Universidad de las Ciencias Informáticas

Facultad 4

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

Título: Herramienta online para el diseño de diagramas de flujos de datos y la generación de código de algoritmos en lenguajes de alto nivel.

Autor: Tomás López Fernández.

Tutor: Ing. José Antonio Soto Pérez.

Co-Tutor: Ing. Arlan Gálvez Alonso.

La Habana, 2011

"Año 53 de la Revolución"

Declaración de Autoría

Declaro ser el único autor del presente trabajo de diploma y reconozco a la Facultad 4 de la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

Para que así conste firmo la presente a los ___ días del mes de junio del año 2011.

Tomás López Fernández	Ing. José Antonio Soto Pérez
_____	_____
Firma del autor	Firma del tutor

Dedicatoria

A mis abuelos: Jorge, Ori, Cira y Alberto, los arquitectos principales de la maravillosa familia a la que pertenezco, por ser mis guías, mis padres, madres, mi todo. Los quiero viejos.

A mi mamá, por ser el pilar fundamental de mi existencia, la fuente de toda mi inspiración, por todo el amor que siempre he recibido de ella, por ser lo más grande y bello de mi vida.

A mi papá, por todo lo bueno que ha sembrado en mí, por la educación que me ha dado siempre, por hacer de mi un mejor hombre cada día.

A mi hermanita, por todo el amor que siempre me ha dado, por ser el complemento de mi felicidad.

A mi hermano, mis tíos, tías, primos, primas, a mi familia, por darme la fuerza y el valor necesario para enfrentar nuevos retos cada día, por su apoyo incondicional.

Agradecimientos

Agradezco a mis abuelos, mis padres, mi familia por toda una vida llena de amor, por el apoyo que siempre he recibido de ustedes, por todo lo que han hecho por mí, por las enseñanzas que han hecho de mí un hombre de bien, por hacerme cada día un mejor hermano, un mejor hijo, un mejor hombre.

A mis amigos incondicionales: Pepe, Robert, Maxi, Wilson, Leonardo, Jandy. A mis amigas Liannis, Grehiliys, Anett, Yisel, Arianna, Jeydi, Mily, Leslie, Lizandra, Aimé, Tatiana. A todos los que han crecido conmigo, los que han compartido un pedazo de vida, a los que ya no están, a los que están por venir. Gracias.

A mi novia Maire, por el apoyo incondicional en estos difíciles meses de tesis. Un beso. Gracias.

A mis compañeros de la universidad, los que llegamos, los que seguimos, los que quedamos. A los muchachos de mi apartamento, a los que compartieron conmigo estos cinco inolvidables años.

A mis maestros y profesores de toda una vida.

A todos los mencionados, a los que por una u otra razón no están aquí. Gracias.

Resumen

Los diagramas de flujos de datos (DFD) son empleados en todo el mundo como métodos de apoyo a los procesos de Enseñanza-Aprendizaje de diversos contenidos relacionados con el diseño de algoritmos computacionales. La Universidad de las Ciencias Informáticas cuenta con una infraestructura tecnológica que posibilita la inserción de las herramientas web como parte de los procesos formativos de su alumnado. El objetivo de la presente investigación es el desarrollo de una herramienta web que permita el diseño de los DFD y la generación de los códigos de los algoritmos correspondientes, posibilitando el uso de los DFD como apoyo a los procesos de aprendizaje de los temas relacionados con el diseño de algoritmos impartidos en la asignatura Introducción a la Programación perteneciente a la disciplina Programación. Para el desarrollo de la misma se empleó la tecnología Applet de Java, el IDE NetBeans en su versión 6.9.1 como editor de código, el lenguaje de programación Java para edición del código fuente y el servidor web Apache HTTP Server en su versión 2.2.6. Utilizando una arquitectura cliente-servidor se hizo uso del patrón Modelo-Vista-Controlador, y como metodología de desarrollo de software se adoptó la metodología ágil XP.

Palabras clave: Diagramas de flujos de datos, herramienta web, algoritmos computacionales, generación de código.

Tabla de contenidos

Índice de tablas

Índice de ilustraciones

Introducción

La revolución informática iniciada hace unos cincuenta años atrás e intensificada incontenible mente en las últimas dos décadas mediante el incesante desarrollo de las nuevas tecnologías de la información y los medios de comunicación, influye determinantemente en los distintos ambientes en los que se desenvuelven los diversos ámbitos de la sociedad. A la par con la creciente globalización de la economía y el conocimiento, la informática conduce a profundos cambios estructurales en todas las naciones, cambios a los que Cuba no está ajena, iniciando en el año 1999 un proceso de informatización de la sociedad enfocado principalmente a la esfera educacional, con la introducción de modernas tecnologías en todas las escuelas y niveles de enseñanza del país.

Como parte de este proceso de informatización, en el año 2002, surge la Universidad de las Ciencias Informáticas (UCI), institución que tiene como uno de sus objetivos la formación de profesionales de las ciencias informáticas altamente preparados y profundamente comprometidos con los valores y principios de la Revolución cubana.

La UCI es una universidad donde los avances de las Tecnologías de la Información y las Comunicaciones (TIC) forman parte indispensable en del proceso de formación de su alumnado. Gracias a la gran infraestructura tecnológica que posee, cuenta con varios sitios web que apoyan a la formación profesional de sus estudiantes. Entre los sitios más frecuentados se encuentran el Entorno Virtual de Aprendizaje (EVA), donde los estudiantes pueden acceder a un gran cúmulo de información referente a las asignaturas y disciplinas que reciben en la docencia, esta información está dividida por temáticas y cuenta además con numerosos materiales didácticos desarrollados sobre plataformas web como apoyo al aprendizaje de los contenidos docentes, facilitándole a los estudiantes medios alternativos para consolidar los conocimientos adquiridos.

