación. Esta herramienta tiene como objetivo demostrar cómo funcionan los procesos de ordenamiento. Actualmente en el centro no se utiliza ninguna herramienta que sirva de apoyo a los temas de la asignatura de **IP** que por lo general son de difícil comprensión para los estudiantes que nunca se han enfrentado a la programación.

De los sistemas anteriormente analizados ninguno cumple con las expectativas del cliente, ya que lo que este necesita es una herramienta que haga hincapié en los temas iniciales que se imparten en la asignatura de introducción a la programación en la **UCI** y que se ajuste a su plan de estudio.

1.5.1 Metodologías de desarrollo de software.

Las Metodologías de Desarrollo de Software son un conjunto de procedimientos, técnicas y pasos a seguir para construir una aplicación. Una Metodología de Desarrollo de Software define quién debe hacer qué, cuándo y cómo para alcanzar un determinado objetivo. Es un proceso y en su modelación se definen como elementos principales los siguientes:

- **Trabajadores (Quién):** Define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo.
- **Actividades (Cómo):** Es una tarea que tiene un propósito claro, es realizada por un trabajador.
- **Artefactos (Qué):** Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.
- **Flujo de actividades (cuándo):** Secuencia de actividades realizadas por trabajadores y que producen un resultado de valor observable para el desarrollo del software. (10)

En la actualidad las tendencias metodológicas tienen su clasificación en dos grandes grupos: ágiles y tradicionales. Entre las metodologías tradicionales se encuentra **RUP** y dentro de las ágiles **XP**.

Rational Unified Process (**RUP**)**.**

La metodología **RUP**, llamada así por sus siglas en inglés (*Rational Unified Process*), fue creada por Jacobson, Rumbaugh y Booch. Se divide en 4 fases el desarrollo del software:

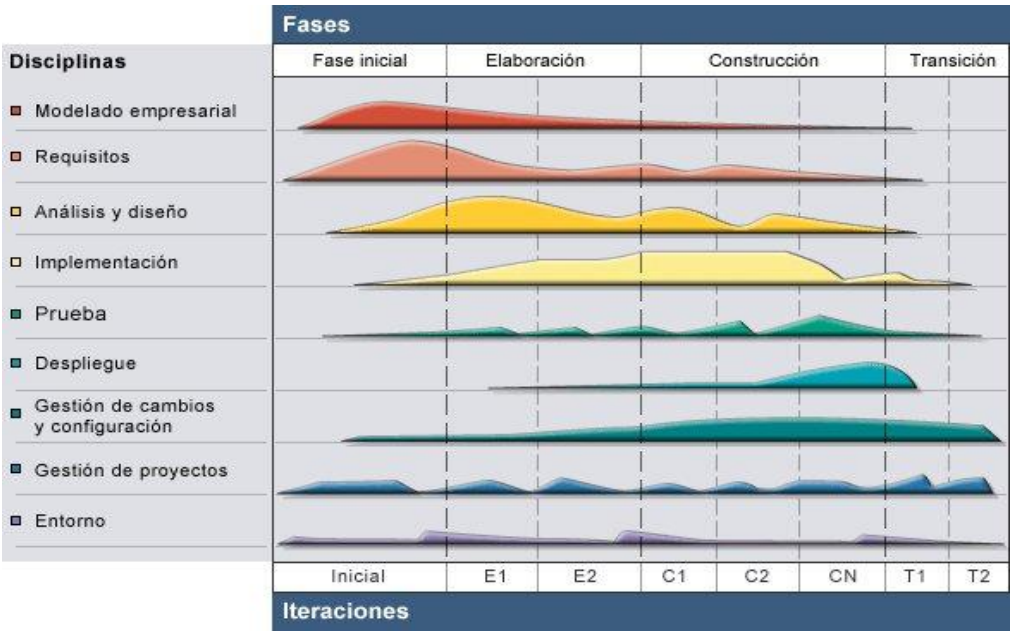

Figura 6 Fases y Flujos de Trabajo de RUP.

1. **Inicio:** Se describe el negocio y se delimita el alcance del proyecto con la determinación de los Casos de Uso del Sistema.
2. **Elaboración:** Se define la arquitectura del sistema y se obtiene una aplicación ejecutable que responde a los Casos de Uso (CU) que la componen. A pesar de que se desarrolla a profundidad una parte del sistema, las decisiones sobre la arquitectura se realizan sobre la base de la comprensión del sistema completo y los requerimientos (funcionales y no funcionales) identificados de acuerdo con el alcance definido.
3. **Construcción:** Se logra un producto listo para su utilización que está documentado y tiene un manual de usuario. Se obtiene una o varias versiones del producto que han pasado las pruebas. Se ponen estos entregables a consideración de un subconjunto de usuarios.
4. **Transición:** La versión ya está lista para su instalación en las condiciones reales. Puede implicar reparación de errores. (10)

Características de RUP.

- Unifica los mejores elementos de metodologías anteriores.
- Forma disciplinada de asignar tareas y responsabilidades en una empresa de desarrollo (quién hace qué, cuándo y cómo).
- Preparado para desarrollar grandes y complejos proyectos.

- Orientado a objetos.
- Se intenta conseguir el objetivo por medio de orden y documentación.
- Recomendado para ser utilizados en proyecto y equipos de desarrollos grandes en cuanto a tamaño y duración.
- Utiliza UML como lenguaje de representación visual. (10)

Para el proceso de desarrollo de la herramienta de representación visual esta metodología no es la más adecuada por las características del proyecto: Proyecto pequeño, el equipo de desarrollo está integrado por dos personas, no se necesita una documentación tan profunda, los requisitos cambian con frecuencia, el cliente forma parte del equipo de desarrollo.

Programación Extrema (XP).

XP es un enfoque de la ingeniería de software formulado por Kent Beck, autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999). Es el más destacado de los procesos ágiles de desarrollo de software. La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. (11)

La Programación Extrema (**XP**) consta de cuatro fases fundamentales: *Planificación*, *Diseño, Desarrollo o codificación* y *Pruebas.* A continuación se describen brevemente:

Primera Fase: Planificación del proyecto.

Historias de usuario: El primer paso de cualquier proyecto que siga la metodología **XP** es definir las historias de usuario con el cliente. Estas tienen la misma finalidad que los casos de uso pero con algunas diferencias: constan de 3 o 4 líneas escritas por el cliente en un lenguaje natural (no técnico) sin hacer mucho hincapié en los detalles.

Planificación de lanzamiento - Release Planning (RP): Después de tener definidas las historias de usuario es necesario crear un plan de publicaciones, donde se indiquen las historias de usuario que se crearán para cada versión del programa y las fechas en las que se publicarán estas versiones.

Iteraciones: Todo proyecto que siga la metodología **XP** se ha de dividir en iteraciones de aproximadamente 3 semanas de duración. Al comienzo de cada una de ellas los clientes deben seleccionar las historias de usuario definidas en el RP que serán

implementadas. También se seleccionan las historias de usuario que no pasaron el test de aceptación que se realizó al terminar la iteración anterior.

Programación en pareja: La metodología **XP** aconseja realizar la programación en parejas pues incrementa la productividad y la calidad del software desarrollado. Este trabajo involucra a dos programadores trabajando en el mismo equipo; mientras uno codifica haciendo hincapié en la calidad de la función o método que está implementando, el otro analiza si ese método o función es adecuado y está bien diseñado. (12)

Segunda Fase: Diseño.

Tarjetas C.R.C: El uso de las tarjetas C.R.C (Clase, Responsabilidad y Colaboración) permite al programador centrarse y apreciar el desarrollo orientado a objetos olvidándose de los malos hábitos de la programación estructurada. Estas representan objetos; la clase a la que pertenece el objeto se puede escribir en la parte de arriba de la tarjeta, en una columna a la izquierda se pueden escribir las responsabilidades que debe cumplir el objeto y a la derecha las clases que colaboran con sus respectivas responsabilidad. (13)

Tercera Fase: Codificación.

Crear test que prueben el funcionamiento de los distintos códigos implementados ayudará a desarrollar dicho código. Se puede dividir la funcionalidad que debe cumplir una tarea a programar en pequeñas unidades, de esta forma se crearán primero los test para cada unidad y a continuación se desarrollará dicha unidad, así poco a poco se conseguirá un desarrollo que cumpla todos los requisitos especificados. (13)

Cuarta Fase: Pruebas.

Uno de los pilares de la metodología **XP** es el uso de pruebas para comprobar el funcionamiento de los códigos que se vayan implementando, en esta fase es importante someter a pruebas las distintas clases del sistema omitiendo los métodos más triviales. (13)

Después de analizar algunas características de ambas metodologías, se decidió utilizar **XP** debido a que se adapta mejor a las condiciones de trabajo, el proyecto es pequeño y esta metodología está concebida para ser utilizada en este tipo de proyectos. Los requisitos del sistema cambian con frecuencia y uno de los principios básicos de esta metodología es que el cambio frecuente de los requerimientos es algo normal en el proceso de desarrollo, además el cliente forma parte del equipo permitiendo una mayor

retroalimentación. Por otra parte, **XP** aconseja la programación en parejas aumentando de esta forma la productividad y fomentando la comunicación entre el equipo de desarrollo.

1.5.2 Lenguaje Unificado de Modelado (UML)

UML (*Unified Modeling Language*) es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema de software. Sus creadores pretendieron con este lenguaje, unificar las experiencias acumuladas sobre técnicas de modelado e incorporar las mejores prácticas en un acercamiento estándar. El **UML** está compuesto por diversos elementos gráficos que se combinan para conformar diagramas y proporciona un estándar que permite al analista de sistemas generar un anteproyecto de varias facetas que sean comprensibles para los clientes, desarrolladores y todos aquellos que estén involucrados en el proceso de desarrollo de los sistemas. Esto se lleva a cabo mediante un conjunto de símbolos y diagramas. (14)

Existe un conjunto de herramientas **CASE** (*Computer Aided Systems Engineering*), que dan asistencia a analistas, ingenieros de software y desarrolladores durante el ciclo de vida de desarrollo de una aplicación.

Herramientas CASE de Modelado con UML

A medida que los sistemas que hoy se construyen se tornan más y más complejos, las herramientas de modelado con **UML** ofrecen mayores beneficios para todos los involucrados en un proyecto, ya que permiten aplicar la metodología de análisis y diseño orientados a objetos y abstraernos del código fuente, en un nivel donde la arquitectura y el diseño se tornan más obvios y más fáciles de entender y modificar. Cuanto más grande es un proyecto, es más importante la utilización de una herramienta **CASE**. Los analistas de negocio/sistema pueden capturar los requisitos del negocio/ sistema con un modelo de casos de uso.

Ventajas de estas herramientas.

- Los diseñadores, arquitectos pueden producir el modelo de diseño para articular la interacción entre los objetos, o los subsistemas de la misma o de diferentes capas (los diagramas **UML** típicos que se crean son los de clases y los de interacción).

- Los desarrolladores pueden transformar rápidamente los modelos en una aplicación funcional y buscar un subconjunto de clases, métodos y un entendimiento mediante las interfaces. (14)

Entre las herramientas **CASE** más utilizadas en la actualidad se encuentran **Rational Rose** y **Visual Paradigm**. A continuación se exponen algunas de las principales características de estas herramientas.

Rational Rose

Es la herramienta **CASE** desarrollada por los creadores de **UML** (Booch, Rumbaugh y Jacobson) que cubre todo el ciclo de vida de un proyecto: concepción, formalización del modelo, construcción de los componentes, transición a los usuarios, certificación de las distintas fases y entregables.

Características principales:

- Herramienta propietaria.
- La ingeniería de código (directa e inversa) es posible para **ANSI C++**, **Visual C++**, **Visual Basic 6**, **Java**, **J2EE/EJB**, **COBRA**, **Ada 83**, **Ada 95**, **Base de datos**: **DB2**, **Oracle**, **SQL92**, **SqlServer**, **Sybase**, **Aplicaciones Web**.
- Habilita asistentes para crear clases y provee plantillas de código que pueden aumentar significativamente la cantidad de código fuente generada. Adicionalmente, se puede aplicar los patrones de diseño, además ha provisto veinte de los patrones de diseño para **Java**.
- Admite la integración con otras herramientas de desarrollo. (15)

Visual Paradigm

Es una herramienta **UML** profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado **UML** ayuda a una rápida construcción de aplicaciones de calidad a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta **UMLCASE** también proporciona abundantes tutoriales de **UML**, demostraciones interactivas de **UML** y proyectos **UML**.

Características:

- Soporte de **UML**.
- Diagramas de Procesos de Negocio.
- Modelo colaborativo con **CVS** y subversión.
- Ingeniería inversa **Java**, **C++**, Esquemas **XML**, **XML**, **.NET** exe/dll, **COBRA**.
- Generación de código.
- Editor de detalles de Casos de Uso.
- Entorno todo en uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Generación de bases de datos.
- Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos.
- Generador de informes para generación de documentación.
- Distribución automática de diagramas.
- Reorganización de las figuras y conectores de los diagramas **UML**.
- Importación y exportación de ficheros **XMI**.
- Integración con **Visio**.
- Dibujo de diagramas **UML** con plantillas de MS **Visio**.
- Editor de figuras. (16)

Para el proceso de modelado se utilizará la herramienta **Visual Paradigm**, ya que permite un diseño centrado en las historias de usuario y enfocado al negocio. También el uso de un lenguaje estándar común a todo el equipo de desarrollo y capacidades de ingeniería directa e inversa. Administra un modelo y código que permanece sincronizado en todo el ciclo de desarrollo. Además, existe una disponibilidad de múltiples versiones, para cada necesidad; una disponibilidad de integrarse en los principales **IDEs** (*Integrated Development Environment*) y una disponibilidad en múltiples plataformas, además la **UCI** cuenta con licencia, permitiéndole su utilización.

1.5.3 Lenguajes de programación.

Un lenguaje de programación permite a un programador especificar de manera precisa: sobre qué datos una computadora debe operar, cómo deben ser éstos almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo

esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural, tal como sucede con el lenguaje léxico. (17)

A continuación se exponen las características de los lenguajes de programación que se tuvieron en cuenta para la selección del lenguaje a utilizar.

C++: Es un lenguaje imperativo orientado a objetos derivado del **C**. Nació para añadirle cualidades y características de las que carecía su antecesor. El resultado es que, como su ancestro, sigue muy ligado al hardware subyacente, manteniendo una considerable potencia para programación a bajo nivel, pero además se le han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción. (17)

Es un lenguaje de programación orientado a objetos, pero que también permite la programación estructurada. Puede ser utilizado tanto para escribir aplicaciones a bajo nivel, entre ellas drivers y componentes de sistemas operativos, como para el desarrollo rápido de aplicaciones, según el marco de trabajo con el que se disponga, como **VCL** (*biblioteca de componentes visuales*) de **Borland C++ Builder.** Además, los compiladores de **C++** generan código nativo con alto grado de optimización en memoria y velocidad, lo que lo convierte en uno de los lenguajes más eficientes.

Otro aspecto importante lo constituye la utilización de punteros por parte de los programadores, que por un lado aporta eficiencia, pero por otro es una fuente potencial de errores. Por este motivo, lenguajes derivados de **C++**, como **C#** y **Java**, liberaron a los programadores de esta responsabilidad y solo permiten referencias a objetos. (17)

Java: Es un lenguaje de programación muy extendido en la actualidad que cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general. Actualmente se utiliza bastante en el desarrollo de aplicaciones para teléfonos celulares y juegos, así como para aplicaciones Web. (18)

A continuación se describen algunos elementos de este lenguaje.

- **Robusto:** Se realiza un descubrimiento de la mayor parte de los errores durante el tiempo de compilación, ya que **Java** es estricto en cuanto a tipos y declaraciones. Respecto a la gestión de memoria, libera al programador del compromiso de tener que controlarla programáticamente. Posee una gestión avanzada de memoria llamada gestión de basura y un manejo de excepciones orientado a objetos integrados.

- **Simple:** Fácil aprendizaje: El único requerimiento para aprender **Java** es tener una comprensión de los conceptos básicos de la programación orientada a objetos. Así se ha creado un lenguaje simple (aunque eficaz y expresivo) pudiendo mostrarse cualquier planteamiento por parte del programador sin que las interioridades del sistema subyacente sean desveladas.

- **Completado con utilidades:** El paquete de utilidades de **Java** viene con un conjunto completo de estructuras de datos complejos y sus métodos asociados, que son de inestimable ayuda para implementar *Applets* y otras aplicaciones más complejas. Se dispone también de estructuras de datos habituales y clases ya implementadas. El **JDK** (*Java Development Kit*) suministrada por **SunMicrosystems** incluye un compilador, un intérprete de aplicaciones, un depurador de línea de comandos y un visualizador de *Applets* entre otros elementos. (18)

C#: Es un lenguaje orientado a objetos, simple y moderno que combina la alta productividad de lenguajes y rápido desarrollo de aplicación con el poder de **C** y **C++**

A continuación se enumeran las principales características que definen al lenguaje de programación C#. Algunas de estas características no son propias del lenguaje sino de la plataforma **.NET**, se listan aquí pues están implicadas directamente con el lenguaje:

- **Flexible:** En **C#** todo el código incluye numerosas restricciones para garantizar su seguridad, no permitiendo el uso de punteros. Sin embargo y a diferencia de **Java**, existen modificadores para saltarse esta restricción, pudiendo manipular objetos a través de punteros. Para ello basta identificar regiones de código con el identificador *unsafe* (inseguro) y podrán usarse en ellas punteros de forma similar a como se hace en **C++**.

- **Sencillez de uso:** Este lenguaje le quita el peso de algunos aspectos de la programación al usuario y lo asume el lenguaje, por ejemplo: elimina elementos añadidos por otros lenguajes y que facilitan su uso y comprensión, como son los ficheros de cabecera. Es por ello que se dice que **C#** es auto contenido, es decir no necesita de ficheros adicionales, tales como ficheros de cabecera.

- **Orientado a componentes:** La propia sintaxis de **C#** incluye elementos propios del diseño de componentes que otros lenguajes tienen que simular. La sintaxis de **C#** incluyen por ejemplo formas de definir propiedades, eventos o atributos.

- **Recolección de basura:** Todo lenguaje incluido en la plataforma **.NET** tiene a su disposición el recolector de basura del **CLR** (*Common Language Runtime,* en español Entorno Común de Ejecución para Lenguajes). Esto implica que no es necesario incluir instrucciones de destrucción de objetos en el lenguaje.

- **Seguridad de tipos:** Incluye mecanismos de control de acceso a tipos de datos, lo que garantiza que no se produzcan errores difíciles de detectar como un acceso inválido a memoria de algún objeto. Para ello, el lenguaje provee de una serie normal de sintaxis, como por ejemplo no realizar conversiones implícitas entre tipos que no sean compatibles. Además, no se pueden usar variables no primitivas que no se hayan inicializado previamente y en el acceso a tablas se hace una comprobación de rangos para que no se excedan ninguno de los índices de la misma. Se controlan los desbordamientos de operaciones aritméticas, produciéndose excepciones cuando se produzcan, además no permite asignarle un valor a una variable que sea de tipo diferente al valor.

- **Instrucciones seguras:** Se han impuesto una serie de restricciones en el uso de instrucciones de control más comunes. Por ejemplo, la evaluación de toda condición ha de ser una expresión condicional y no aritmética. Así se evitan errores por confusión del operador igualdad con el de asignación. Otra restricción que se impone en la instrucción de selección *switch*, imponiendo que todo *case* de la instrucción finalice con una instrucción *break* o *goto* que indique cuál es la siguiente acción a realizar.

- **Documentación, librerías y módulos**: La ayuda que trae **Visual Studio** puede ser instalada en el disco duro local del ordenador pero está también disponible en Internet además de un gran cúmulo de información de múltiples fuentes. El uso de módulos y librerías incluidos lo hacen bastante atractivo ya que son de gran utilidad al programador, permitiéndole ahorrar tiempo utilizando códigos ya desarrollados. (19)

Ventajas de C# sobre otros lenguajes:

Ventajas frente a C y C++.

- Recolección de basura automática.
- Libera al programador de la responsabilidad de manejar los punteros ya que el manejo de los mismos son fuente frecuente de errores.
- Soporta definición de clases dentro de otras.
- No existen funciones, ni variables globales, que no pertenezcan a una clase, ya sea estática o no.
- No se pueden utilizar valores no booleanos (enteros, como flotante, etc.) para condicionales, lo que permite un lenguaje más natural y menos propenso a errores.

Ventajas frente a Java.

- Indizadores que permiten acceder a cualquier objeto como si se tratase de un arreglo.
- Permite el uso (limitado) de punteros cuando realmente se necesiten, como al acceder a librerías nativas que no se ejecuten sobre la máquina virtual. (19)

Por todas las características analizadas anteriormente se decide utilizar el lenguaje **C#** para desarrollar la herramienta de apoyo a la docencia principalmente por ser un lenguaje muy utilizado en la UCI, fácil de aprender y existe abundante documentación sobre este lenguaje, es por esto que es considerado el que más se ajusta a los propósitos de la investigación.

1.5.4 IDE (Entorno Integrado de Desarrollo) a utilizar para la programación.

Un entorno de desarrollo integrado o **IDE** (*Integrated Development Environment*), es un programa informático compuesto por un conjunto de herramientas de programación. (20)

A continuación se analizarán algunos **IDEs** de programación: **SharpDevelop**, **MonoDevelop**, **Visual C#**, **Express Microsoft Visual Studio 2010** y las características distintivas analizadas de cada uno de ellos.

SharpDevelop:

Es una herramienta gratuita que actúa como un fiel entorno de desarrollo de programación basándose en lenguajes como: **C#** y **VisualBasic .NET** con ciertas características interesantes. Tiene la opción de completado para los lenguajes **C#**, **VisualBasic .NET** y **Boo**, aumentando la facilidad de programación. Integra una gran diversidad de plantillas de códigos de programación, creados para añadir o crear ficheros, proyectos o compiladores. Esto es de gran utilidad para aquellos que ya conocen la programación avanzada, pues el trabajo se les hace más fácil.

Además, es posible la edición de programación **XML** así como la pre-visualización a la hora de desarrollar el código. También se incluye un buscador de referencias dentro del mismo código, así como un asistente de reemplazamiento de fragmentos del código.

El programa tiene completa disponibilidad en español y puede ser extensible con *plugins* que están disponibles en su misma página oficial, como también algunas herramientas adicionales lo que hace que sea más práctico y completo. (21)

MonoDevelop:

Es un proyecto de código abierto creado por Ximián y actualmente impulsado por Novell con el objetivo de crear un grupo de herramientas libres, basado en **GNU/Linux** y compatibles con **.NET.** Es un entorno de desarrollo integrado libre y gratuito, diseñado principalmente para **C#** y otros lenguajes **.NET** así como para **Nemerle**, **Boo**, **Java** y **Python**.

MonoDevelop funciona tanto en **Linux**, **Windows** y **MacOS**. Permite depurar **ASP.NET**, **Moonlight** y programas de escritorio así como aplicaciones para *Iphone.* Integra características similares a las de **Eclipse** y **Visual Studio** como es el *Intellisense*. Además, tiene ayuda para los lenguajes **C #**, **Java**, **Boo**, **Nemerle**, **VisualBasic .NET**, **CIL**, **C** y **C++**. Entre otras funcionalidades, mantiene una copia en el disco duro de todas las ediciones hechas en un archivo, incluso si el archivo no se ha guardado, permitiendo poder recuperar aquellos fragmentos que no se almacenaron.

En un cierto plazo, el proyecto de **MonoDevelop** fue absorbido en el resto del proyecto **Mono** y es mantenido activamente por Novell y la comunidad de **Mono**. (22)

Visual C# Express.

Entre sus características se destacan:

- **Totalmente Personalizable:** Se ha diseñado para permitir crear una experiencia de desarrollo personalizada. Las ventanas como Explorador de Soluciones, Cuadro de Herramientas y Propiedades pueden redistribuirse fácilmente para adaptarse a sus necesidades.
- **Importación y exportación de configuraciones:** Posee un asistente de configuraciones permitiendo importar y exportar fácilmente la configuración y las preferencias del usuario a otros equipos.
- **Diseño Visual a Aplicaciones para Windows:** Proporciona un espacio de trabajo visual, eficaz para crear rápida y fácilmente aplicaciones interactivas para Windows.
- **Fácil de configurar e implementar:** El diseñador de proyectos es el lugar centralizado para configurar una aplicación. Permite configurar la pantalla de presentación de la aplicación, el formulario de inicio, el icono de la aplicación, los recursos, las opciones y mucho más (23).

Microsoft Visual Studio 2010.

Visual Studio 2010 Professional es un entorno integrado que simplifica la creación, depuración e implementación de aplicaciones. El mismo permite diseños potentes y métodos innovadores de colaboración para los desarrolladores y diseñadores. Trabaja dentro de un entorno personalizado, dirigido a un gran número de plataformas. Entre sus características se destacan:

- **Aplicaciones basadas en Windows:** Los nuevos formularios *Windows Forms* son intuitivos, muy eficaces y permiten ahorrar tiempo a la hora de realizar la parte visual de la aplicación. Estos formularios son compatibles con cualquier lenguaje basado en **.NET**, incluidos **Microsoft Visual Basic .NET** y **Microsoft Visual C# .NET**. Con la herencia visual, los programadores pueden simplificar enormemente la creación de aplicaciones basadas en Windows. Utilizando delimitadores y acoplamiento, los programadores pueden generar formularios que cambian de dimensión automáticamente, mientras el editor de menús permite crear menús de manera visual directamente desde el diseñador de *Windows Forms*.
- **Productividad:** Con la función para completar instrucciones de *IntelliSense* y la comprobación automática de errores de sintaxis, **Visual Studio 2010** informa a los

programadores cuando el código es incorrecto y proporciona el dominio inmediato de las jerarquías de clases y las **API**. Con el explorador de 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. Los asistentes para aplicaciones, las plantillas de proyecto y los ejemplos de código fuente permiten a los programadores crear aplicaciones con rapidez para **Windows**, la Web y dispositivos con una inversión inicial mínima.

- **Arquitectura:** El Explorador de arquitectura ayuda a conocer y liberar el valor de los activos de código existentes y sus interdependencias. Se pueden producir modelos detallados de cómo está construida exactamente una aplicación e incluso, analizar con profundidad áreas específicas para conocerlas mejor.
- **Herramientas de pruebas: Visual Studio Test Professional 2010** permite capturar y actualizar requisitos de pruebas, automatizar la navegación de pruebas manuales, acelerar la solución y aceptar el ciclo mediante la captura de todo el contexto de la prueba. (24)

Se seleccionó como **IDE** de programación a **Microsoft Visual Studio 2010** por la eficiencia tanto comprobando el código introducido como autocompletándolo, además permite el diseño de formularios de forma sencilla e intuitiva. En la **UCI** los estudiantes que reciben la asignatura de IP en el lenguaje **C#** utilizan este **IDE** en sus clases y a pesar de ser software propietario el producto final no se vería afectado pues se utilizaría dentro de las instalaciones de la Universidad.

1.5.5 Generadores de analizadores Léxico y Sintáctico.

Los generadores de analizadores son conocidos como compiladores de compiladores dado que ayudan a implementar las primeras fases de los mismos. Estas herramientas facilitan en gran medida la implementación, puesto que a partir de un fichero de especificación es posible generar algunas etapas de un compilador en diferentes lenguajes.

Gppg (analizador Sintáctico) y **Gplex** (analizador Léxico) son generadores de analizadores léxico y sintáctico escritos en lenguaje **C#,** estas herramientas brindan la posibilidad de generar informes **HTML** permitiendo una fácil navegación del autómata finito y reconociendo los prefijos del lenguaje especificado, además muestran el camino

más corto para llegar a cualquier estado. Los analizadores se generan para ser totalmente compatibles entre ellos. En la actualidad existe muy poca documentación sobre estas herramientas, cada una de ellas solo puede ser usada para generar una de las etapas del compilador y solo se pueden utilizar para generar analizadores en lenguaje **C#.**

ANTLR es un generador de analizadores que ayuda a implementar compiladores, es un programa escrito en **Java** por lo que necesita de su máquina virtual, **ANTLR** se encarga de generar las dos primeras fases de un compilador y facilita en gran medida el trabajo en el resto. La especificación de las gramáticas con que trabaja es bastante intuitiva y uniforme. El código que genera es legible y puede ser generado en alguno de los siguientes lenguajes: **C++**, **Java**, **Python** y **C#**. Se puede mencionar además que es independiente de la plataforma, es software libre por lo que en su sitio oficial se puede descargar tanto los ficheros compilados como el código fuente.

Para generar las 2 primeras etapas de la herramienta de representación de conceptos básicos se decidió utilizar **ANTLR** ya que dicha herramienta permite generar ambas etapas del intérprete con un único archivo de especificación, además existe más documentación en varios idiomas incluido el español.

1.6 Conclusiones parciales

En este capítulo se abordaron los conceptos fundamentales vinculados al campo de acción, una descripción de las fases del compilador, además se abordó el estado actual de las herramientas de soporte a la enseñanza, en especial la enseñanza de la programación, así como un estudio de las tecnologías y tendencias actuales en cuanto a desarrollo de aplicaciones, seleccionando las siguientes para el desarrollo de la herramienta de apoyo a la asignatura de Introducción a la Programación: como metodología de desarrollo de software Extreme Programing (**XP**), como herramienta Case Visual Paradigm, lenguaje de programación **C#** y como entorno de desarrollo (**IDE**) el **Microsoft Visual Studio 2010**.

Capítulo 2 “Propuesta del Sistema”.

2.1 Introducción.

El presente capítulo tiene como objetivo realizar una valoración de las principales características de la herramienta a desarrollar teniendo en cuenta la situación problémica que dio origen al mismo, se describen las necesidades de los usuarios, describiéndose las funcionalidades que se van a automatizar, también se presentará una propuesta de solución especificando detalladamente los requisitos funcionales y no funcionales. Por último, se abordarán los Patrones de Diseño y Estándares de Codificación.

2.2 Descripción de las acciones vinculadas al campo de acción.

Actualmente en la Universidad de Ciencias Informáticas (**UCI**) el traceo del código creado en lenguaje **C#** se hace de forma manual o en el Inspector de Objeto de **Microsoft Visual Studio**, a pesar de las ventajas con que cuenta este **IDE**, realizar esta tarea mediante estas opciones resulta menos atractiva que de forma gráfica.

2.3 Flujo actual de los procesos.

El análisis del flujo actual de los procesos conlleva a que se pueda conocer cómo funciona en realidad el negocio y obtener posibles resultados. Estos últimos pueden ser: un servicio, una información determinada, un producto o combinaciones de ellos. El análisis de cómo se llevan a cabo estos procesos permite identificar un conjunto de problemas que tienen lugar en la entidad donde se desarrollan y que dificultan el buen desarrollo de las actividades que allí tienen lugar, como por ejemplo: pérdida, duplicación, demoras en la búsqueda o incorrecto formato de la información, entre otras.

Los estudiantes de nuevo ingreso de la **UCI** en el curso de nivelación reciben una asignatura denominada **Algoritmización**, durante este período se apoyan en una herramienta llamada **Scratch** pero al comenzar el semestre en la asignatura de **IP** (Introducción a la Programación) no cuentan con una herramienta que tenga como objetivo tracear el código y ayudar a comprenderlo. En esta etapa se apoyan en el traceo del código y del inspector de objeto que tiene **Microsoft Visual Studio**.

Antes de comenzar a construir la herramienta en cuestión es necesario destacar por qué se decide realizarla:

- Obtener una herramienta que permita la retroalimentación del estudiante para comprender el código.
- Permitir al estudiante la posibilidad de aprender de sus errores.
- Proporcionar a los estudiantes algunos ejemplos que les servirán de apoyo en el proceso de aprendizaje.
- Visualizar y animar los resultados en tiempo real del código creado.

2.4 Descripción de los procesos que serán objeto de automatización.

Del proceso de traceo de código se automatizarán algunas acciones por ser largas y en ocasiones difíciles.

Se automatizará la forma de mostrar los resultados del código, pues se realizará a través de una animación consecutiva, lo que facilitará en gran medida el trabajo y la comprensión del mismo. Además, no será necesario utilizar puntos de ruptura pues el código se irá traceando línea por línea permitiéndole al estudiante modificar la velocidad de animación para facilitar su comprensión.