El uso de aplicaciones basadas en tecnologías web en la UCI, posibilita el acceso de todos los estudiantes a los distintos recursos disponibles en los sitios de apoyo a la docencia, los que pueden ser consultados a cualquier hora del día o la noche todos los días de la semana, además de contar con foros, mensajería instantánea y correo electrónico, lo que favorece la comunicación entre los propios estudiantes así como con los profesores internos en la universidad. Esto posibilita que las aplicaciones web constituyan

herramientas fundamentales como materiales de apoyo al proceso de Enseñanza-Aprendizaje de los estudiantes de la UCI.

En la UCI, con un modelo de formación centrado en el aprendizaje, los estudiantes reciben una preparación intensiva en distintas disciplinas necesarias para su formación, teniendo como parte del ciclo básico la disciplina de Programación, dentro de la que se encuentra la asignatura Introducción a la Programación. Esta asignatura se divide por contenidos que se van impartiendo siguiendo un orden, establecido por la dificultad del tema tratado, pasando de temas más sencillos a otros más complejos en el transcurso del semestre.

En el mundo existen numerosos métodos que permiten apoyar el aprendizaje de los estudiantes respecto a determinados temas de programación como la construcción de algoritmos computacionales, uno de estos métodos es el llamado Diagrama de flujos de datos (DFD), que constituye una estructura gráfica de un número limitado de elementos con significado que permiten indicar el camino que toman los datos durante el desarrollo de un algoritmo determinado.

Durante el semestre en que se imparte la asignatura Introducción a la Programación, se orienta el uso de los DFD como apoyo para el aprendizaje de los temas referentes al diseño de algoritmos, los cuales deben ser confeccionados de manera manuscrita por los propios estudiantes, lo impide que se pueda comprobar si los algoritmos diseñados funcionan correctamente. No permite visualizar el código ya que este debe ser escrito por los alumnos pudiendo no tener relación con el diagrama diseñado, perdiéndose así la funcionalidad del DFD como medio para apoyar el entendimiento y aprendizaje de las distintas estructuras que posibilitan el diseño de algoritmos.

Algunas herramientas como la Cristal Flow para C++ y la Crystal Revs para C++ son empleadas a nivel internacional para diseñar DFD y generar los códigos de los diagramas correspondientes, el uso de estas herramientas posibilitaría que los estudiantes de la UCI hicieran un uso más práctico de los DFD, interactuando de manera gráfica con las estructuras que conforman un algoritmo computacional. Además de permitir la generación del código de manera automática, lo que facilitaría la comprobación de la funcionalidad del algoritmo diseñado. Estas herramientas poseen licencias de software propietario lo que restringe su uso, además son aplicaciones de escritorio, por lo que sólo pueden ser usadas en las computadoras donde están instaladas, limitando así el acceso a las mismas ya que deben ser

descargadas e instaladas en las máquinas clientes, donde no todos los estudiantes tienen permisos suficientes para realizar las mencionadas acciones. Producto a las más de ocho mil computadoras conectadas en red que posee la UCI, es parte de la cultura de los estudiantes y profesores el uso de las plataformas web como parte inseparable de los procesos de Enseñanza-Aprendizaje, lo que posibilita la accesibilidad y disponibilidad de la información presente en los sitios de apoyo a la docencia existentes en la universidad.

El uso de las tecnologías web para realizar el diseño de DFD y generar el código de los algoritmos correspondientes, posibilitaría que los estudiantes de la UCI pudieran acceder a una herramienta que les permitiera utilizar las ventajas de los DFD como apoyo al aprendizaje de los temas relacionados con el diseño de algoritmos computacionales, pudiendo hacer uso de la misma a cualquier hora cualquier día de la semana y desde cualquier punto de la universidad.

Al no existir en la UCI un mecanismo que posibilite el uso de los DFD y la generación de códigos de los algoritmos correspondientes, que sirva apoyo a los procesos de Enseñanza-Aprendizaje de los temas correspondientes al diseño de algoritmos impartidos en la asignatura Introducción a la Programación, se desaprovechan las potencialidades de los DFD como métodos didácticos de apoyo a la docencia, teniendo en cuenta la infraestructura tecnológica que posee la UCI y el uso de las plataformas web que diariamente hacen sus estudiantes y profesores.

El **problema a resolver** derivado de la situación problémica es: ¿Cómo apoyar el aprendizaje de los contenidos referentes a la construcción de algoritmos impartidos en la asignatura Introducción a la Programación haciendo uso de los DFD y la generación de código de los algoritmos diseñados?

Como **objeto de estudio** se define el proceso de desarrollo de aplicaciones web que permitan diseñar DFD y la generación del código correspondiente a los algoritmos diseñados. El **campo de acción** es el proceso de desarrollo de una aplicación web que permita el diseño los DFD y la generación del código correspondiente a los algoritmos diseñados, estudiados en el tema de diseño de algoritmos computacionales de la asignatura Introducción a la Programación impartida en la UCI.

El **objetivo general** de este trabajo es:

Desarrollar una herramienta web que posibilite el diseño de los DFD y la generación de código en los lenguajes Java, C++ y CSharp (C#), estudiados en la UCI en la disciplina Programación, correspondiente a los algoritmos diseñados.

Para darle solución al objetivo general planteado se definen los siguientes **objetivos específicos:**

- Realizar un estudio del estado del arte sobre las herramientas que permiten el diseño de los DFD y la generación de código de los algoritmos correspondientes para definir la posición del investigador.

- Implementar la herramienta web para el diseño de diagramas de flujo y la generación de códigos de algoritmos simples en los lenguajes Java, C# y C++.

- Realizar la validación de la propuesta de solución.

Los **posibles resultados** que se esperan obtener con el desarrollo de esta aplicación haciendo uso de tecnologías orientadas a plataformas web son: una herramienta web que permita diseñar algoritmos simples haciendo uso de los elementos que componen los DFD, así como la generación de códigos en el lenguaje Java, C# y C++ de los algoritmos diseñados. Esta herramienta le permitirá a los estudiantes, principalmente los de primer año, acceder desde cualquier punto de la universidad a una aplicación que les facilite el diseño de algoritmos simples utilizando los DFD, brindándoles la posibilidad de visualizar las diferentes técnicas de diseños de algoritmos, haciendo más interactivo el aprendizaje de las mismas a través una interfaz visual fácil de manipular.