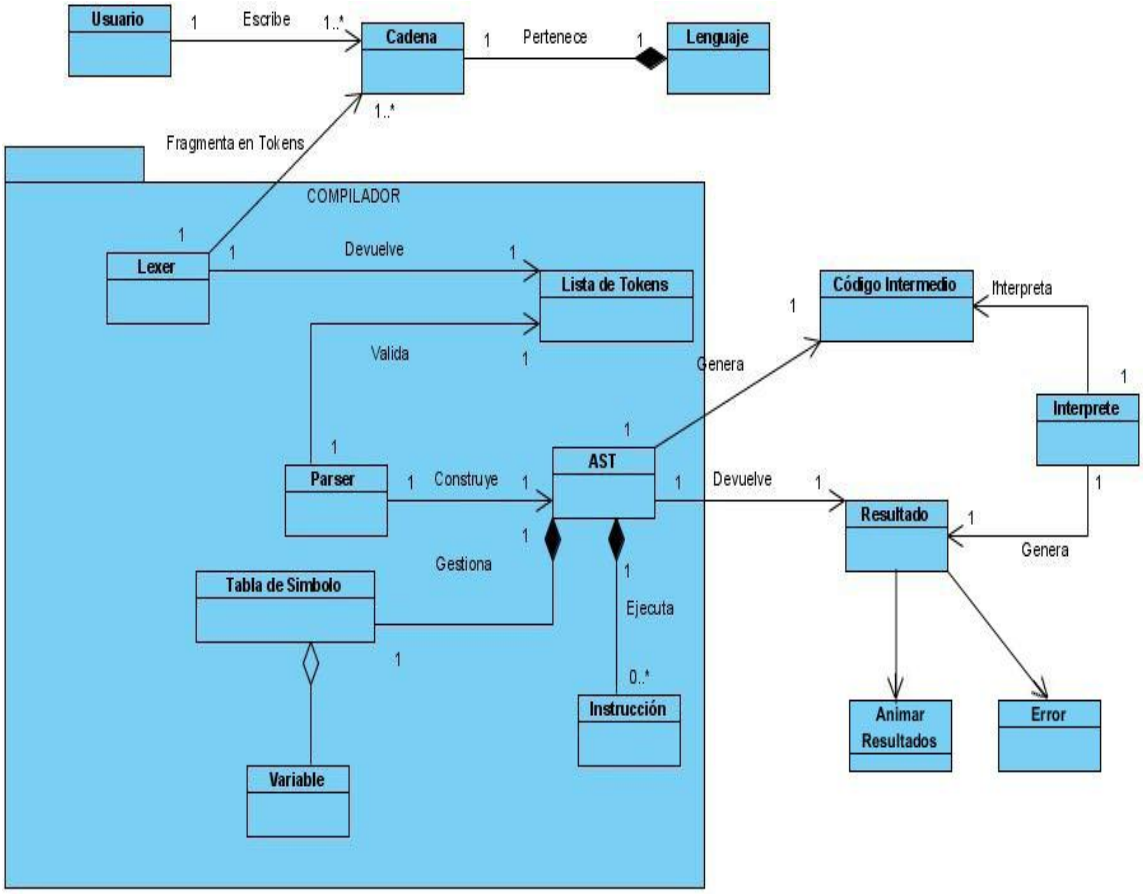

Figura 7 Proceso Principal.

2.5 Funcionalidades del sistema.

Una vez conocidos todos los conceptos que rodean el objeto de estudio, se puede analizar qué debe hacer el sistema para que se cumplan los objetivos planteados al inicio de este trabajo. Para ello se enumeran a través de requerimientos funcionales las funcionalidades que el sistema será capaz de realizar.

1. Realizar análisis léxico.

1.1 El sistema debe ser capaz de reconocer:

- Las operaciones aritméticas (Suma, Resta, Multiplicación, División, Resto).
- Las expresiones lógicas y operaciones de comparación (Menor, Mayor, Menor Igual, Mayor Igual, Igualdad, Diferencia, O lógico, Y lógico).
- Instrucciones condicionales y repetitivas (IF-ELSE, FOR, WHILE, SWITCH).

2. Realizar análisis Sintáctico.

2.1 El sistema debe ser capaz de reconocer si las expresiones introducidas por el usuario son sintácticamente correctas.

3. Realizar análisis Semántico.

3.1 El sistema debe ser capaz de reconocer si las expresiones introducidas por el usuario son semánticamente correctas.

4. Mostrar Animación.

4.1 Una vez que el código introducido está correcto el sistema debe mostrar de manera gráfica la ejecución del programa creado por el usuario.

4.2 El sistema debe ser capaz de representar gráficamente:

- Operaciones aritméticas (Suma, Resta, Multiplicación, División, Resto).
- Las expresiones lógicas y operaciones de comparación (Menor, Mayor, Menor Igual, Mayor Igual, Igualdad, Diferencia, O lógico, Y lógico).
- Instrucciones condicionales y repetitivas (IF-ELSE, FOR, WHILE, SWITCH).

5. Cargar y salvar código.

5.1 El sistema debe permitir guardar el código introducido por el usuario.

5.2 El sistema debe permitir cargar código guardado con anterioridad y modificarlo en caso necesario.

6. Tratar Errores.

6.1. El sistema debe ser capaz de informar los diferentes errores que puedan presentarse (Lexicográficos, Semánticos, Sintácticos).

2.6 Características del sistema.

Los requerimientos no funcionales se refieren directamente a las propiedades emergentes del sistema, es decir, a las características que este debe tener.

Soporte.

- La herramienta contará con una ayuda que contendrá indicaciones básicas para utilizar el software.

Requerimientos de Hardware.

- 256 MB de RAM como mínimo.
- Microprocesador de 1,66Ghz.
- Espacio libre en el disco duro de 12 Mb.

Software.

- Las computadoras que utilizarán el software deberán tener instalado Windows XP o una versión superior.
- Microsoft .NET Framework 4.

2.7 Patrones de diseño.

Un patrón es una descripción de un problema bien conocido que suele incluir: descripción, escenario de Uso, solución concreta, las consecuencias de utilizar este patrón, ejemplos de implementación, lista de patrones relacionados. (25)

Patrones **GRASP**: son patrones generales de software para asignación de responsabilidades, es el acrónimo de "General Responsibility Assignment Software

Patterns". Aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendables en el diseño de software. (25)

1. Experto: se encarga de asignar una responsabilidad al experto en información, o sea, la clase con información suficiente para cubrir la necesidad. La clase Lexer y la clase Parser son responsables cada una de una fase del intérprete, la clase Lexer contiene y es responsable del análisis lexicológico y cubre las necesidades durante esa etapa, así mismo ocurre con la clase Parser que se encarga del análisis sintáctico del intérprete.

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

- B contiene a A.
- B es una agregación (o composición) de A.
- B almacena a A.
- B tiene los datos de inicialización de A (datos que requiere su constructor).
- B usa a A.

La clase Lexer es la encargada de generar la tabla de símbolos reconocidos por el sistema y la clase Parser es la encargada de hacer el análisis sintáctico partiendo de la tabla de símbolos que tiene la clase Lexer. En la clase Parser se evidencia el uso de este patrón mediante la creación de una instancia de la clase Lexer.

3. Alta Cohesión: Cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos.

La forma en que está organizado el proyecto permite trabajar con clases con una alta cohesión, cada clase tiene una función única dentro de la herramienta, ejemplo de ello es la clase Lexer, Parser y Semántica, encargadas de los análisis léxico, sintáctico, semántico.

4. Bajo Acoplamiento: Garantiza que exista poca dependencia entre las clases ya que mucha dependencia entre ellas dificulta su uso.

De modo general en el sistema el acoplamiento es muy bajo así como la dependencia entre las clases, no existe una herencia profunda.

5. Controlador: Propone asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades. El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. (25)

La clase Controladora evidencia la utilización de este patrón ya que controla la información y los eventos del sistema.

2.8 Estándares de codificación.

Un código escrito con buen estilo, en cualquier lenguaje es aquel que cumple con las siguientes propiedades:

- Está organizado.
- Es sencillo de entender.
- Es fácil de mantener.
- Posibilita la detección de errores con facilidad.
- Es eficiente. (26)

Encapsulación y Ocultación: Ayudan a organizar mejor el código y evitan el acoplamiento entre funciones. La encapsulación permite agrupar elementos afines del programa. Para lograr esto se deben seguir las siguientes reglas:

- Declarar las variables y tipos como locales a los subprogramas que los utilizan.
- Pasar los parámetros por valor para evitar cambios indeseados.
- Asegurar que un procedimiento sólo modifique los parámetros pasados en su llamada.

Indentación: La indentación debe ser a cuatro espacios sin caracteres de tabulación, se debe realizar con tabulaciones, por tanto, se debe fijar éste en cuatro caracteres que es la opción por defecto del entorno Visual Studio .NET.

```
namespace WindowsFormsApplication3
{
    static class Program
    {
        static void Main()
        {
            int a = 0;
        }
    }
}
```

Comentarios: Permiten describir ciertas líneas de código o deshabilitar código no necesario. El estilo de los comentarios debe ser (/* */ o //). Éstos deben ser elaborados con un lenguaje formal siempre teniendo en cuenta la ortografía, además deben ser simples y directos.

```
/*
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
Application.Run(new Form1());
*/
tokens.GetTokens(); //Permite Obtener la lista de Tokens identificados.
```

Estructuras de Control: Deben utilizarse las llaves en cualquier caso "{ }", incluso en situaciones donde son técnicamente opcionales. Su uso incrementa la capacidad de lectura y reduce la probabilidad de errores lógicos que son introducidos cuando líneas nuevas se agregan. Éstas incluyen: if, for, while, switch, etc.

```
if (a == true)
{
    return a;
}
```

Llamadas de Función: Las funciones deberán ser llamadas sin espacios entre el nombre de la función, el paréntesis abierto y el primer parámetro. Tampoco debe tener espacios entre el último parámetro y el cierre del paréntesis. Los parámetros se separarán por coma.

```
utils.GetName(id, true);
```

Definiciones de Función: En las declaraciones de función, la llave de apertura comienza debajo de la línea de definición. Además, deberán ser definidas sin espacios entre el nombre de la función, el paréntesis abierto y el primer parámetro. Tampoco deben tener espacios entre el último parámetro y el cierre del paréntesis. Los parámetros se separarán por coma.

```
public bool NameExists(string name)
{
   ...
}
```

Convención de Nombres de Variables, Funciones, Clases:

Variables: Comenzarán con letra minúscula, en caso de ser un nombre compuesto se utilizará una mayúscula al comenzar la siguiente palabra.

```
string name;
int birthdayYear;
```

Funciones: Estas deberán nombrarse con la primera letra en mayúscula y en caso de ser un nombre compuesto se utilizará una mayúscula al comenzar la siguiente palabra.

```
public bool NameExists(string name)
{
   ...
}

public bool Exists(string name)
{
   ...
}
```

Clases: Estas deberán nombrarse con la primera palabra iniciando con mayúscula y en caso de ser un nombre compuesto se utilizará una mayúscula al comenzar la siguiente palabra.

```
public class Program
{

   ...

}
```

2.9 Propuesta del sistema.

La solución idónea para resolver el problema es la creación de un intérprete de **C#** que sirva de apoyo al proceso docente educativo en la **UCI**, pues en la misma se imparte este lenguaje mediante el uso de las nuevas Tecnologías de la Información y las Comunicaciones (**TIC**).

La herramienta permitirá eliminar los posibles errores que cometan los estudiantes al tracear un código. El sistema estará conformado por varias áreas de trabajo, cada una brindando diferentes facilidades, tendrá un área de trabajo permitiendo al usuario introducir código, además de botones que permitan controlar la animación de un escenario. El espacio de trabajo estará dividido en tres secciones: declaración de variables y constantes, evaluación de expresiones y trabajo con arreglos. Son necesarias

dichas divisiones pues las mismas posibilitan mostrar la información de una forma sencilla, así como la ayuda que facilitará el entendimiento de cada área de trabajo y sus ventajas.

2.10 Conclusiones parciales.

En este capítulo se analizaron las herramientas y métodos empleados para la impartición de los principios básicos de la asignatura de Introducción a la Programación en la **UCI**, basándose en las deficiencias encontradas y las necesidades del usuario se definieron las funcionalidades que la herramienta de apoyo debería cumplir con el propósito de brindar al estudiante una vía más fácil para la aprehensión de los principios antes mencionados. Por último, se realizó la selección de los patrones de diseño y estándares de codificación, permitiendo la estandarización y optimización del código.

CAPÍTULO 3 ¨Desarrollo de la Solución¨.

3.1 Introducción.

En el presente capítulo se desarrolla con mayor especificación la fase de exploración y planificación, haciendo referencia a todo lo concerniente a la misma y una descripción de cada uno de los artefactos generados, dentro de los cuales se registran las historias de usuario y el plan de iteración. Además, se detallan las historias de usuario (**HU**) para luego establecer el orden en que serán implementadas atendiendo a su prioridad.

Fase de exploración y planificación.

3.2 Historias de usuario (HU).

Las **HU** se utilizan para representar los requerimientos del sistema, tarea que corresponde desarrollar al usuario integrado en el equipo de desarrollo. En aras de conseguir simplicidad y no generar documentación demasiado pesada, las **HU** son una descripción de las necesidades funcionales que no deben ocupar muchas líneas, con algunos campos más. La idea es que sea sencilla. (13)

Plantilla de historia de usuario.

Para definir las historias de usuario (**HU**) se utiliza la plantilla siguiente, la cual contiene todos los datos necesarios para desarrollar la funcionalidad descrita:

Tabla 1 Plantilla de Historia de Usuario

Historia de Usuario	**Tiempo Real:** <Permite conocer que tiempo demoró la implementación>	**Ptos. Estimación: <***Permiten estimar duración de implementación***>**
Número: <*Número de la HU, incremental en el tiempo*>	**Nombre de la Historia de Usuario: <***El nombre de la HU, sirve para identificarla fácilmente entre los desarrolladores y los clientes***>**	
Modificación de Historia de Usuario:		
Usuario: <*El usuario del sistema que utiliza o protagoniza la historia***>**		**Iteración Asignada: <***La iteración (liberación en nuestro proceso) a la que corresponde***>**
Prioridad en Negocio: <*Qué tan importante es para el cliente***>**		**Riesgo en Desarrollo: <***Qué tan difícil es para el desarrollador* **>**

Descripción:< *La descripción de la historia, detallando las operaciones del usuario y opcionalmente las respuestas del sistema.>*
Observaciones: <*Algunas observaciones de interés, como glosario, información sobre usuarios, etc. >*

Durante todo el proceso se identificaron 6 Historias de Usuario.

- Análisis Léxico.
- Análisis Sintáctico.
- Análisis Semántico
- Animar Resultados.
- Salvar y Guardar Código.
- Tratar Errores.

Tabla 2 HU Análisis Léxico.

Historia de Usuario	**Tiempo Real:** 1	**Ptos. Estimación:** 2
Número: 1	**Nombre de la Historia de Usuario:** Análisis Léxico	
Modificación de Historia de Usuario:		
Usuario: Estudiantes y profesores.	**Iteración Asignada:** 1	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Medio	
Descripción: El sistema recibe el código introducido por el usuario lo analiza y produce como salida un conjunto de tokens.		
Observaciones: Hace referencia al requisito 1.		

Tabla 3 HU Análisis Sintáctico.

Historia de Usuario.	**Tiempo Real:** 2.	**Ptos. Estimación:** 2.
Número: 2.	**Nombre de la Historia de Usuario:** Análisis Sintáctico.	
Modificación de Historia de Usuario:		
Usuario: Estudiantes y profesores.	**Iteración Asignada:** 1.	
Prioridad en Negocio: Alta.	**Riesgo en Desarrollo:** Medio.	

Descripción: El sistema impone una organización jerárquica a la cadena de tokens recibidos del análisis léxico, la misma es representada en forma de árbol.
Observaciones: Hace referencia al requisito 2.

Tabla 4 HU Análisis Semántico.

Historia de Usuario.	**Tiempo Real:** 3.	**Ptos. Estimación:** 3.
Número: 3	**Nombre de la Historia de Usuario:** Análisis Semántico.	
Modificación de Historia de Usuario:		
Usuario: Estudiantes y profesores.	**Iteración Asignada:** 1.	
Prioridad en Negocio: Alta.	**Riesgo en Desarrollo:** Medio.	
Descripción: El sistema debe revisar el código fuente a partir del árbol sintáctico generado, comprobar las restricciones de tipo y otras limitaciones semánticas.		
Observaciones: Hace referencia al requisito 3.		