Las **tareas** orientadas a darle solución a los objetivos específicos son las siguientes:

1. Análisis profundo sobre los procesos de diseño de los diagramas de flujos de datos.
2. Estudio y selección de las herramientas, tecnologías y metodologías a utilizar en el desarrollo de la herramienta.
3. Análisis y diseño del módulo para el diseño de diagramas de flujos de datos haciendo uso de la metodología seleccionada.
4. Implementación del módulo de diseño de diagramas.

5. Análisis y diseño del módulo para la generación del código de los algoritmos diseñados en los lenguajes Java, C# y C++.
6. Implementación del módulo para la generación de código.
7. Evaluación de la herramienta mediante la realización de pruebas que permitan comprobar que las funcionalidades de la herramienta fueron implementadas correctamente.

Métodos de investigación a emplear

Histórico-Lógico: haciendo uso de este método se investigó sobre los antecedentes de los diagramas de flujos de datos así como en el estado del arte de las herramientas que permiten el diseño de los mismos. A través de esta investigación se obtiene información que brinda la posibilidad de realizar una selección de las herramientas, técnicas y metodologías más factibles a utilizar en la construcción de la aplicación basada en la evolución de las tecnologías de diseño de diagramas y la generación de códigos de algoritmos en lenguajes de alto nivel.

Analítico-Sintético: al aplicar este método se posibilitó el estudio de los diferentes elementos que conforman los diagramas de flujos de datos así como los procesos que utilizan las aplicaciones existentes para lograr la diagramación y la generación de código partiendo de los diagramas diseñados. Esto permite la integración de los elementos existentes en una solución aplicable al modelo de enseñanza-aprendizaje de la asignatura Programación en la UCI.

Modelación: a través de la modelación se realizaron las representaciones gráficas de los diferentes elementos que se utilizarán en la construcción de la propuesta de solución, muy útil en el diseño de los diferentes modelos de Programación Orientada a Objetos (POO) que se desea emplear en la solución de la presente investigación.

Métodos empíricos

Entrevista: a través de la entrevista el equipo de desarrollo pudo obtener información sobre el tema investigado a través de la interacción personal con profesionales o expertos en el tema, lo que posibilita abarcar los conocimientos de los implicados en la solución del problema.

Descripción capitular

El presente trabajo de diploma se encuentra estructurado de la siguiente manera:

Capítulo I “Fundamentación Teórica”: en este capítulo, se realiza un estudio del estado del arte sobre el uso actual de los diagramas de flujos de datos y las herramientas más usadas en el diseño de los mismos, pudiendo tomarse como soluciones a tenerse en cuenta. Se deja plasmado el por qué las herramientas estudiadas no son soluciones factibles para el problema. Además se realiza un estudio de las metodologías, tecnologías y herramientas de desarrollo de software más utilizadas en la actualidad, seleccionándose así las más viables a emplear en la construcción de la solución.

Capítulo II “Características del Sistema. Exploración y Planificación“: en el presente capítulo se tiene como objetivo hacer una valoración de las principales características del sistema a desarrollar, así como se desarrollan las primeras iteraciones de la metodología de desarrollo seleccionada, recogiéndose los artefactos iniciales de la misma.

Capítulo III “Construcción y validación de la solución propuesta”: en este capítulo se exponen los pasos a seguir para la construcción exitosa de la propuesta de solución haciendo uso de la metodología seleccionada, así como la descripción de las pruebas realizadas para validar la calidad del producto obtenido.

1. Capítulo 1. Fundamentación Teórica

1.1 Definición de algoritmos simples

Una definición que se establece en la presente investigación es la de algoritmo simple, que se refiere a los algoritmos que se podrán diseñar haciendo uso de los DFD a través de la herramienta propuesta para la solución. Se considera un algoritmo simple aquel que posea las siguientes características:

- Un único estado inicial y un único estado final.
- Operaciones aritméticas entre dos variables y un solo operador algebraico.
- Estructuras condicionales (if) con no más de dos bloques de instrucciones.
- Estructuras repetitivas (for) con no más de un bloque de instrucciones.
- Estructuras condicionales (if) con una estructura repetitiva (for) dentro del bloque de instrucciones, cumpliendo con las características de la misma definida anteriormente.
- Estructura repetitiva (for) con una estructura condicional (if) dentro del bloque de instrucciones, cumpliendo con las características de la misma definida anteriormente.

Haciendo uso de estos elementos, se pueden construir los algoritmos básicos estudiados en los temas de diseño de algoritmos dentro de la asignatura Introducción a la Programación, perteneciente a la disciplina Programación, impartida en el primer semestre de la carrera.

1.2 Definición de Diagrama de flujos de datos

En la actualidad existen diversas definiciones para describir lo que sería un DFD. Autores como la Dra. María Mercedez Vitturini (Argentina, 2010) los definen como: *"un modelo que representa el flujo de la información y los cambios que se le aplican desde las entradas hasta las salidas"* (1). Otra definición es la del Ing. Teodoro Córdova Neri (Perú, 2005) que plantea que: *"el diagrama de flujos de datos permite ilustrar de forma gráfica como diseñar procedimientos o sentencias con coherencia lógica, que representan la solución a un problema dado"* (2). Estos criterios expresan con claridad qué es un DFD. Para el desarrollo de la presente investigación, el autor considera que los DFD son una secuencia limitada de elementos con significado que permiten visualizar de forma gráfica el camino que toman los datos durante la ejecución de un algoritmo.

1.3 Características de los DFD

Los DFD se dibujan generalmente antes de comenzar a programar el código frente a la computadora. La realización de un correcto diseño de los diagramas facilita el entendimiento de algoritmos muy complejos o muy largos, también es una forma más simple para explicar los diferentes pasos definidos para alcanzar la solución a otras personas. Una vez que se diseña el diagrama suele ser muy fácil escribir el código en cualquier lenguaje de programación de alto nivel, ejemplo Java, C++ y C#. Los DFD se pueden considerar como vías muy efectivas para facilitar la comunicación entre los diferentes miembros de un equipo de desarrollo, ya que definen un marco común para todos, tanto los programadores como los entes del negocio.

Para realizar el diseño de los DFD se deben tener presentes un grupo de reglas que se mencionan a continuación (2):

1. Los diagramas de flujo de datos deben escribirse de arriba hacia abajo y/o de izquierda a derecha.
2. Los símbolos se unen con líneas, las cuales tienen en la punta una flecha que indica la dirección hacia donde fluye la información, se deben de utilizar solamente líneas de flujo horizontal o vertical (nunca diagonal).
3. Se debe evitar el cruce de líneas, para lo cual se quisiera separar el flujo del diagrama a un sitio distinto, se pudiera realizar utilizando los conectores. Se debe tener en cuenta que solo se van a utilizar conectores cuando sea estrictamente necesario.
4. No deben quedar líneas de flujo sin conectar.
5. Todo texto escrito dentro de un símbolo debe ser legible, preciso, evitando el uso de muchas palabras.
6. Todos los símbolos pueden tener más de una línea de entrada, a excepción del símbolo final.

Para la realización del diseño de los DFD se cuenta con un grupo de componentes visuales, los que abarcan o permiten definir las diferentes operaciones que se pueden representar a través de los diagramas. Entre las operaciones que se pueden representar se encuentra la entrada y salida de datos, asignación de valores, bifurcaciones o condiciones de decisión, repeticiones, entre otras. Los componentes que a continuación se presentan son los más utilizados en el diseño de los DFD.

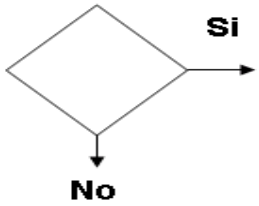
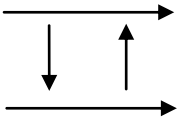
	Inicio / Fin.
	Operación algebraica o de asignación.
	Salida. Representan cualquier fuente salida de datos.
	Expresa condiciones y asociaciones alternativas de una decisión lógica.
	Operación cíclica repetitiva.
	Flujos.

Tabla 1. Componentes de los DFD.

Existen otros componentes que se pueden emplear en el diseño de los DFD, en el desarrollo de la aplicación propuesta para la solución de la presente investigación solo se utilizarán los componentes aquí mencionados, ya que los algoritmos que se podrán modelar en la primera versión de herramienta son de baja y mediana complejidad, acorde con los temas estudiados en la primera etapa del primer año de la carrera.

1.4 Herramientas existentes para el diseño de DFD

En la actualidad se pueden encontrar numerosas herramientas que permiten el diseño de diagramas de flujos de datos así como permiten la generación de código de algoritmos en diferentes lenguajes. A continuación se describen algunas de las herramientas encontradas durante el desarrollo del estudio del estado del arte sobre los diferentes programas para el diseño de los DFD.

1.4.1 Crystal Flow para C++ v3.83

Entre las diferentes opciones y facilidades que brinda esta herramienta se encuentran (3):

- Generación del código de diagrama de flujo para tener una vista clara del código.
- Implementación de la estructura llamada/llamante/fichero enriquecido, utilización de los árboles de clases, y posibilidad de realizar el formateo automático de código y/o comentarios para mejorar la legibilidad.
- Realiza la exportación a distintos formatos como son BMP/JPEG/Visio, y realiza las búsquedas mediante la incorporación de un explorador de código.

Permite realizar numerosas ediciones, entre las más significativas se encuentran:

- Diagrama de datos, Diagrama de flujos de objetos, Diagrama de llamadas, expansión en línea de diagramas de flujo.
- Documentación: documentos HTML extensos con diagramas de flujo/árboles.

Esta herramienta posee características similares a las que se desea incorporar a la propuesta de solución. Principalmente se debe tomar como ejemplo para el módulo de diseño de los DFD, la forma en la que esta herramienta permite mover los componentes a través de eventos del periférico mouse o ratón. También posee varias desventajas que la convierten en una opción no viable para su aplicación como solución de la presente investigación, ya que la misma se presenta como herramienta orientada a escritorio, lo que significa que no puede utilizarse a través de la web. Presenta licencia de software propietario con un costo de $279.00 USD, que no se aplica los principios de migración a software libre que hoy se lleva a cabo en la UCI y en Cuba.

1.4.2 Crystal Revs para C++

Esta herramienta posee variadas opciones que permiten la creación de diversos diagramas así como la generación de códigos de algoritmos en numerosos lenguajes de programación. Algunas de sus principales características se exponen a continuación (4):

- Generación de código a partir de diagramas de flujo para una vista más clara del código.
- Verificación y correcciones de funciones. Detección de errores.
- Visualización por niveles, entre ellos DFD óptimos, bucles y condiciones por área.
- Sincronización de código y diagrama de flujo al navegar por el explorador de soluciones.
- Diagramas de flujo con comentarios para ampliar la audiencia.
- Exportación de diagramas de flujo a formatos BMP/JPEG/Visio.
- Diagramas de flujo con sólo comentarios, sólo código o con ambos.
- Impresión de largos diagramas de flujo en una sola página.

Esta herramienta brinda muchas de las funcionalidades que se desean obtener en el resultado de la presente investigación. Entre las características que esta herramienta posee que pudieran servir como ejemplo para la solución que se desea desarrollar, se encuentra la generación de código en el lenguaje C++, que puede ser la base para la generación de los códigos en los lenguajes C# y Java. Aunque es una posible solución, por las ventajas y excelentes prestaciones que posee la herramienta, no se puede considerar como una factible debido a que está desarrollada para plataformas de software propietario como una aplicación orientada a escritorio, lo que imposibilita su uso en la web desaprovechando las bondades de la UCI respecto a estas tecnologías y posee una licencia privativa de costo $ 399.00 USD.