Tabla 5 HU Animar Resultados.

Historia de Usuario.	**Tiempo Real:** 3.	**Ptos. Estimación:** 3.
Número: 4.	**Nombre de la Historia de Usuario:** Animar Resultados.	
Modificación de Historia de Usuario:		
Usuario: Estudiantes y profesores.	**Iteración Asignada:** 2.	
Prioridad en Negocio: Alta.	**Riesgo en Desarrollo:** Medio.	
Descripción: Inicia después que se comprobó que el código introducido es válido. Las operaciones y resultados del código introducido se muestran en el área de animación.		
Observaciones: Brinda la posibilidad de detener, adelantar y rebobinar la animación. Hace referencia al requisito 4		

Tabla 6 HU Salvar y Cargar Código.

Historia de Usuario.	**Tiempo Real:** 1.	**Ptos. Estimación:** 1.
Número: 5.	**Nombre de la Historia de Usuario:** Salvar y Cargar Código.	
Modificación de Historia de Usuario:		
Usuario: Estudiantes y profesores.	**Iteración Asignada:** 1.	
Prioridad en Negocio: Alta.	**Riesgo en Desarrollo:** Bajo.	
Descripción: Inicia cuando el usuario desea guardar el código introducido dando clic en el botón guardar o cuando desee abrir las instrucciones que guardó anteriormente haciendo clic en el botón cargar.		
Observaciones: Hace referencia al requisito 5		

Tabla 7 HU Tratar Errores.

Historia de Usuario.	**Tiempo Real:** 2.	**Ptos. Estimación:** 2.
Número: 6.	**Nombre de la Historia de Usuario:** Tratar Errores.	
Modificación de Historia de Usuario:		
Usuario: Estudiantes y profesores.	**Iteración Asignada:** 2.	
Prioridad en Negocio: Media.	**Riesgo en Desarrollo:** Bajo.	
Descripción: Inicia después que el usuario compila el código, en caso que presente errores se deben mostrar una notificación del error, la línea donde se encuentra el mismo y posibles soluciones.		
Observaciones: Hace referencia al requisito 6.		

3.2.1 Plan de entrega.

Las historias de usuario se utilizan para crear el plan estimado de entrega. Este se utiliza para crear los planes correspondientes a cada iteración. Para crearlos se convoca a una reunión donde participan los desarrolladores y los usuarios. Con cada historia de usuario previamente evaluada, el cliente las agrupará según la importancia. De esta

forma, se puede trazar el plan de entrega en función de estos dos parámetros: el tiempo de desarrollo ideal y el grado de importancia para el cliente. Las iteraciones individuales son planificadas en detalle justo antes de que comience cada iteración. Las estimaciones de esfuerzo asociado a la implementación de las historias se establecen utilizando como medida el punto. Un punto equivale a una semana ideal de programación, por ello generalmente valen de 1 a 3 puntos dichas estimaciones. Por otra parte, se mantiene un registro de la velocidad de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración. La velocidad del proyecto es utilizada para establecer cuantas historias se pueden implementar antes de una fecha determinada o cuanto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según el alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniéndose el número de iteraciones necesarias para su implementación. (13)

A continuación se muestra la tabla donde aparecen datos relacionados con las historias de usuario:

Tabla 8 Plan de esfuerzo e iteración por Historias de Usuario.

No.	Nombre.	Prioridad.	Riesgo.	Esfuerzo.	Iteración.
1	Análisis Léxico.	Alta.	Medio.	2.	1.
2	Análisis Sintáctico	Alta.	Medio.	2.	1.
3	Análisis Semántico	Alta.	Medio.	3.	1.
4	Animar Resultados.	Alta.	Medio.	3.	2.
5	Salvar y Cargar Código.	Alta.	Bajo.	1.	1.
6	Tratar Errores.	Media.	Bajo.	2.	2.

El plan de entrega elaborado para la fase de implementación se presenta a continuación.

Tabla 9 Plan de duración de entrega.

Módulos.	Final de 1era iteración: Primera quincena de marzo.	Final de 2da iteración: Segunda quincena mayo.
Análisis Léxico.	1.0 Final.	Finalizado.
Análisis Sintáctico.	1.0 Final.	Finalizado.
Análisis Semántico.	1.0 Final.	Finalizado.
Animar Resultados.		1.0 Final.
Salvar y Cargar Código.	1.0 Final.	Finalizado.
Tratar Errores.		1.0 Final.

3.2.2 Plan de iteración.

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El plan de entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura. Los elementos que deben tomarse en cuenta durante la elaboración del plan de iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresable, pero llevadas a cabo por parejas de programadores. Un plan de iteración puede verse como:

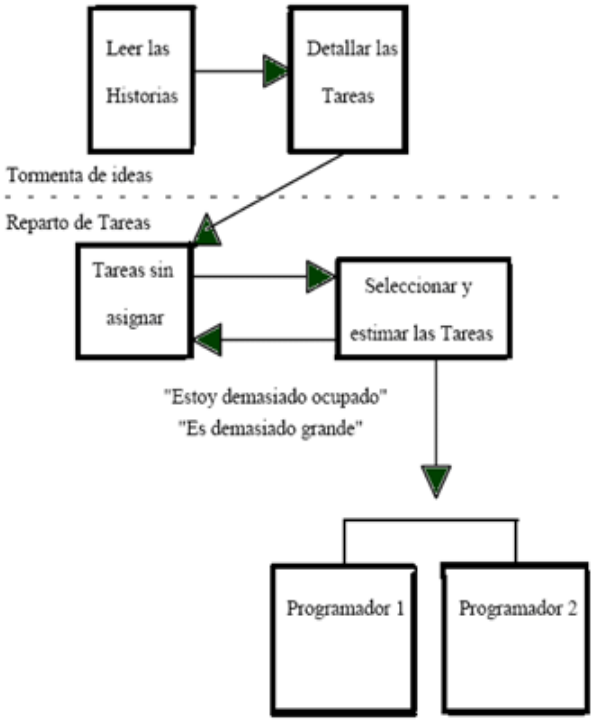

Figura 8 Plan de Iteración. (13)

Cada historia de usuario es transformada en tareas de desarrollo. Cada tarea de desarrollo corresponde a un período ideal de uno a tres días de desarrollo. Es importante vigilar la velocidad del proyecto y el movimiento de historias de usuario. Puede ser necesario volver a calcularlas y negociar el plan de entrega de tres a cinco iteraciones. Como se estará siempre implementando las HU más importantes para el cliente, se hará lo máximo posible por él y la dirección. (13)

Iteración1: En la primera iteración se entregarán las funcionalidades de las historias de usuario 1, 2, 3, y 5 que tienen prioridad alta, las mismas son:

- Análisis Léxico.
- Análisis Sintáctico.
- Análisis Semántico.
- Salvar y Cargar Código.

Al terminar la iteración el cliente tendrá disponible una versión a la cual se le aplicarán los primeros tests para probar su calidad funcional.

Iteración2: En esta iteración se realizan las restantes historias 2 y 4:

- Animar Resultados.
- Tratar Errores.

El objetivo principal de las pruebas en esta iteración consiste en encontrar deficiencias en la solución así como inconformidades del cliente.

3.2.3 Plan de duración de las iteraciones.

Partiendo de que cada iteración corresponde a un período de tiempo de entre 1 y 3 semanas, al principio de cada iteración se debe convocar a una reunión en la cual se trace el plan de iteración correspondiente. Se utiliza entonces la velocidad del proyecto para determinar si una iteración está sobrecargada. La suma de los días que cuesta desarrollar todas las tareas de la iteración no debe sobrepasar la velocidad del proyecto de la iteración anterior. Si la iteración está sobrecargada, el cliente debe decidir que historias de usuario retrasar a una iteración posterior.

Tabla 10 Plan de duración de las iteraciones.

Fase	Artefactos Generados	Duración total de las iteraciones
Exploración.	• Captura de Historias de Usuario.	1 semanas
Diseño.	• Targetas CRC	2 semanas
Implementación.	• Análisis Léxico. • Análisis Sintáctico. • Análisis Semántico. • Salvar y Cargar Código.	9 semanas
	• Animar Resultados. • Tratar Errores.	10 semanas
Pruebas.	• Pruebas unitarias y de aceptación.	2 semanas

3.3 Conclusiones parciales

Con la realización del presente capítulo se inició el desarrollo de la propuesta de solución, se describieron las fases de la metodología **XP**; haciendo énfasis en la información correspondiente a la fase de Planificación. Así como el plan de iteraciones agrupando las historias de usuario según la prioridad asignada y el plan de entrega para estimar la duración del proyecto, quedando plasmado que para desarrollar la herramienta se realizarán dos iteraciones, programando en pareja como propone la metodología utilizada y con una duración de 24 semanas.

Capítulo 4 ¨CONSTRUCCIÓN Y PRUEBA DE LA PROPUESTA DE SOLUCIÓN¨.

4.1 Introducción

En este capítulo se describe la gramática y el archivo de especificación utilizados en el generador de analizadores Léxicos y Sintácticos **ANTLR.** Además, se muestran los diagramas de presentación correspondientes a la vista de presentación, las mismas permiten observar cuáles son las interfaces que interactúan con el usuario. Por último se muestran los resultados de las pruebas realizadas al producto final, permitiendo conocer si este cumple o no con las especificidades y requerimientos solicitados.

Diseño del sistema.

4.2 Tarjetas CRC

Las tarjetas **CRC** permiten a los desarrolladores enfocarse hacia el diseño de las clases dentro del sistema. Son una técnica de modelado introducida inicialmente con el objetivo de enseñar los conceptos del paradigma orientado a objeto. Están divididas en tres secciones que representan los objetos de las clases, sus responsabilidades y las clases que colaboran con él. (27)

Las clases representan un conjunto de objetos, en las tarjetas **CRC** se escriben en singular dado que representan a una instancia en particular. Las responsabilidades son los datos que el objeto conoce o las acciones que es capaz de realizar, para las cuales en ocasiones no tiene la información necesaria. Las colaboraciones son las clases que le pueden brindar los datos necesarios para completar sus responsabilidades. (27)

La metodología **XP** plantea el uso de las tarjetas **CRC** como parte de sus reglas o prácticas para el diseño por los detalles que aportan al diseño de las clases dentro del sistema. Este proceso incluye a todo el equipo de trabajo lo cual le aporta al diseño mayor cantidad de ideas para así obtener un resultado mejor. La herramienta propuesta consta de cinco clases fundamentales que se describen a continuación.

Clase Scanner

Tabla 11 Descripción de la CRC Lexer.

Responsabilidades	Clases relacionadas
Scan(): Recibe un flujo de código en C# y devuelve un *token*	Parser Compiler

asociado a ese código. Chequea errores Léxicos. **IsLetter():** Dado un carácter pasado por parámetro, devuelve si es una letra o no. **FillKeywordsTable():** Llena la tabla de símbolos con los valores predeterminados.	

Clase Parser

Tabla 12 Descripción de la CRC Parser.

Responsabilidades	Clases relacionadas
Parse(): Recibe un conjunto de *tokens* que devuelve la clase Lexer y genera el árbol de sintaxis abstracta. Chequea errores sintácticos. **ParseDeclaration():** Dado un conjunto de tokens correspondientes a una declaración, devuelve el objeto de tipo Declaration asociado. **GetType():** Dado un objeto de tipo TokenKind que recibe por parámetro, devuelve el EasyTypes asociado.	Lexer Compiler AST

Clase Compiler

Tabla 13 Descripción de la CRC Compiler.

Responsabilidades	Clases relacionadas
Compile(): Dirige el proceso de compilación del código y se encarga de la interacción entre las clases.	Lexer Parser AST

Clase AST

Tabla 14 Descripción de la CRC AST.

Responsabilidades	Clases relacionadas
Clase base del árbol de sintaxis abstracta, ella y sus clases derivadas se encargan del chequeo semántico y la ejecución gráfica del código. Chequea errores semánticos.	Parser Compiler

Clase MainForm

Tabla 15 Descripción de la CRC MainForm.

Responsabilidades	Clases relacionadas
Graficar_Click(): Inicia la animación de la ejecución del código introducido. **MainForm_Load():** Carga los componentes visuales y le asigna valores por defecto a sus propiedades.	Compiler

Fase de Implementación.

4.3 Gramática utilizada.

(Ver anexo 2).

4.4 Archivo de especificación utilizado en el generador de analizadores (ANTLR).

(Ver anexo 3).

4.5 Historias de usuarios divididas en tareas.

Cada una de estas historias de usuario se transforma en tareas que luego son desarrolladas por los programadores, dentro del equipo de desarrollo, aplicando la práctica de la programación en parejas.

Tabla 16 Historias de Usuario divididas por tareas.

Nombre Historia de Usuario	Tarea
Análisis Léxico.	1- Definir los tokens en el formato ANTLR a partir de la gramática utilizada.
Análisis Sintáctico.	1- Definir el análisis sintáctico por cada regla de producción en el formato de ANTLR. 2- Definir en el fichero de especificación utilizado por ANTLR el conjunto de reglas de producción. 3- Crear jerarquía del árbol de sintaxis abstracta.

Análisis Semántico.	1- Implementar la semántica del árbol de sintaxis abstracta.
Animar resultados.	1- Ejecutar visualmente los resultados a partir del árbol de sintaxis abstracta.
Salvar y Cargar Código.	1- Diseñar la Interfaz gráfica de la herramienta. 2- Implementar las operaciones salvar y guardar código.
Tratar errores.	1- Recopilar los errores durante las tres fases de la compilación del código. 2- Mostrar errores.

Tabla 17 Tarea: Definir los tokens en el formato ANTLR a partir de la gramática utilizada.

Tarea	
Número de tarea: 1	**Número de la Historia de Usuario:** 1
Nombre de la tarea: Definir los *tokens* en el formato ANTLR a partir de la gramática utilizada.	
Tipo de tarea: Desarrollo.	
Programador responsable: Efren Olamendiz Miqueli.	
Descripción: Definir en el fichero de especificación de ANTLR el conjunto de expresiones regulares y caracteres que derivarán en *tokens*.	

Tabla 18 Tarea: Definir el análisis sintáctico por cada regla de producción en el formato de ANTLR.

Tarea	
Número de tarea: 1	**Número de la Historia de Usuario:** 2
Nombre de la tarea: Definir el análisis sintáctico por cada regla de producción en el formato de ANTLR.	
Tipo de tarea: Desarrollo.	
Programador responsable: Efren Olamendiz Miqueli	
Descripción: Definir la gramática en un fichero de especificación reconocido por ANTLR. Agregar las opciones requeridas por ANTLR en dicho fichero.	

Tabla 19 Tarea: Definir en el fichero de especificación utilizado por ANTLR el conjunto de reglas de producción.

Tarea	
Número de tarea: 2	**Número de la Historia de Usuario: 2**
Nombre de la tarea: Definir en el fichero de especificación utilizado por ANTLR el conjunto de reglas de producción.	
Tipo de tarea: Desarrollo.	
Programador responsable: Efren Olamendiz Miqueli.	
Descripción: Definir, a partir de las asociaciones de los *tokens,* las instrucciones en lenguaje **C#** que se utilizarán. Aquí se definen los terminales, no terminales y las reglas de producción necesarias que permitirán chequear la sintaxis de cada instrucción.	

Tabla 20 Tarea: Crear jerarquía del árbol de sintaxis abstracta.

Tarea	
Número de tarea: 3	**Número de la Historia de Usuario: 2**
Nombre de la tarea: Crear jerarquía del árbol de sintaxis abstracta.	
Tipo de tarea: Desarrollo.	
Programador responsable: Efren Olamendiz Miqueli.	
Descripción: Crear la jerarquía de clases en correspondencia con la gramática definida. En el fichero de especificación para cada regla de producción devolver un objeto de la clase correspondiente permitiendo que se vaya generando el árbol de sintaxis abstracta.	

Tabla 21 Tarea: Implementar la semántica del árbol de sintaxis abstracta.