1.4.3 Origramy Flash Graph Component

El software Origramy Flash Graph Component (Componente gráfico Flash Origramy) es un componente gráfico diseñado para realizar la construcción, edición y visualización de gráficas, presentaciones, esquema del flujo de trabajo, diagramas, etc. El mismo presenta, entre otras, las siguientes características (5):

- Dirigido por XML.
- Interfaz amigable e intuitiva.

- Soporte para el lenguaje JavaScript.
- Amplia gama de maneras de organización de datos.
- Múltiples formatos de exportación.

La herramienta Componente gráfico Flash Origramy posee varias de las características que se desean implementar, como es el estar desarrollado sobre la web. Esta herramienta, realiza el diseño de los DFD a través de eventos del periférico mouse o ratón, los cuales son ejemplos que se desean implementar en la herramienta resultante de la presente investigación. También posee algunas limitantes que lo convierten en una solución poco factible para la solución de la presente investigación, ya que el mismo no permite la revisión o ejecución del código de un algoritmo diseñado mediante un diagrama de flujo de datos. Por lo que se descarta como una posible factible solución.

1.5 Metodologías de desarrollo de software

Las metodologías de desarrollo de software surgen por la necesidad de controlar y documentar proyectos cada vez más complejos. Estas contienen numerosos requisitos y establecen las pautas a seguir para alcanzar un resultado favorable durante todas las fases del desarrollo de un software determinado. A continuación se describen las ventajas del uso de las distintas metodologías de desarrollo de software.

Ventajas del uso de las metodologías de desarrollo de software (6)

- Todos los integrantes del proyecto trabajan bajo un marco común.
- Se estandarizan conceptos, actividades y nomenclaturas.
- Las actividades de desarrollo se ven apoyadas por procedimientos y guías.
- Resultados de desarrollo potencialmente predecibles.
- Uso de herramientas de Ingeniería de Software.
- Se planifican actividades en base a un conjunto de tareas definidas y a la experiencia de otros proyectos.
- Recopilación de mejores prácticas para proyectos futuros.
- Se asegura que los productos cumplen con los objetivos de calidad propuestos.
- Detección de errores de manera temprana.
- Se garantiza la trazabilidad del producto a lo largo del desarrollo.

- El cliente percibe el orden en que se ejecutan los procesos.
- Facilita al cliente el seguimiento de la evolución del proyecto.
- Se establecen mecanismos para asegurar que los productos desarrollados cumplen con las expectativas del cliente.

En la actualidad existen diferentes propuestas metodológicas para llevar a cabo el desarrollo de proyectos de desarrollo de software. Estas propuestas se identifican mediante la existencia de dos clasificaciones, ambas contribuyen a un buen y organizado desarrollo de software aunque presenten marcadas diferencias. A continuación se presentan estas clasificaciones:

- Las metodologías tradicionales, haciendo énfasis en la planeación.
- Las metodologías ágiles, haciendo énfasis en la adaptabilidad del proceso.

1.6 Metodología tradicional

La metodología tradicional impone una disciplina de trabajo sobre los procesos de desarrollo de software, haciendo énfasis en la planificación total de todo el trabajo a realizar con el fin de alcanzar un exitoso desarrollo de los sistemas deseados. Una vez que todo está detallado correctamente, comienza el ciclo de desarrollo del producto de software, que se centra especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada de todo lo que sucede. La metodología tradicional no se adapta fácilmente a los cambios, por lo que no son factibles en entornos donde los requisitos no pueden predecirse o bien pueden cambiar continuamente (7).

1.6.1 Rational Unified Process (RUP)

La metodología de desarrollo de software tradicional RUP, es un proceso formal, la misma provee un acercamiento disciplinado para la asignación de tareas y responsabilidades dentro de una organización de desarrollo. Como objetivo persigue asegurar la producción de software de alta calidad que satisfaga las necesidades de los clientes respetando cronograma y presupuesto. Fue desarrollada por Rational Software, y está integrada por toda la suite Rational de herramientas. Esta metodología puede ser adaptada y extendida para satisfacer las necesidades de la organización que la adopte, es guiada por

caso de uso, centrado en la arquitectura, iterativo e incremental y utiliza el Lenguaje Unificado de Modelado (UML) como lenguaje de modelado (8).

Características de RUP

- Establece un desarrollo iterativo.
- Permite la administración de requisitos.
- Hace uso de una arquitectura basada en componentes.
- Permite llevar un control de cambios.
- Permite realizar un modelado visual del software.
- Verifica la calidad del software.

El uso de metodologías robustas, como RUP, no es factible para el desarrollo de proyectos pequeños que puedan estar sujetos a constantes cambios en los requerimientos, ya que un ligero cambio en uno de ellos, podría ocasionar que se tuviera que generar toda la documentación nuevamente, lo que prolongaría el tiempo de desarrollo del proyecto. Por tal motivo, se decide el uso de una metodología que esté más enfocada a darle cumplimiento a las funcionalidades que a realizar una documentación exhaustiva de todo el proceso de desarrollo, descartando el uso de RUP como metodología de desarrollo.

1.7 Metodologías ágiles

Tras una reunión realizada en Utah, Estados Unidos, en el 2001 nace el término ágil referido al desarrollo de software. En esta reunión participaron diecisiete expertos de la industria del software, incluyendo algunos de los creadores e impulsores de las metodologías de desarrollo de software. Su principal objetivo fue el de enmarcar los valores y principios que permitieran a los equipos desarrollar software rápidamente y responder ante los cambios que surgieran durante el ciclo de vida del proyecto. Estos expertos pretendían ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera por cada una de las actividades desarrolladas (9).

A continuación se describen algunas de las metodologías ágiles estudiadas durante la presente investigación que son empleadas en el mundo para el desarrollo de software.

1.7.1 SCRUM

SCRUM, más que una metodología de desarrollo software, es una forma de auto-gestión de los equipos de programadores. Esta metodología ayuda a los trabajadores a trabajar todos juntos, en la misma dirección, con un objetivo claro (10).

Principios de SCRUM

- Los equipos son auto-gestionados.
- Una vez dimensionadas las tareas no es posible agregarle trabajo extra.
- Se realizan reuniones diarias en las que los miembros del equipo se plantean tres cuestiones:

  ¿Qué has hecho desde la última revisión?