Tarea	
Número de tarea: 1	**Número de la Historia de Usuario: 3**
Nombre de la tarea: Implementar la semántica del árbol de sintaxis abstracta.	
Tipo de tarea: Desarrollo.	
Programador responsable: Efren Olamendiz Miqueli y Jorge Manuel Ramy.	
Descripción: Implementar, para cada clase de la jerarquía creada anteriormente, el chequeo semántico correspondiente a la instrucción que representa.	

Tabla 22 Tarea: Ejecutar visualmente los resultados a partir del árbol de sintaxis abstracta.

Tarea	
Número de tarea: 1	**Número de la Historia de Usuario:** 4
Nombre de la tarea: Ejecutar visualmente los resultados a partir del árbol de sintaxis abstracta.	
Tipo de tarea: Desarrollo.	
Programador responsable: Jorge Manuel Ramy.	

Descripción: Si el código es válido en la sección de animación comienza una presentación mediante escenarios mostrando los valores, resultados y estado de cada línea del código introducido.

Tabla 23 Tarea: Diseñar la interfaz gráfica de la herramienta.

Tarea	
Número de tarea: 1	**Número de la Historia de Usuario:** 5
Nombre de la tarea: Diseñar la Interfaz gráfica de la herramienta.	
Tipo de tarea: Desarrollo	
Programador responsable: Jorge Manuel Ramy.	
Descripción: Crear una interfaz amigable que permita al usuario introducir código y ver los resultados de la ejecución del mismo.	

Tabla 24 Tarea: Implementar las operaciones salvar y cargar código.

Tarea	
Número de tarea: 2	**Número de la Historia de Usuario:** 5
Nombre de la tarea: Implementar las operaciones salvar y guardar código.	
Tipo de tarea: Desarrollo.	
Programador responsable: Efren Olamendiz Miqueli.	
Descripción: Se debe permitir al usuario guardar el código introducido previamente en un dispositivo de almacenamiento así como poder abrir el código almacenado.	

Tabla 25 Tarea: Recopilar errores.

Tarea	
Número de tarea: 1	**Número de la Historia de Usuario:** 6
Nombre de la tarea: Recopilar los errores durante las 3 fases de la compilación del código.	
Tipo de tarea: Desarrollo	
Programador responsable: Jorge Manuel Ramy	
Descripción: Se deben recopilar los errores ocurridos en cada fase en el orden en que se producen.	

4.6 Diagramas de presentación.

Las interfaces de usuario (**IU**) son otro elemento importante en la fase de diseño, dado que le dan una visión general al usuario sobre el sistema. La herramienta propuesta cuenta entre sus requisitos el diseño de una interfaz cómoda y sencilla.

4.6.1 Interfaz de usuario ¨Pantalla principal¨.

Cuando el usuario ejecuta la herramienta de apoyo a la asignatura de **IP** aparece la pantalla principal. **(Ver anexo 4)**

4.6.2 Interfaz de usuario ¨Resultado de la compilación¨.

Cuando el usuario introduce algún código en el área de interacción y presiona el botón compilar se muestra en la pestaña consola un resumen con la cantidad de errores detectados. **(Ver anexo 5)**

4.6.3 Interfaz de usuario ¨Errores¨.

Cuando el código introducido por el usuario presenta errores, en la pestaña con igual nombre aparecerán enumerados, mostrando además su ubicación. **(Ver anexo 6)**

4.6.4 Interfaz de usuario ¨Preparando escenario¨.

Si el código introducido en el área de interacción no presenta errores y el usuario presiona el botón graficar aparece en el área de animación un mensaje indicando que se está preparando la animación. **(Ver anexo 7)**

4.6.5 Interfaz de usuario ¨Animar resultados¨.

Después que se termina de preparar el escenario se comienza con la animación del código introducido por el usuario. **(Ver anexo 8)**

4.7 Fase de pruebas.

La metodología **XP** divide las pruebas en dos grupos: pruebas unitarias y pruebas de aceptación. Las pruebas unitarias, desarrolladas por los programadores, son las encargadas de verificar el código; y las pruebas de aceptación son las destinadas a evaluar si al final de una iteración se obtuvo la funcionalidad requerida, además de comprobar que dicha funcionalidad sea la esperada por el cliente. (Beck, Kent. 2000).

4.7.1 Pruebas unitarias

Microsoft Visual Studio 2010 permite realizar pruebas unitarias de una forma sencilla y eficiente. Éstas son utilizadas para comprobar que un método concreto del código funciona correctamente, verificar las regresiones y otros aspectos de interés por parte del programador. (28)

Características de una buena prueba unitaria:

- Se deben poder ejecutar sin necesidad de intervención manual. Esta característica posibilita que se pueda automatizar su ejecución.
- Tienen que poder repetirse tantas veces como se desee. Por este motivo, la rapidez de las mismas tiene un factor clave.
- Deben cubrir casi la totalidad del código de la aplicación.
- La ejecución de una prueba no puede afectar la ejecución de otra. Después de su ejecución el entorno debería quedar igual que estaba antes de realizarse la misma. (28)

Visual Studio 2010 permite crear un proyecto de pruebas seleccionando la parte del código que por su complejidad es de interés probar. Cuando se crea el nuevo proyecto en el explorador de soluciones aparecen dos nuevos archivos **Sample.vsmdi** y **LocalTestRun.testtrunconfig**, el primero permite acceder al editor de la lista de pruebas y el segundo contiene la configuración de algunos aspectos relacionados con la ejecución de las pruebas. Una vez realizada la prueba se ejecuta y en el editor de la lista, se exponen los resultados de todas las pruebas realizadas. A continuación se muestran capturas de pantalla de una de las pruebas realizadas al método **Scan()** de la clase **Scanner.**

```
[TestMethod()]
public void ScanTest()
{
    SourceStream source = new StringSourceStream("int"); // TODO: Initialize to an appropriate value
    SymbolsTable symbolsTable = new SymbolsTable(); // TODO: Initialize to an appropriate value
    Scanner target = new Scanner(source, symbolsTable); // TODO: Initialize to an appropriate value
    Token expected = new Token(TokenKind.Int,"int", new SourcePosition(0,2,1)); // TODO: Initialize to an appropriate value
    Token actual;
    actual = target.Scan();
    Assert.AreEqual(expected.Kind, actual.Kind);
    Assert.AreEqual(expected.Lexeme, actual.Lexeme);
    Assert.AreEqual(expected.Position.Line, actual.Position.Line);

}
```

Figura 9 Prueba realizada al método Scan comprobando que devuelve el token correcto.

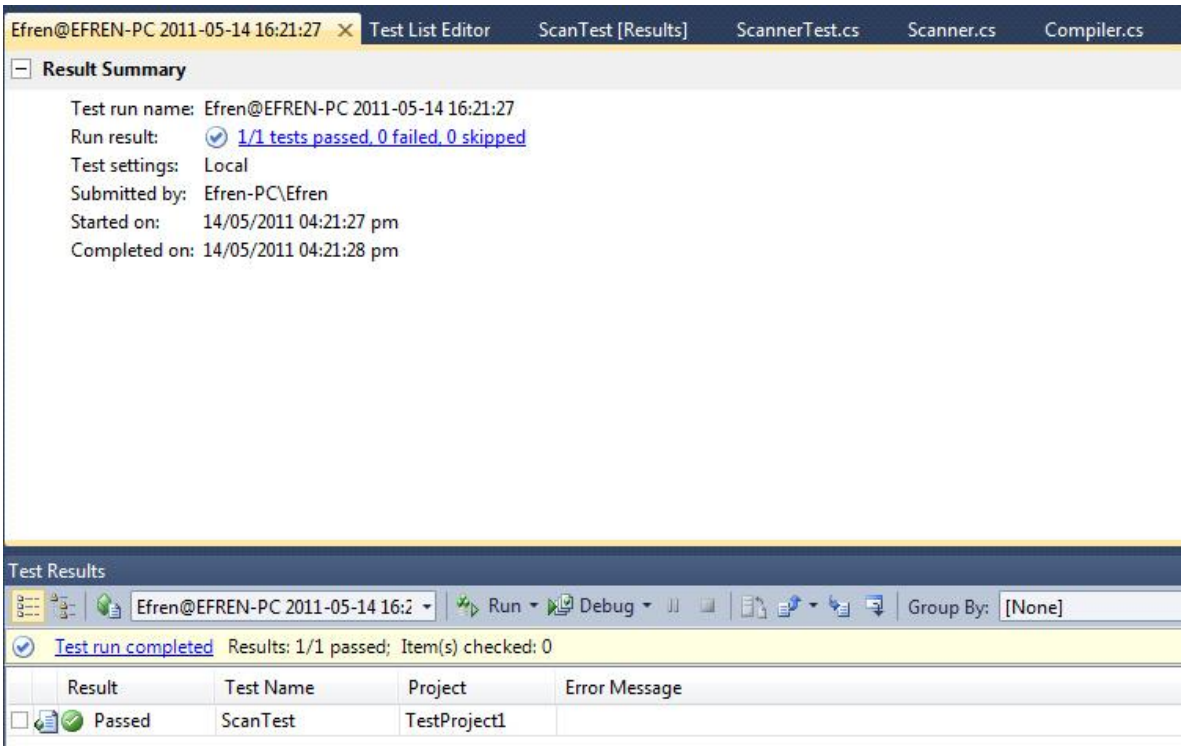

Figura 10 resultado de la prueba de la figura 14.

A continuación se muestran los resultados de las pruebas unitarias realizadas a una selección de los métodos más significativos.

Tabla 26 Resultados de las pruebas unitarias.

Métodos.	Iteraciones.	Pruebas realizadas.	Respuesta esperada.	Respuestas incorrectas.
IsDigit.	1era.	1	1	0
	2da.	1	1	0
Scan.	1era.	3	2	1
	2da.	3	3	0
IsLetter.	1era.	1	0	1
	2da.	1	1	0
IsSeparator.	1era.	1	1	0
	2da.	1	1	0
ParseAssigment.	1era.	6	4	2
	2da.	6	6	0
Parse.	1era.	5	4	1
	2da.	5	5	0
ParseDeclaration.	1era.	10	9	1
	2da.	10	10	0
ParseIfThenElse.	1era.	19	16	3
	2da.	19	19	0

4.7.2 Pruebas de aceptación.

Las pruebas de aceptación son pruebas de caja negra, son definidas por el cliente, aseguran que las funcionalidades del sistema cumplan con lo que se espera de ellas, marcan el camino a seguir indicando las funcionalidades más importantes, permiten que el cliente sepa cuando el sistema está funcionando, además que los programadores conozcan que le resta por hacer. Tienen una importancia crítica para el éxito de una iteración. Estas pruebas se crean a partir de las **HU** y durante las iteraciones las **HU** seleccionadas son traducidas a pruebas de aceptación. (29)

Tabla 27 Caso de prueba ¨Compilar código¨

Caso de prueba de aceptación	
Código: HU1-P1, HU2-P1,HU3-P1	**No. HU: 1,2,3**
Nombre: Compilar código.	
Descripción: Prueba que permite verificar el análisis léxico, sintáctico y semántico.	
Condiciones de ejecución: El usuario debe introducir código en el área de interacción.	
Entrada/Pasos de ejecución: El usuario introduce código en lenguaje **C#** y cuando activa el botón compilar se le realiza el análisis léxico, sintáctico y semántico al código introducido.	
Resultado esperado: El código no presenta errores.	
Evaluación de prueba: Prueba satisfactoria	

Tabla 28 Caso de prueba ¨Animar resultados.¨

Caso de prueba de aceptación	
Código: HU4-P1	**No. HU: 4**
Nombre: Animar resultados	
Descripción: Prueba para controlar las animaciones.	
Condiciones de ejecución: El código introducido por el usuario debe estar libre de errores.	
Entrada/Pasos de ejecución: El usuario activa el botón.	
Resultado esperado: Comienza la animación línea por línea del código introducido.	
Evaluación de prueba: Prueba satisfactoria	

Tabla 29 Caso de prueba ¨Velocidad de animación.¨

Caso de prueba de aceptación	
Código: HU4-P2	**No. HU: 4**
Nombre: Velocidad Animación	
Descripción: Prueba para controlar la velocidad de la animación.	
Condiciones de ejecución: La animación debe haber iniciado.	
Entrada/Pasos de ejecución: El usuario ajusta la velocidad de la animación en el control de velocidad ubicado en la parte inferior de la ventana.	
Resultado esperado: Se modificó la velocidad de la animación.	
Evaluación de prueba: Prueba satisfactoria	

Tabla 30 Caso de prueba ¨Cargar código.¨

Código: HU5-P1	**No. HU: 5**
Nombre: Cargar código	
Descripción: Prueba que permite comprobar si un código guardado es cargado correctamente.	
Condiciones de ejecución: Debe existir algún proyecto guardado con anterioridad.	
Entrada/Pasos de ejecución: El usuario activa el botón abrir y selecciona el archivo deseado.	
Resultado esperado: El código guardado es cargado en el source.	
Evaluación de prueba: Prueba satisfactoria	

Tabla 31 Caso de prueba ¨Salvar código¨.

Código: HU5-P2	**No. HU: 5**
Nombre: Salvar código	
Descripción: Prueba que permite comprobar si un código introducido por el usuario es almacenado.	
Condiciones de ejecución: Debe introducir código en lenguaje **C#** en el source.	
Entrada/Pasos de ejecución: El usuario activa el botón guardar y selecciona la dirección donde desea guardar el proyecto.	
Resultado esperado: El código guardado correctamente.	
Evaluación de prueba: Prueba satisfactoria	

Tabla 32 Caso de prueba ¨Tratar errores¨.

Código: HU6-P1	No. HU: 6
Nombre: Tratar errores	
Descripción: Prueba que permite comprobar si los errores encontrados durante el proceso de compilación son identificados correctamente.	
Condiciones de ejecución: Debe introducir código en lenguaje **C#** en el source.	
Entrada/Pasos de ejecución: El usuario activa el botón compilar.	
Resultado esperado: En la pestaña error aparecerá una lista con los errores identificados durante el proceso de compilación, indicando la línea donde se encuentra el error y el tipo de error identificado.	
Evaluación de prueba: Prueba satisfactoria	

Resultados de las pruebas de aceptación:

En la primera iteración se detectaron errores de funcionalidad y ortográficos:

- Errores ortográficos en la interfaz.
- El botón borrar no limpia el área de interacción.
- Se retrasa el proceso de animación de las operaciones aritméticas.

En la segunda iteración se detectaron errores de validación, generándose las siguientes no conformidades:

- El botón borrar se activa sin introducir código en el área de interacción con el usuario.
- El botón graficar no limpia el área de graficación al iniciar una nueva animación.

En la tercera iteración se solucionaron todas las deficiencias detectadas en las iteraciones anteriores.

Tabla 33 Resultados de las pruebas de aceptación.

Iteración	No conformidades	Cerradas	No Proceden
1era	7	7	0
2da	2	2	0
3era	0	0	-

4.8 Conclusiones parciales

En el capítulo se elaboraron las tarjetas CRC y la gramática a utilizar describiendo y delimitando el lenguaje de programación seleccionado, además se trazaron las tareas a desarrollar por cada historia de usuario, por último, se le realizaron dos iteraciones de pruebas unitarias y tres iteraciones de pruebas de aceptación corrigiéndole las deficiencias encontradas durante las mismas, con el objetivo de brindarle al cliente una herramienta que cumpla con las necesidades especificadas.

Conclusiones.

El presente trabajo se propuso como objetivo desarrollar una herramienta de apoyo a la enseñanza de la asignatura de Introducción a la programación en lenguaje C# que permita la representación visual de las operaciones aritméticas, condicionales y estructuras repetitivas. En correspondencia con el mismo se arribó a las siguientes conclusiones.

- Se realizó una investigación de las técnicas de enseñanza de la programación en el ámbito internacional y nacional, además del estudio de las herramientas, lenguajes y metodologías existentes, lo que permitió identificar funcionalidades que el sistema debe cumplir y diseñar un software con una interfaz agradable, capaz de facilitar la comprensión de conceptos básicos de la programación.
- Se realizó el análisis, diseño, implementación y pruebas de la herramienta de representación visual, permitiendo la obtención de un producto robusto, funcional y confiable.
- Con el desarrollo de la herramienta de representación visual se obtiene un producto capaz de apoyar la enseñanza de la programación en la asignatura de **IP** de la carrera de informática.

Recomendaciones.