  ¿Qué obstáculos te impiden cumplir la meta?

  ¿Qué vas a hacer antes de la próxima reunión?
- Las iteraciones de desarrollo tienen una frecuencia inferior a un mes, al final de las cuales se presenta el resultado a los externos del equipo de desarrollo, y se realiza una planificación de la siguiente iteración, guiada por el cliente.
- Entrega de un producto funcional al finalizar cada Sprint.
- Posibilidad de ajustar la funcionalidad en base a la necesidad de negocio del cliente.
- Visualización del proyecto día a día.
- Alcance acotado y viable.

Las características de SCRUM la convierten en una posible metodología de desarrollo a emplear en el desarrollo de la presente investigación, pero el autor considera que algunas de las prácticas que la misma define no se ajustan o no son necesarias teniendo en cuenta las características del equipo de desarrollo integrado por un solo miembro.

1.7.2 Crystal

La metodología Crystal se basa en un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del

número de artefactos producidos. Fue desarrollada por Alistair Cockburn. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear (3 a 8 miembros) y Crystal Orange (25 a 50 miembros) (11).

Crystal da vital importancia a las personas que componen el equipo de un proyecto, y por tanto sus puntos de estudio son:

- Aspecto humano del equipo.
- Tamaño de un equipo (número de componentes).
- Comunicación entre los componentes.
- Distintas políticas a seguir.
- Espacio físico de trabajo.

Los roles principales definidos por esta metodología son:

- Patrocinador ejecutivo (Executive Sponsor).
- Jefe de proyecto (Project Manager).
- Experto en el dominio (Domain Expert).
- Experto de uso (Usage Expert).
- Programador diseñador (Designer-Programmer).
- Diseñador de interfaz de usuario (Designer).
- Realizador de pruebas (Tester).
- Programador técnico (Technical).

Algunas de las técnicas propuestas por esta metodología son:

- Entrevistas de proyectos.
- Talleres de reflexión.
- Planeamiento Blitz.

- Estimación Delphi.
- Encuentros de a pie.
- Miniatura de procesos.
- Gráficos de quemado.
- Programación lado a lado.

Crystal es una metodología centrada en fortalecer las relaciones humanas, reducir el número de artefactos generados y en la cual las políticas de trabajo deben estar bien definidas de acuerdo con el tamaño del equipo de desarrollo. El autor del presente trabajo considera que las técnicas definidas para el desarrollo de software utilizando la metodología Crystal no son las ideales, teniendo en consideración que el equipo de desarrollo está integrado por un solo miembro, por lo que el juego de planificación y técnicas de documentación harían más complejo el trabajo para alcanzar el objetivo final.

1.7.3 eXtreme Programming (XP)

Creada por Kent Beck, se caracteriza por retomar prácticas tradicionales y llevarlas al extremo, de ahí su nombre. Esta metodología detalla varios valores de los métodos ágiles en busca de mejoras en el desarrollo de software y con bases fundamentales en la ingeniería de software. Se basa en la realimentación continua entre cliente y el equipo de desarrollo, la comunicación fluida entre todos los participantes, la simplicidad en las soluciones implementadas y el coraje para enfrentar los cambios (12).

XP está basada en cinco valores fundamentales para su desarrollo, los mismos son:

- Comunicación: la comunicación permanente es fundamental en XP. Dado que la documentación es escasa, el diálogo frontal, cara a cara, entre desarrolladores, gerentes y el cliente es el medio básico de comunicación. Una buena comunicación tiene que estar presente durante todo el proyecto.
- Simplicidad: la sencillez es esencial para que todos puedan entender el código, y se trata de mejorar mediante recodificaciones continuas.
- Feedback: básicamente el continuo contacto con el usuario, al irle entregando las sucesivas versiones, en funcionamiento del producto, permite que este nos dé su valoración y nos comunique, cada vez mejor, lo que realmente quiere en el producto.

- Coraje: básicamente es trabajar muy duro durante las horas dedicadas a ello.
- Respeto: si los miembros de un equipo no se preocupan por sí mismos y por su trabajo, la metodología no puede funcionar. Es necesario ser respetuoso con sus colegas, sus contribuciones, su organización y con las personas cuya vida se toca por el sistema que está escribiendo.

Basándose en estos cinco valores, los proyectos que adopten la metodología XP deben tener en cuenta varias prácticas, las cuales se integran unas con otras para ofrecer una base consistente para un favorable desempeño, alta productividad e incalculables beneficios, entre estas prácticas se destacan:

- Ciclo semanal.
- Programación en parejas.
- Incremento del diseño.
- Primer test programación.
- Participación real de clientes.

Teniendo en cuenta las prácticas que propone la metodología XP, que se enfoca en la programación por encima de toda documentación, con propuestas de documentación sencillas y de fácil manejo que se adaptan a las características del equipo de desarrollo, que está compuesto por un solo integrante, con un proyecto a desarrollar de dimensiones pequeñas y basado en el uso de nuevas tecnologías que está expuesto a múltiples riesgos técnicos, una planificación realizada a corto plazo por estar sujeta a las necesidades de la producción donde se exigen entregas en cortos intervalos de tiempo, se decide por parte del autor adoptar la metodología ágil XP como metodología de desarrollo de software a emplear para la construcción de la propuesta de solución.

1.8 Herramientas CASE y lenguaje de modelado a emplear

Computer Aided Software Engineering (CASE), que traducido al español significaría ingeniería de software asistida por computadoras, es el conjunto de herramientas que permiten modelar los procesos de negocios de las empresas. Tienen como objetivo principal enfrentar y solucionar problemas de calidad sobre los proyectos desarrollados así como problemas con la documentación de los mismos, además de posibilitar una generación automática de programas a partir de una especificación a nivel de diseño (13).