Por la importancia de este trabajo de investigación como apoyo a la enseñanza de los contenidos que se imparten en la asignatura de IP, se considera:

- Debe ser divulgado por las instancias centrales encargadas del quehacer científico técnico de la universidad y que el mismo sea utilizado en las facultades, como apoyo bibliográfico para investigaciones futuras relacionadas con el tema, para su presentación en eventos científicos nacionales e internacionales con carácter informático y pedagógico.
- Se recomienda agregarle contenidos de otras asignaturas relacionadas tales como: P1 (Programación 1), P2 (Programación 2).
- Hacer extensiva la utilización de la herramienta en otros centros de enseñanza superior para apoyar la asignatura de programación.
- Migrar la solución propuesta hacia una plataforma de software libre.

Referencias Bibligráficas.

1. **Sara, Tena Garcia.** Desarrollo de un escenario en el entorno greenfoot. España, Universidad Carlos III : s.n., 2009.

2. **Guillermo, Jiménez Díaz.** Entornos virtuales basados en técnicas de aprendizaje activo para la enseñanza de la orientación a objetos. Madrid : s.n., 2008.

3. **Eugenio, Herrera Hernandez.** *Tecnicas de Compilacion I.* La Habana : facultad fisica- matemática dpto. cibernética-matematica, 1982.

4. **Wang, W.** *Beginning Programming.* United States : Wiley Publishing, 2007.

5. **Albarrán, M. Gómez.** *Una revisión de métodos pedagógicos innovadores para la enseñanza de la programación.* Madrid : s.n., 2008.

6. **Rosa, Romero Gomez.** *Desarrollo de un escenario basado en la herramienta Greenfoot para el apoyo de la enseñanza temprana de la Programación Orientada a Objetos.* Madrid : s.n., 2009.

7. **Michael, Barner J David y Kolling.** *Objects First with Java.* 2008.

8. **Nyko, Miller.** Jeliot 3. *Jeliot 3.* [En línea] [Citado el: 25 de 1 de 2010.] http://cs.joensuu.fi/~jeliot/pub.php.

9. **Carnegie Mellon University.** Alice. *Sitio oficial del software Alice.* [En línea] Oracle, 2008. [Citado el: 23 de Noviembre de 2010.]

10. **Jacomson, Ivar, Booch, Grady y Rumbaugh, James.** *El proceso unificado de desarrollo de Software.* s.l. : Editorial Félix Varela, 1999.

11. **Gerardo, Fernández Escribano.** Introducción a Extreme Programming. [En línea] [Citado el: 2011 de Noviembre de 15.] http://www.dsi.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf.

12. **José H. Canós, Patricio Letelier y Carmen Penadés.** Métodologías Ágiles en el Desarrollo de Software. [En línea] [Citado el: 14 de 12 de 2010.] http://www.willydev.net/descargas/prev/TodoAgil.pdf..

13. **Alejandro, Aguilar Sierra.** *Introducción a la Programación Extrema.* Mexico : Universidad Autónoma de México, 2006.

14. Comparación de Herramientas de Modelado UML. *Enterprise Architect y Rational Rose.* [En línea] febrero de 2009. [Citado el: 2011 de Noviembre de 18 .] http://www.apexnet.com.ar/index.php/news/main/38/event=view..

15. Rational Rose: Procedimientos básicos para desarrollar un proyecto con UML. [En línea] IBM, febrero de 2007. [Citado el: 2011 de Noviembre de 5.] http://www.ibm.com/software/awdtools/developer/rose/.

16. Visual Paradigm. [En línea] Visual Paradigm, Diciembre de 2010. [Citado el: 2011 de Noviembre de 15.] http://www.visual-paradigm.com/product/vpuml/.

17. **Daconta., Michael C.** Pointers and Dynamic Memory Management Edit. John Wiley & Sons. *http://www.zator.com/Cpp/E1_2.htm.* [En línea] 26 de 03 de 2008. [Citado el: 22 de 10 de 2012.] http://www.zator.com/Cpp/E1_2.htm..

18. **Gonzales., Álvarez Marañón.** ¿Qué es Java? Sitio oficial de la IEC. [En línea] 2008 de 11 de 4. [Citado el: 2011 de 10 de 28.] http://www.iec.csic.es/criptonomicon/java/quesjava.html#top..

19. Conceptos fundamentales sobre la programación en C#. [En línea] [Citado el: 2011 de 11 de 2.] http://www.iec.csic.es/criptonomicon/java/quesjava.html#top..

20. **Microsoft.** Lo nuevo de Microsoft. [En línea] Microsoft. [Citado el: 6 de Diciembre de 2011.] http://msdn.microsoft.com/es-es/library/ms172620(v=vs.80).aspx.

21. **Cristóbal, González Almirón.** *Desarrollo de aplicaciones .NET con SharpDevelop.* Noviembre : Barrabes Editorial, 26.

22. **Peter, Bodnar Jan.** go-mono. [En línea] 2 de enero de 2009. [Citado el: 21 de enero de 2011.] http://monodevelop/tutorials/gtksharptutorial/.

23. **Patrice, Pelland.** *Microsoft® Visual C#® 2008 Express Edition: Build a Program Now.* 2008. ISBN 13:9780735625426.

24. **Pelland Patrice, Pare Pascal, Haines Ken.** *Moving to Microsoft Visual Studio 2010.* Washington : Microsoft Press, 2011. ISBN:2010934433.

25. **Roberto, Canales Mora.** Autentia. [En línea] 23 de 12 de 2003. [Citado el: 13 de 02 de 1011.] http://www.adictosaltrabajo.com//.

26. **Ruben, Collado.** RCH. [En línea] Enero de 2010. [Citado el: 26 de Febrero de 2010.] http://www.rubencollado.net/v2/index.php?option=com_content&view=article&id=6:artdocumentacionguiaestiloc&catid=4:catdocumentacionguiaestilo&Itemid=4.

27. **Amber, Scot W.** Agile Modeling. *Effective Practices for Modeling and Documentation.* [En línea] 2 de marzo de 2010. [Citado el: 16 de febrero de 2011.] http://www.agilemodeling.com/.

Bibliografía

1. **Sara, Tena Garcia.** Desarrollo de un escenario en el entorno greenfoot. España, Universidad Carlos III : s.n., 2009.

2. **Guillermo, Jiménez Díaz.** Entornos virtuales basados en técnicas de aprendizaje activo para la enseñanza de la orientación a objetos. Madrid : s.n., 2008.

3. **Eugenio, Herrera Hernandez.** *Tecnicas de Compilacion I.* La Habana : facultad fisica- matemática dpto. cibernética-matematica, 1982.

4. **Wang, W.** *Beginning Programming.* United States : Wiley Publishing, 2007.

5. **Albarrán, M. Gómez.** *Una revisión de métodos pedagógicos innovadores para la enseñanza de la programación.* Madrid : s.n., 2008.

6. **Rosa, Romero Gomez.** *Desarrollo de un escenario basado en la herramienta Greenfoot para el apoyo de la enseñanza temprana de la Programación Orientada a Objetos.* Madrid : s.n., 2009.

7. **Michael, Barner J David y Kolling.** *Objects First with Java.* 2008.

8. **Nyko, Miller.** Jeliot 3. *Jeliot 3.* [En línea] [Citado el: 25 de 1 de 2010.] http://cs.joensuu.fi/~jeliot/pub.php.

9. **Carnegie Mellon University.** Alice. *Sitio oficial del software Alice.* [En línea] Oracle, 2008. [Citado el: 23 de Noviembre de 2010.]

10. **Jacomson, Ivar, Booch, Grady y Rumbaugh, James.** *El proceso unificado de desarrollo de Software.* s.l. : Editorial Félix Varela, 1999.

11. **Gerardo, Fernández Escribano.** Introducción a Extreme Programming. [En línea] [Citado el: 2011 de Noviembre de 15.] http://www.dsi.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf.

12. **José H. Canós, Patricio Letelier y Carmen Penadés.** Métodologías Ágiles en el Desarrollo de Software. [En línea] [Citado el: 14 de 12 de 2010.] http://www.willydev.net/descargas/prev/TodoAgil.pdf..

13. **Alejandro, Aguilar Sierra.** *Introducción a la Programación Extrema.* Mexico : Universidad Autónoma de México, 2006.

14. Comparación de Herramientas de Modelado UML. *Enterprise Architect y Rational Rose.* [En línea] febrero de 2009. [Citado el: 2011 de Noviembre de 18 .] http://www.apexnet.com.ar/index.php/news/main/38/event=view..

15. Rational Rose: Procedimientos básicos para desarrollar un proyecto con UML. [En línea] IBM, febrero de 2007. [Citado el: 2011 de Noviembre de 5.] http://www.ibm.com/software/awdtools/developer/rose/.

16. Visual Paradigm. [En línea] Visual Paradigm, Diciembre de 2010. [Citado el: 2011 de Noviembre de 15.] http://www.visual-paradigm.com/product/vpuml/.

17. **Daconta., Michael C.** Pointers and Dynamic Memory Management Edit. John Wiley & Sons. *http://www.zator.com/Cpp/E1_2.htm.* [En línea] 26 de 03 de 2008. [Citado el: 22 de 10 de 2012.] http://www.zator.com/Cpp/E1_2.htm..

18. **Gonzales, Álvarez Marañón.** ¿Qué es Java? Sitio oficial de la IEC. [En línea] 2008 de 11 de 4. [Citado el: 2011 de 10 de 28.] http://www.iec.csic.es/criptonomicon/java/quesjava.html#top.

19. Conceptos fundamentales sobre la programación en C#. [En línea] [Citado el: 2011 de 11 de 2.] http://www.iec.csic.es/criptonomicon/java/quesjava.html#top..

20. **Microsoft.** Lo nuevo de Microsoft. [En línea] Microsoft. [Citado el: 6 de Diciembre de 2011.] http://msdn.microsoft.com/es-es/library/ms172620(v=vs.80).aspx.

21. **Cristóbal, González Almirón.** *Desarrollo de aplicaciones .NET con SharpDevelop.* Noviembre : Barrabes Editorial, 26.

22. **Peter, Bodnar Jan.** go-mono. [En línea] 2 de enero de 2009. [Citado el: 21 de enero de 2011.] http://monodevelop/tutorials/gtksharptutorial/.

23. **Patrice, Pelland.** *Microsoft® Visual C#® 2008 Express Edition: Build a Program Now.* 2008. ISBN 13:9780735625426.

24. **Pelland Patrice, Pare Pascal, Haines Ken.** *Moving to Microsoft Visual Studio 2010.* Washington : Microsoft Press, 2011. ISBN:2010934433.

25. **Roberto, Canales Mora.** Autentia. [En línea] 23 de 12 de 2003. [Citado el: 13 de 02 de 1011.] http://www.adictosaltrabajo.com//.

26. **Ruben, Collado.** RCH. [En línea] Enero de 2010. [Citado el: 26 de Febrero de 2010.] http://www.rubencollado.net/v2/index.php?option=com_content&view=article&id=6:artdocumentacionguiaestiloc&catid=4:catdocumentacionguiaestilo&Itemid=4.

27. **Amber, Scot W.** Agile Modeling. *Effective Practices for Modeling and Documentation.* [En línea] 2 de marzo de 2010. [Citado el: 16 de febrero de 2011.] http://www.agilemodeling.com/.

29. *¿Hacia dónde vamos?* **Soto, L. D.** 20, Barcelona : s.n., 2008, Vol. Software Guru.

30. The British Standards Institution. [En línea] BSI, 2009. [Citado el: 25 de 11 de 2010.] http://www.bsigroup.com.mx/es-mx/Auditoria-y-Certificacion/Sistemas-de-Gestion/De-un-vistazo/Que-son-los-sistemas-de-gestion.

31. **Valdés, Damián Pérez.** Maestros de la web. [En línea] [Citado el: 28 de 11 de 2010.] http://www.maestrosdelweb.com/principiantes/los-diferentes-lenguajes-de-programacion-para-la-web.

32. Conceptos básicos del servidor web. [En línea] [Citado el: 1 de 12 de 2010.] http://www.cibernetia.com/manuales/instalacion_servidor_web/1_conceptos_basicos.php.

33. The Apache Software Foundation. [En línea] [Citado el: 4 de 12 de 2010.] http://www.apache.org.

34. Internet Information Services. [En línea] [Citado el: 6 de 12 de 2010.] http://www.microsoft.com/spain/windowsserver2003/technologies/webapp/iis.mspx.

35. cavsi. [En línea] [Citado el: 6 de 12 de 2010.] http://www.cavsi.com/preguntasrespuestas/que-es-un-sistema-gestor-de-bases-de-datos-o-sgbd.

36. **Daniel, Pecos.** PostgreSQL. [En línea] [Citado el: 9 de 12 de 2010.] http://danielpecos.com/docs/mysql_postgres/x15.html.

37. **Mora, Marc Gibert Ginestà.Oscar Pérez.** [En línea] [Citado el: 12 de 12 de 2010.] http://ocw.uoc.edu/computer-science-technology-and-multimedia/bases-de-datos/bases-de-datos/P06_M2109_02152.pdf.

38. Todo expertos . [En línea] [Citado el: 12 de 12 de 2010.] http://www.todoexpertos.com/categorias/tecnologia-e-internet/bases-de-datos/oracle/respuestas/14706/vetajas-y-desventajas..

39. **Ivar Jacobson, Grady Booch y James Rumbaugh.** El Proceso Unificado de Desarrollo. *La guía completa del Proceso Unificado.* Madrid Addison Wesley : s.n., 2000.

40. Morón, Universidad. Metodologías Ágiles. [En línea] [Citado el: 15 de 12 de 2010.] http://noqualityinside.com/nqi/nqifiles/Metodologias_Agiles.pdf.

41. **Calderon, Anyelin.** Sistematizando Aprendizajes. [En línea] [Citado el: 15 de 12 de 2010.] http://anyelincalderon.blogspot.com/2010/04/algo-sobre-metodologias-agiles.html..

42. **Mendoza Sanchez, María A.** [En línea] 6 de 7 de 2004. [Citado el: 16 de 12 de 2010.] http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html .

43. **GERSBACH, M.** Principales características de Drupal. [En línea] [Citado el: 16 de 12 de 2010.] http://www.factoriadigital.com/ir.php/a/aplicacionesweb/drupal..

44. Tufuncion.com . [En línea] 2007. [Citado el: 6 de 1 de 2011.] http://www.tufuncion.com/zend-studio..

Glosario de Términos.

API: (Application Program Interface). Conjunto de convenciones internacionales que definen cómo debe invocarse una determinada función de un programa desde una aplicación. Cuando se intenta estandarizar una plataforma, se estipulan unos APIs comunes a los que deben ajustarse todos los desarrolladores de aplicaciones.

UML:(Unified Modeling Language) Lenguaje Unificado de Modelado, es un lenguaje que proporciona un vocabulario y reglas para permitir una comunicación.

Lex: es un programa que genera analizadores léxicos ("scanners" o "lexers"). Se utiliza comúnmente con el generador de análisis sintáctico yacc.

ANTLR: Se trata de un generador de analizadores léxico y sintácticos.

TIC: (Tecnologías de la Informática y las Comunicaciones) son herramientas que amplían nuestras capacidades físicas y mentales así como las posibilidades de desarrollo social.

GRASP: (General Responsibility Assignment Software Patterns) en español patrones generales de software para asignar responsabilidades, el nombre se eligió para indicar la importancia de captar (grasping) estos principios, si se quiere diseñar eficazmente el software orientado a objetos.

RAM :(Random Access Memory Module) se compone de uno o más chips y se utiliza como memoria de trabajo para programas y datos. Es un tipo de memoria temporal que pierde sus datos cuando se queda sin energía (por ejemplo, al apagar la computadora), por lo cual es una memoria volátil.

RUP: (Rational Unified Process o Proceso Unificado de Desarrollo de Software) Su objetivo es garantizar la producción de alta calidad de software que satisface las necesidades de sus usuarios finales, en un previsible calendario y el presupuesto.

Anexos

Anexo 1: Análisis y Tabulación de la encuesta: "Uso de técnicas de soporte a la asignatura de Introducción a la Programación".

Estimado estudiante el presente cuestionario es parte de una investigación encaminada a la creación de una herramienta que contribuye a la preparación en la asignatura de introducción a la programación para lo cual solicitamos que tus respuestas sean totalmente veraces.

1. ¿Utilizó alguna herramienta que le sirviera de complemento a las clases de introducción a la programación que recibió?

   Si __ No__ No se__

   Si su respuesta fue afirmativa siga contestando.

2. ¿La herramienta que utilizó le facilitó el desarrollo de sus actividades?

   Nunca___ Algunas veces___ Casi siempre___ Siempre___

3. ¿Están incluidos ejercicios de traceo de código en la asignatura de Introducción a la Programación?

   Si___ No___ No se___

4. ¿Se realizan frecuentemente ejercicios de este tipo?

   Si___ No___ No se___

5. ¿Considera que para mejorar esta actividad se necesita de una herramienta que muestre visualmente el resultado del código?

   Si___ No___

Criterios	**Total**	**Si**	**No**	**No se**	**%**
Utilización de herramienta de complemento a la	200	146	30	24	73

asignatura IP.					
Su uso facilitó sus actividades	200	32	168	_	16
Inclusión de ejercicios de traceo de código en la asignatura de IP.	200	198	0	2	99
Frecuencia de realización de ejercicios de traceo de código.	200	171	10	19	85,5
Se necesita de una herramienta que muestre visualmente el resultado del código.	200	194	0	6	97

Anexo 2:

La gramática creada se define como un cuádruplo **G= {N, ∑, P, S}** donde:

N: Tipo, Id, Programa, Enun, Bloque, Exprif, ExprParen, Exprfor, Controlfor, ListExpr, Exprwhile, Declaracion, DecsVar, DecVar, IniVar, IniArr, Exprdowhile, Exprswitch, Bloqueswitch, Caseswitch, Expresion, Exprcond, Expror, Exprand, ExprIg, ExprRel, Oprel, Exprarit, Expmult, Expunaria, Expunaria2, Primario, CreArr, Literal, Opasig, IniArrder, Elseif, Exprcondder, Exprder, Exprandder, Exprigder, Exprrelder, Expmultder, Expraritder, Expmultder, Expunariader, Expun2der, Expun2der2, Entero, Cadena, Char.

∑: (,), [,], {, }, ;, :; “”, ‘, \, ,, ?, !, >, <, ==, !=, >=, <=, +, -, *, /, %, =, +=, -=, *=, /=, %=, ++, --, &&, ||, if, else, for, while, do, new, break, continue, switch, case, default, null, true, false, int, double, char, string, void, static, main, public.

P: Conjunto de reglas de producción:

```
Programa      ->  Enun Programa | e

Enun          ->  Exprif | Exprfor | Exprwhile | Exprdowhile
                  | Declaracion | Bloque | Exprswitch
                  | Break ; | Expresion ; | ; | Continue ;

Bloque        ->  { Programa }

Exprif        ->  if  ExprParen Enun Elseif

Elseif        ->  else Enun | e
```

```
ExprParen      ->  ( Expresion )

Exprfor        -> ( Controlfor ) Enun

Controlfor     -> ListExpr; Expresion ; ListExpr

ListExpr       -> Expresion , ListExpr | Expresion

Exprwhile      -> while  ExprParen  Enun

Declaracion    -> Tipo DecsVar; | Tipo [] DecsVar;

Tipo           -> int | string | char | double

DecsVar       -> DecVar , DecsVar | DecVar

DecVar         -> Id = IniVar | id

IniVar         -> IniArr | Expresion

IniArr         -> { } | { IniArrder }

IniArrder     -> IniVar, IniArrder | IniVar

Exprdowhile    -> do Enun While ExprParen ;

Exprswitch     -> switch ExprParen { Exprswitchder }

 Exprswitchder-> Bloqueswitch Exprswitchder | e

Bloqueswitch  -> Caseswitch Enun

Caseswitch     -> case Expresion : | default  :

Expresion      -> Exprcond Exprder

Exprder        -> Opasig Expresion | e

Exprcond       -> Expror Exprcondder

Exprcondder    -> ? Expresion : Expresion |e

Expror         -> Exprand Exprandder

Exprandder     -> || Exprand Expresion | e

Exprand        -> Exprig Expresion

Exprigder      -> && Exprig Exprigder | e

Exprig         -> Exprrel Expresion

Exprrelder     -> = Exprrel Exprrelder | != Exprrel Expresion
```

```
                | e
Exprrel      -> Exprarit Expraritder
Expraritder  -> Oprel Exprarit Expraritder | e
Oprel        -> >=|<=|>|<
Exprarit     -> Expmult Expmultder
Expmultder   -> + Expmult Expmultder | - Expmult Expmultder
                | e
Expmult      -> Expunaria Expunariader
Expunariader -> * Expresion | / Expresion
                | % Expresion | e
Expunaria    -> + Expresion | - Expresion
                | ++ Primario | -- Primario
                | Expunaria2
Expunaria2   -> ! Expunaria | Primario Expun2der
Expun2der    -> [Expresion] Expunder2 | Expunder2
Expunder2    ->  ++ | -- | e
Primario     -> ExprParen | Literal | Id | new tipo CreArr
CreArr       -> [] IniArr | [ Expresion ]
Literal      -> Entero | Cadena | true | false
                | Char | null
Opasig       -> = | += | /= | -= | *= | %=
```

S: Programa

Anexo 3:

```
grammar SpecFile;
options
{
   language = 'CSharp';
   k=2;
   backtrack=true;
   memoize=true;

}

@header {
using IDE.AbstractSyntaxTrees;
}


//tokens

//OTROS
        PARENTESISABIERTO:'(';
        PARENTESISCERRADO:')';
        CORCHETEABIERTO:'[';
        CORCHETECERRADO:']';
        LLAVEABIERTA:'{';
        LLAVECERRADA:'}';
        PUNTOYCOMA:';';
        DOSPUNTOS:':';
        COMILLADOBLE:'"';
        COMILLASIMPLE:'\'';
        COMA:',';
        INTERROGACION:'?';
        EXCLAMACION:'!';

//OPERADORES COMPARACION
        MAYORQUE:'>';
        MENORQUE:'<';
        IGUAL:'==';
        DISTINTO:'!=';
        MAYORIGUAL:'>=';
        MENORIGUAL:'<=';


//OPERADORES ARITMETICOS
        MAS: '+';
        MENOS: '-';
        MULTIPLICACION:'*';
        DIVISION:'/';
        RESTO:'%';

//ASIGNACION
        ASIGNACION:'=';
        MASIGUAL:'+=';
        MENOSIGUAL:'-=';
        PORIGUAL:'*=';
        ENTREIGUAL:'/=';
        RESTOIGUAL:'%=';
        MASMAS:'++';
        MENOSMENOS:'--';

//OPERADORES LOGICOS
        AND:'&&';
```

```
        OR:'||';

//PALABRAS CLAVES
        PC_IF:'if';
        PC_ELSE:'else';
        PC_FOR:'for';
        PC_WHILE:'while';
        PC_DO:'do';
        PC_NEW:'new';
        PC_BREAK:'break';
        PC_CONTINUE:'continue';
        PC_SWITCH:'switch';
        PC_CASE:'case';
        PC_DEFAULT:'default';
        PC_NULL:'null';
        PC_TRUE:'true';
        PC_FALSE:'false';
        PC_STATIC: 'static';
        PC_VOID:'void';
        PC_MAIN:'Main';
        PC_FOREACH:'foreach';
        PC_IN:'in';

//TIPOS DATOS
tipo: 'int'|'string'|'float'|'char'|'bool';

//EXPRESIONES REGULARES
fragment LETRA  :       'a'..'z'|'A'..'Z';
fragment DIGITO :       '0'..'9';
ID      :       (LETRA(LETRA|DIGITO|'_')*)|('_'(LETRA|DIGITO|'_')+);
ENTERO          :       (DIGITO)+;
ESPACIOBLANCO   :   (' '| '\t'| '\r'| '\n') {$channel=HIDDEN;};

CADENA          :       COMILLADOBLE (ESC_SEQ|~('\\'|COMILLADOBLE))*
COMILLADOBLE;
CHAR            :       COMILLASIMPLE (ESC_SEQ|~('\\'|COMILLASIMPLE))?
COMILLASIMPLE;
fragment ESC_SEQ
        :       '\\'('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\');
FLOAT           :       ENTERO'.'ENTERO
        |       '.'ENTERO;


//PARSER

programa returns [Programa program]

        :  PC_STATIC PC_VOID PC_MAIN PARENTESISABIERTO PARENTESISCERRADO
           LLAVEABIERTA enun=enunciacion LLAVECERRADA {$program=enun);}
        |  PC_STATIC PC_VOID PC_MAIN PARENTESISABIERTO PARENTESISCERRADO
           lista=listaenunciaciones{$program=lista;};

listaenunciaciones returns [ListaEnunciaciones lista]
        @init{
              List<Enunciacion> enunciaciones = new List<Enunciacion>();
        }
        : (enun=enunciacion{enunciaciones.Add(enun);})*EOF{$lista=
           new ListaEnunciaciones(enunciaciones)};

enunciacion returns [Enunciacion enunciacion]
        :       opcont = opcontrol{$enunciacion = opcont;}
        |       decl=declaracion {$enunciacion = decl;}
```

```
        |       bloq=bloque {$enunciacion = bloq;}
        |       pos=PC_BREAK PUNTOYCOMA {$enunciacion =
                new ExpresionBreak(pos.CharPositionInLine,pos.Line);}
        |       exp=expresion PUNTOYCOMA {$enunciacion = exp;}
        |       PUNTOYCOMA
        |       pos=PC_CONTINUE PUNTOYCOMA {$enunciacion =
                new ExpresionContinue(pos.CharPositionInLine,pos.Line);};


opcontrol returns [OpControl opcontrol]
        :       expif = exprif {$opcontrol = expif;}
        |       expfor= exprfor {$opcontrol = expfor;}
        |       expwhile= exprwhile {$opcontrol=expwhile;}
        |       expdowhile=exprdowhile{$opcontrol=expdowhile;}
        |       expforeach=exprforeach{$opcontrol=expforeach;};


bloque returns [Bloque bloque]
        :       LLAVEABIERTA prog=programa LLAVECERRADA{$bloque =
                new Bloque(prog);};

exprif returns [ExpresionIF exprif]
        :       pos = PC_IF parteif = expresionparentesis partethen =
                enunciacion (PC_ELSE) => PC_ELSE parteelse = enunciacion
                {$exprif = new ExpresionIF(pos.CharPositionInLine, pos.Line,
                parteif, partethen, parteelse)};

fragment expresionparentesis returns [Expresion expr]
        :       PARENTESISABIERTO exp=expresion PARENTESISCERRADO{$expr =
                exp};

exprforeach returns [ExpresionForEach exprforeach]
        :       pos=PC_FOREACH PARENTESISABIERTO tip=tipo id1=ID PC_IN id2=ID
                PARENTESISCERRADO enun=enunciacion{$exprforeach = new
                ExpresionForEach(pos.CharPositionInLine,pos.Line,id1,id2,
                enun)};

exprfor returns [ExpresionFor exprfor]
        @init{
                bool isDec = false;
                }
        @after{
                $exprfor = new ExpresionFor(pos.CharPositionInLine, pos.Line,
                id.Text, expr1, expr2, expr3, enunc, isDec)
        }
        :       pos =PC_FOR PARENTESISABIERTO ('int'{isDec = true;})? id=ID
                ASIGNACION expr1=expresion PUNTOYCOMA expr2 = expresion
                PUNTOYCOMA expr3=expresion PARENTESISCERRADO
                enunc= enunciacion;


exprwhile returns [ExpresionWhile exprwhile]
        :       pos=PC_WHILE expwhile=expresionparentesis enun=enunciación
                {$exprwhile=new ExpresionWhile(pos.CharPositionInLine,
                pos.Line,expwhile, enun); };

declaracion returns [Declaracion declara]
        @init{
                bool isArray = false;
                }
```

```
		@after
			{
			if(isArray)
			{
				$declara=new DeclaracionArreglo(typ.CharPositionInLine,
				typ.Line,id.Text,typ.Text, exp);
			}
			else
			{
				$declara=new DeclaracionVariable(
				typ.CharPositionInLine, typ.Line, id.Text, typ.Text,
				exp);

			}
			}
		:	typ=tipo (CORCHETEABIERTO CORCHETECERRADO{isArray = true;})?
			(id=ID IGUAL exp=expresion | id=ID) PUNTOYCOMA;


exprdowhile returns [ExpresionDoWhile exprdowhile]
		:	pos=PC_DO enunc=enunciacion PC_WHILE cond=expresionparentesis
			PUNTOYCOMA{$exprdowhile = new ExpresionDoWhile(
			pos.CharPositionInLine, pos.Line, cond, enunc};

expresion returns [Expresion expr]
		@init{
			int op = 0;
			}
		@after
		{
			switch(op)
			{
				case 1:
				{
					$expr = new ExpresionAsignacion(
					pos.CharPositionInLine, pos.Line,expr1,expr2);
					break;
				}
				case 2:
				{
					$expr = new ExpresionMasIgual(
					pos.CharPositionInLine, pos.Line,expr1,expr2);
					break;
				}
				case 3:
				{
					$expr = new ExpresionEntreIgual(
					pos.CharPositionInLine, pos.Line,expr1,expr2);
					break;
				}
				case 4:
				{
					$expr = new ExpresionMenosIgual(
					pos.CharPositionInLine, pos.Line,expr1,expr2);
					break;
				}
				case 5:
				{
					$expr = new ExpresionPorIgual(
					pos.CharPositionInLine, pos.Line,expr1,expr2);
					break;
				}
```

```
                        case 6:
                        {
                              $expr = new ExpresionRestoIgual(
                              pos.CharPositionInLine, pos.Line,expr1,expr2);
                              break;
                        }

                        default:
                        {
                              $expr = expr1;
                              break;
                        }
                  }
            }
            :     expr1=exprcondor ((pos=ASIGNACION{op = 1;}
            |     pos=MASIGUAL{op = 2;}
            |     pos=ENTREIGUAL{op = 3;}
            |     pos=MENOSIGUAL{op = 4;}
            |     pos=PORIGUAL{op = 5;}
            |     pos=RESTOIGUAL{op = 6;}) expr2=expresion{esAsign = true;})?;

exprcondor returns [ExpresionOr expresionor]
            @init
            {
                  bool esOr = false;
            }
            @after
            {
                  if(esOr)
                        $expresionor = new ExpresionOr(pos.CharPositionInLine,
                        pos.Line, expr1, expr2);
                  else
                        $expresionor = expr1;
            }
            :     expr1=exprcondand (pos=OR expr2=exprcondor{esOr= true;})?;

exprcondand returns [ExpresionAnd expresionand]
            @init
            {
                  bool esAnd=false;
            }
            @after
            {
                   if(esAnd)
                         $expresionand = new ExpresionAnd(
                         pos.CharPositionInLine, pos.Line, expr1, expr2);
                   else
                         $expresionand=expr1;
            }
            :     expr1=exprigualdad (pos=AND expr2=exprcondand{esAnd=true;})?;

exprigualdad returns [ExpresionCondicional expresioncond]

            @init
            {
                  bool esigual=false, esdistinto = false;
            }
            @after
            {
                  if(esigual)
```

```
                        $expresioncond = new ExpresionIgual(
                        pos.CharPositionInLine, pos.Line,expr1, expr2)
                  else if(esdistinto)
                        $expresioncond = new ExpresionDistinto(
                        pos.CharPositionInLine, pos.Line,expr1, expr2)
                  else
                        $expresioncond = exprrelacional;
            }
            :     expr1=exprrelacional ((pos=IGUAL{esigual=true;}
            |     pos=DISTINTO{esdistinto=true;})expr2=exprigualdad)?;

exprrelacional returns [ExpresionComparacion exprcomp]
            @init
            {
                  bool mayorigual = false, menorigual = false, mayorque = false,
                  menorque = false;
            }
            @after
            {
                  if(mayorigual)
                        $exprcomp = new ExpresionMayorIgual(
                        pos.CharPositionInLine, pos.Line, expr1, expr2)
                  else if(menorigual)
                        $exprcomp = new ExpresionMenorIgual(
                        pos.CharPositionInLine, pos.Line, expr1, expr2)
                  else if(mayorque)
                        $exprcomp = new ExpresionMayorQue(
                        pos.CharPositionInLine, pos.Line, expr1, expr2)
                  else if(menorque)
                        $exprcomp = new ExpresionMenorQue(
                        pos.CharPositionInLine, pos.Line, expr1, expr2)
                  else
                        $exprcomp = expr1;
            }
            :     expr1=exprarit ((pos=MAYORIGUAL{mayorigual=true;}
            |     pos=MENORIGUAL{menorigual=true;}
            |     pos=MAYORQUE{mayorque=true;}
            |     pos=MENORQUE{menorque=true;}) expr2=exprrelacional)?;


exprarit returns [ExpresionAritmetica exprar]
            @init
            {
                  bool esmas=false, esmenos=false;
            }
            @after
            {
                  if(esmas)
                        $exprar= new ExpresionSuma(pos.CharPositionInLine,
                        pos.Line, expr1, expr2);
                  else if(esmenos)
                        $exprar= new ExpresionResta(pos.CharPositionInLine,
                        pos.Line, expr1, expr2);
                  else
                        $exprar = expr1;

            }
            :     expr1=expmult ((pos=MAS {esmas=true;}
            |     pos=MENOS{esmenos=true;})expr2=exprarit)?;

expmult   returns [ExpresionAritmetica exprmult]
```

```
        @init
        {
              bool mult=false, div=false, resto=false;
        }
        @after
        {
              if(mult)
                    $exprmult= new ExpresionMultiplicacion(
                    pos.CharPositionInLine, pos.Line, expr1, expr2);
              else if(div)
                    $exprmult= new ExpresionDivision(
                    pos.CharPositionInLine, pos.Line, expr1, expr2);
              else if(resto)
                    $exprmult= new ExpresionResto(pos.CharPositionInLine,
                    pos.Line, expr1, expr2);
              else
                    $exprmult = expr1;

        }
        :     expr1=expunaria ((pos=MULTIPLICACION{mult = true;}
        |     pos=DIVISION{div = true;}
        |     pos=RESTO{resto = true;})expr2=expmult)?;

expunaria returns [ExpresionUnaria exprun]
        :     pos=EXCLAMACION expr1=primario{$exprun= new
              ExpresionNot(pos.CharPositionInLine, pos.Line, expr1 );}
        |     expr2=expunaria2{$exprun=expr2;};

expunaria2 returns [ExpresionUnaria2 exprun2]
        @init{
              int op = 0, arit = 0;
        }
        @after{
                    if(op == 0)
                    {
                          if(arit == 0)
                                $exprun2 = pr;
                          else
                          {
                                if(arit == 1)
                                {
                                      $exprun2 = new ExpresionMasMas(
                                      pr.CharPositionInLine,
                                      pr.Line,pr.Text,null,false);
                                }
                                else
                                {
                                      $exprun2 = new ExpresionMenosMenos(
                                      pr.CharPositionInLine,
                                      pr.Line,pr.Text,null,false);
                                }
                          }
                    }
                    else
                    {
                          if(arit == 0)
                          {
                                $exprun2 = new ValorDerechoArreglo(
                                pr.CharPositionInLine,
                                pr.Line,pr.Text,expr1);
                          }
                          else
```

```
                                {
                                        if(arit == 1)
                                        {
                                                $exprun2 = new ExpresionMasMas(
                                                pr.CharPositionInLine,
                                                pr.Line,pr.Text,expr1,true);
                                        }
                                        else
                                        {
                                                $exprun2 = new ExpresionMenosMenos(
                                                pr.CharPositionInLine,
                                                pr.Line,pr.Text,expr1,true);
                                        }
                                }

                        }

        }
        :       pr=primario (CORCHETEABIERTO expr1=expresion
                CORCHETECERRADO{op = 1;})? (MASMAS{arit = 1;} |
                MENOSMENOS{arit = 2;})?;


primario returns [Primario primario]
        :       exppar=expresionparentesis{$primario=exppar;}
        |       ltr=literal{$primario= ltr;}
        |       id=ID{$primario = new ValorDerecho(id.CharPositionInLine,
                id.Line,id.Text);}
        |       pos=PC_NEW typ=tipo creacion=creacionarreglo{$primario= new
                CreacionArreglo(pos.CharPositionInLine, pos.Line, typ.Text,
                creacion);};

creacionarreglo returns [Expresion expr]
        :       CORCHETEABIERTO expr1=expresion CORCHETECERRADO
                {$expr= expr1;};

literal   returns [Literal ltr]
        :       dbl=FLOAT{$ltr=new LiteralFloat(dbl.CharPositionInLine,
                dbl.Line, dbl.Text);}
        |       ent=ENTERO{$ltr=new LiteralEntero(ent.CharPositionInLine,
                ent.Line, ent.Text);}
        |       cad=CADENA{$ltr=new LiteralCadena(cad.CharPositionInLine,
                cad.Line, cad.Text);}
        |       bol=PC_TRUE{$ltr=new LiteralBinario(bol.CharPositionInLine,
                bol.Line, bol.Text);}
        |       bol=PC_FALSE{$ltr=new LiteralBinario(bol.CharPositionInLine,
                bol.Line, bol.Text);}
        |       chr=CHAR{$ltr=new LiteralChar(chr.CharPositionInLine,
                chr.Line, chr.Text);}
        |       nll=PC_NULL{$ltr=new LiteralNull(nll.CharPositionInLine,
                nll.Line,nll.Text);};
```