1.8.1 Rational Rose

Esta suite de herramientas cuenta con una plataforma independiente que ayuda a la comunicación entre los miembros del equipo, a entender el entorno de los sistemas y a monitorear el tiempo de desarrollo. Una de sus principales ventajas es que utiliza una notación estándar en la arquitectura de software (UML), la cual permite a todos los implicados en el desarrollo de un proyecto, el entendimiento del sistema completo utilizando un lenguaje común. Otra de las ventajas de esta herramienta es que los diseñadores pueden modelar sus componentes e interfaces de forma individual y luego integrarlas con otros componentes del proyecto (13).

Características

- Mantiene la consistencia de los modelos del sistema software.
- Chequeo de la sintaxis UML.
- Generación automática de la documentación.
- Generación de código a partir de los modelos: se puede generar código en distintos lenguajes de programación a partir de un diseño en UML.
- Ingeniería inversa: crear modelo a partir código.

Rational Rose constituye una suite de herramientas CASE de referencia mundial, formando parte de las herramientas más empleadas en ámbitos de desarrollo de software a nivel internacional. Por limitaciones de uso restringido sólo a sistemas operativos propietarios y poseer licencias privativas, se decide por parte del equipo de desarrollo no emplear Rational Rose, ya que existen otras herramientas con características similares que se ajustan más a las políticas de migración a software libre implementadas en la UCI.

1.8.2 Visual Paradigm para UML v6.04

Visual Paradigm para UML 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 más rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código

desde diagramas y generar documentación. Además proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML (14).

Entre las múltiples características y funcionalidades de esta herramienta se pueden destacar:

- Soporte de UML versión 2.1.
- Ingeniería inversa - Código a modelo, código a diagrama.
- Generación de código - Modelo a código, diagrama a 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.
- Diagramas de flujo de datos.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos - Desde sistemas gestores de bases de datos (DBMS) existentes a diagramas de Entidad-Relación.
- 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 XML.

Visual Paradigm para UML es una herramienta con soporte multiplataforma que permite llevar el control de los artefactos generados de manera sencilla y ágil. Algunas de las funcionalidades de generación de códigos sobre diagramas integrando el lenguaje Java, permiten tomar esta herramienta como ejemplo para algunas funcionalidades que se desean desarrollar en la presente tesis, por lo que se decide el uso de Visual Paradigm para UML como herramienta CASE.

1.8.3 Lenguaje de modelado UML

UML es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos (15). Se decide el uso de este lenguaje de modelado por las características que se describen a continuación:

- UML es un lenguaje de modelado de propósito general que pueden usar todos los modeladores. Está basado en el común acuerdo de gran parte de la comunidad informática.
- UML no pretende ser un método de desarrollo completo. No incluye un proceso de desarrollo paso a paso. UML incluye todos los conceptos que se consideran necesarios para utilizar un proceso moderno iterativo, basado en construir una sólida arquitectura para resolver requisitos dirigidos por casos de uso.
- Debe ser tan simple como sea posible, aunque manteniendo la capacidad de modelar toda la gama de sistemas que se necesita construir. UML necesita ser lo suficientemente expresivo para manejar todos los conceptos que se originan en un sistema moderno, tales como la concurrencia y distribución, así como también los mecanismos de la ingeniería de software, como son la encapsulación y componentes.
- Debe ser un lenguaje universal, como cualquier lenguaje de propósito general.
- Pretende imponer un estándar mundial.
- Es orientado a objetos, permitiéndole al programador que organice su programa de acuerdo con abstracciones de más alto nivel, siendo estas más cercanas a la forma de pensar de las personas.

1.9 Lenguaje de programación y tecnologías a utilizar

1.9.1 Java

Java es un lenguaje de programación de uso general, su sintaxis y semántica son lo bastante completas para abarcar programas de todo tipo. Se trata de un lenguaje orientado a objetos, originalmente desarrollado por un grupo de ingenieros de la empresa Sun Microsystems, fue utilizado por Netscape posteriormente como base para JavaScript. Si bien su uso se destaca en la Web, sirve para crear todo tipo de aplicaciones (locales, intranet o internet). Java puede ser ejecutado en múltiples plataformas. Es uno de los escasos lenguajes cuyos programas pueden ser transportados de sistema operativo, computadora o entorno, sin necesidad de cambiar el código.

Esta característica ha sido la que alimentó su auge en Internet: una red que pueda ser accedida desde cualquier máquina, debe apelar a tecnologías multiplataforma. Java ha sido concebido, desde sus orígenes, como un lenguaje capaz de producir código totalmente transportable (16). Esa filosofía y su

facilidad para crear aplicaciones para redes TCP/IP ha hecho que sea uno de los lenguajes más utilizados en la actualidad.

1.9.2 JavaScript

JavaScript es una de las múltiples maneras que han surgido para extender las capacidades del lenguaje HTML. Al ser la más sencilla, es por el momento la más extendida. JavaScript no es un lenguaje de programación propiamente dicho. Es un lenguaje script u orientado a documento, como pueden ser los lenguajes de macros que tienen muchos procesadores de texto. Haciendo uso de Javascript no se puede hacer un programa, sirve principalmente para mejorar la gestión de la interfaz cliente/servidor. Un código JavaScript insertado en un documento HTML permite reconocer y tratar localmente, es decir, en el cliente, los eventos generados por el usuario. Estos eventos pueden ser el recorrido del propio documento HTML o la gestión de un formulario (17).

1.9.3 PHP

El **PHP** (Hypertext Preprocessor), es un lenguaje interpretado de alto nivel, embebido en páginas HTML y ejecutado en el servidor. Con PHP se pueden realizar múltiples acciones, por ejemplo el procesamiento de información en formularios, foros de discusión, manipulación de cookies y páginas dinámicas. Un sitio con páginas dinámicas es el que permite interactuar con el visitante, de modo que cada usuario que visita la página vea la información modificada para requisitos articulares.

Una de sus características más potentes es su suporte para gran cantidad de bases de datos. Entre su soporte pueden mencionarse InterBase, mSQL, MySQL, Oracle, Informix, PosgreSQL, entre otras. PHP también ofrece la integración con las varias bibliotecas externas, que permiten que el desarrollador haga casi cualquier cosa, desde generar documentos en PDF hasta analizar código XML (18).