Anexo 4:

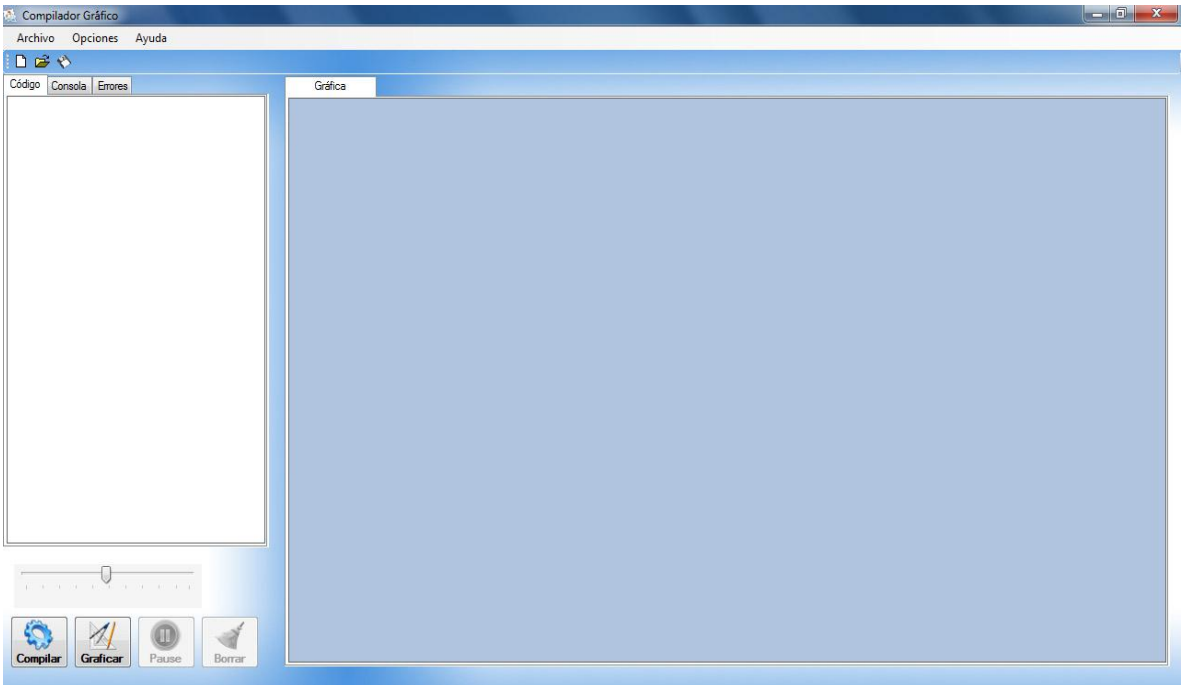

Figura 11 Interfaz de usuario: Pantalla principal.

Anexo 5:

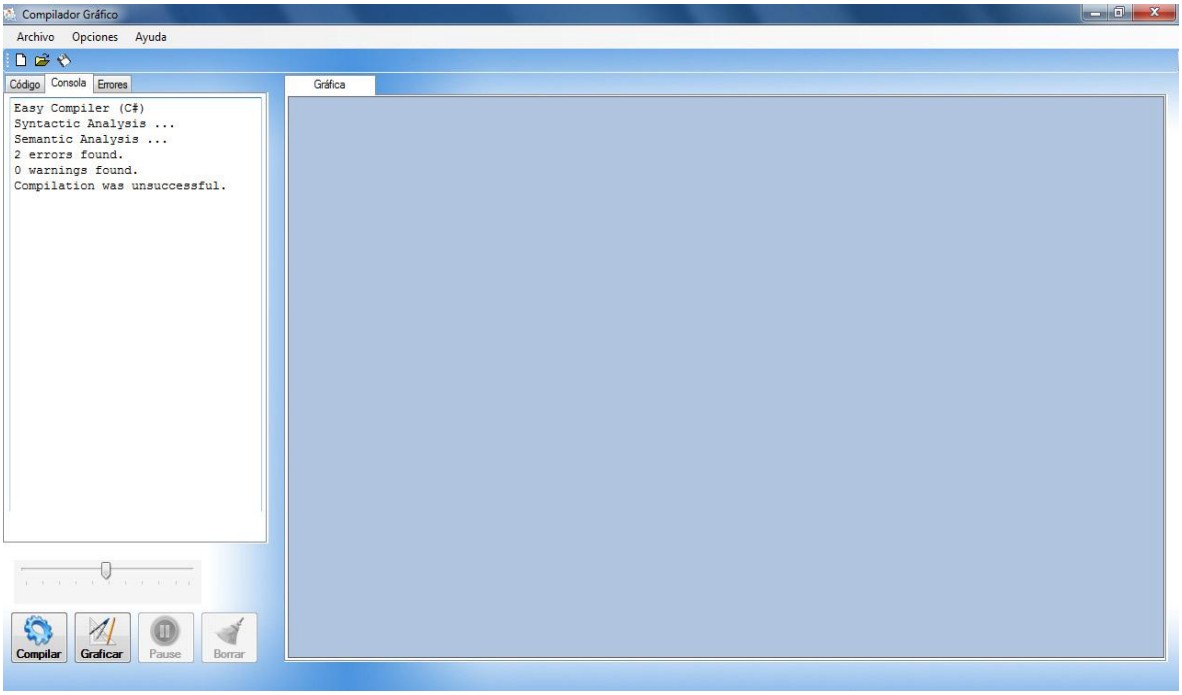

Figura 12 Interfaz de usuario: Resultado de la compilación.

Anexo 6:

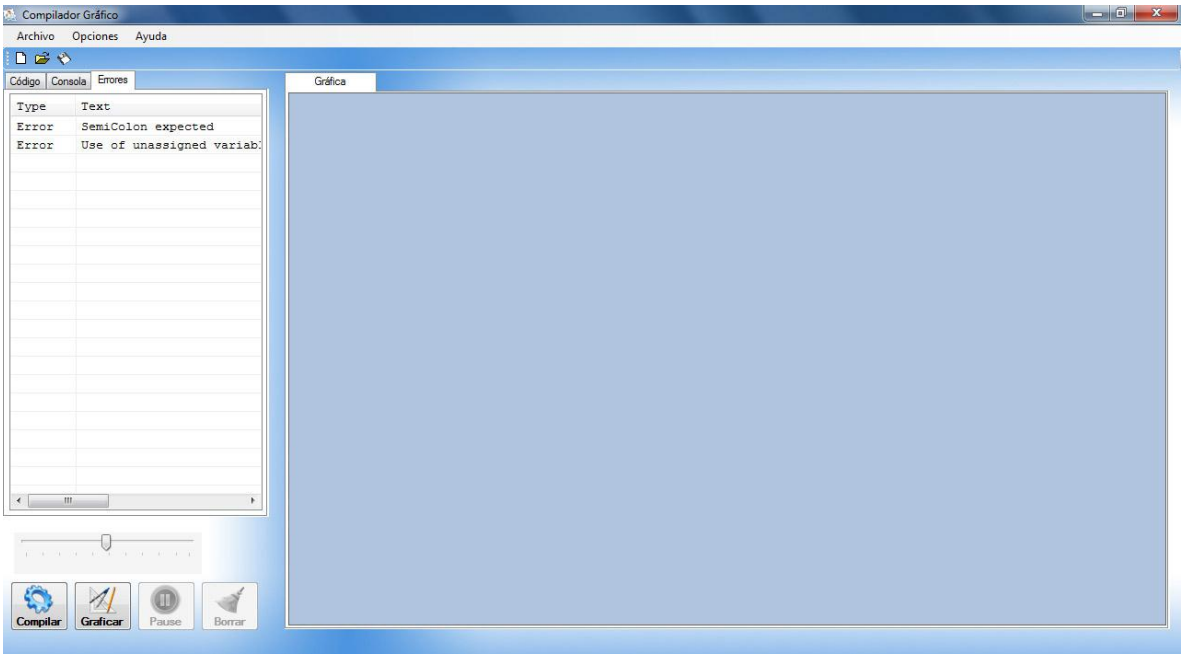

Figura 13 Interfaz de usuario: Errores.

Anexo 7:

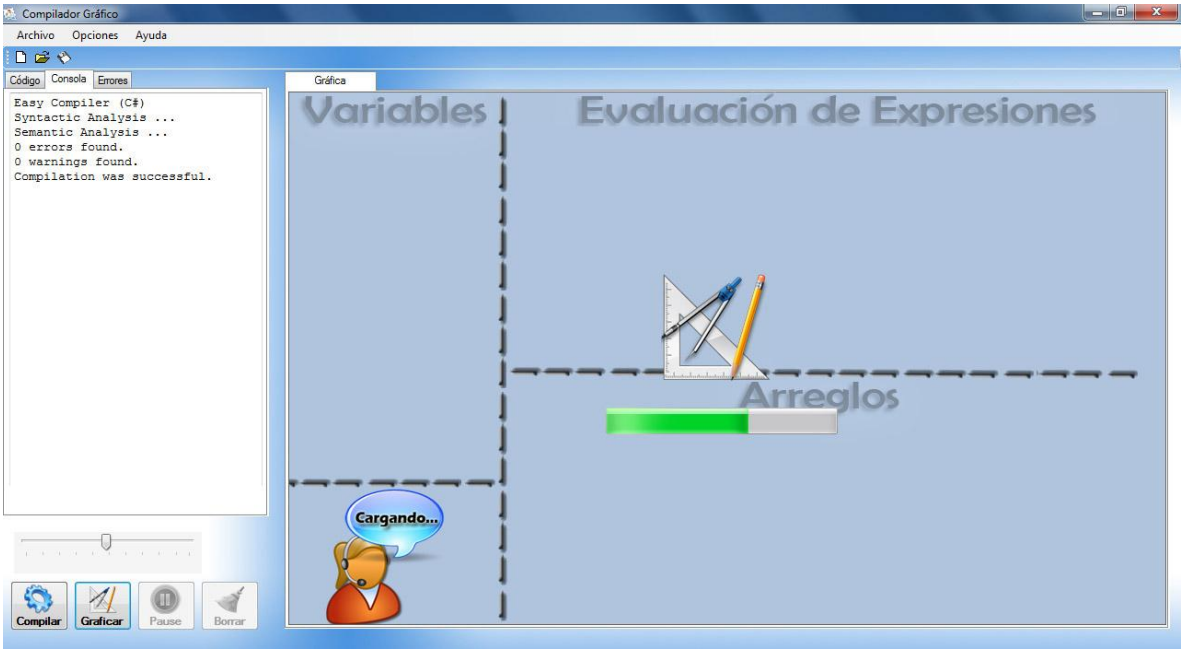

Figura 14. Interfaz de usuario: Preparando escenario.

Anexo 8:


Compilador Gráfico
Archivo
Opciones
Ayuda
Código
Consola
Errores
static void Main()
{
int a,b;
bool c;
a = 4+5;
b = 5*2;
c=false;
while(c==false)
{
while(a<b)
{
a=a+1;
}
c=true;
}
}
Compilar
Graficar
Pause
Borrar
Gráfica
Variables
int a 10
int b 10
bool c True
Evaluación de Expresiones
4 + 5 = 9
5 * 2 = 10
False == False = True
9 < 10 = True
9 + 1 = 10
10 < 10 = False
Arreglos
Graficando