1.9.4 Applet de Java

En el lenguaje de programación Java, un applet es un programa que puede incrustarse en un documento HTML, lo que es equivalente a incrustarlo en una página web. Esto permite crear aplicaciones que pueden ser ejecutadas con solo cargar una página en un navegador web (19).

Las principales ventajas de los applets son:

- Multiplataforma (funcionan en Linux, Windows, Mac OS, y en cualquier sistema operativo para el cual exista una Java Virtual Machine (JVM))
- El mismo applet puede trabajar en "todas" las versiones de Java, y no sólo la última versión.
- Es soportado por la mayoría de los navegadores web.
- Puede ser almacenado en la memoria cache de la mayoría de los navegadores web, de modo que se cargará rápidamente cuando se vuelva a cargar la página web.
- Puede tener acceso completo a la máquina en la que se está ejecutando, si el usuario lo permite.
- Puede ejecutarse con velocidades comparables a la de otros lenguajes compilados, como C++ dependiendo de la versión de la JVM.
- Puede trasladar el trabajo del servidor al cliente, haciendo una solución web más escalable tomando en cuenta el número de usuarios / clientes.

Estas aplicaciones también poseen algunas desventajas, como son:

- Requiere el accesorio de Java, que no está disponible por defecto en todos los navegadores web.
- No puede iniciar la ejecución hasta que la JVM esté en funcionamiento, y esto puede tomar tiempo la primera vez que se ejecuta un applet.
- Si no está firmado como confiable, tiene un acceso limitado al sistema del usuario - en particular no tiene acceso directo al disco duro del cliente o al portapapeles.
- Algunas organizaciones sólo permiten la instalación de software a los administradores. Como resultado, muchos usuarios (sin privilegios para instalar el accesorio en su navegador) no pueden ver los applet.

Haciendo uso de los applets, se pueden crear herramientas orientadas a plataformas web como si se estuvieran construyendo aplicaciones de escritorio. Los mismos definen una serie de clases y librerías que facilitan el trabajo con gráficos e imágenes de manera general, lo que hace que la construcción de

aplicaciones donde se generen diagramas y animaciones a través de componentes visuales facilita aún más el trabajo de los programadores.

1.9.5 HTML

Hyper Text Markup Language (Lenguaje de Marcado de Hipertexto), es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. HTML se escribe en forma de "etiquetas", rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir un *script* (por ejemplo JavaScript), el cual puede afectar el comportamiento de los navegadores web y otros procesadores de HTML (20).

1.10 Lenguajes de programación estudiados para la generación de código

1.10.1 C++

C++ es un potente lenguaje de programación que apareció en 1980. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. C++ abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos. Las principales características radican en las facilidades que proporciona para la programación orientada a objetos y para el uso de plantillas o programación genérica. Posee además un conjunto de propiedades difíciles de encontrar en otros lenguajes de alto nivel:

- Posibilidad de redefinir los operadores (sobrecarga de operadores).
- Identificación de tipos en tiempo de ejecución (RTTI).

En la actualidad, el C++ es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones. El C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original (21).

1.10.2 CSharp (C#)

C# es un robusto lenguaje creado por la compañía Microsoft especialmente para su plataforma .NET, con el objetivo de dotar a los desarrolladores de un lenguaje excepcional que les permitiera elevar la calidad de sus productos y disminuir el tiempo de producción de los mismos. Su sintaxis y estructuración de C# es muy similar a la C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Algunas de las principales características de este moderno lenguaje son (22):

- Sencillez.
- Modernidad.
- Orientación a objetos.
- Orientación a componentes.
- Gestión automática de memoria.

1.10.3 Justificación de los lenguajes de programación a emplear

Luego del estudio de los lenguajes de programación posibles a utilizar en la construcción de la propuesta de solución, se decidió el uso de Java, como lenguaje para la implementación de la aplicación. Esta selección se justifica debido a las numerosas ventajas que brinda para el desarrollo de aplicaciones orientadas a la Web, como es el caso de los applet de Java. Este lenguaje presenta numerosas ventajas sobre otros lenguajes empleados para el trabajo en la Web, superior a Javascript por estar orientado a objetos y por las numerosas clases y librerías para el trabajo con gráficos e imágenes. También es superior a PHP por permitir directamente el trabajo sobre interfaces visuales, permitiendo el control de los componentes de manea muy sencilla y el uso de las ventajas de la programación orientada a objetos.

Se decide la construcción de un applet, por las ventajas que los mismos brindan para el trabajo con gráficos e imagen en las plataformas web. Los lenguajes Java, C++ y C#, serán utilizados como ejemplos para la obtención de las sintaxis de los códigos de los mismos, ya que la herramienta propuesta para la solución deberá generar códigos de algoritmos simples en estos tres lenguajes. Se utilizará HTML como lenguaje de marcado para que la aplicación propuesta sea accesible por los distintos navegadores web.

1.11 Servidor web (24)

Un servidor web es un programa ejecutado continuamente en una computadora (generalmente llamada servidor), el cual se encarga de contestar de forma adecuada las peticiones de ejecución que le hará un usuario mediante la red; entregando como resultado una página web (código HTML) o información de todo tipo de acuerdo con los comandos solicitados, implementando el protocolo HTTP.

El servidor HTTP Apache ofrece un mejor uso de las especificaciones de HTTP existentes, es libre y de distribución gratuita, e implementa funcionalidades demandadas con frecuencia, tales como:

- Personalizar las respuestas a los errores y problemas.
- Establecer ficheros que son devueltos por el servidor en las respuestas a los errores y problemas, realizando diagnósticos “en el momento” para los usuarios y el administrador.
- Aceptar un número ilimitado de directivas Alias y Redirect.
- Admitir servidores *“multi-home”*, que permiten distinguir entre las peticiones de diferentes direcciones IP.

Para el desarrollo de la aplicación se utilizará Apache como servidor web, por la eficiencia que este brinda además de ser multiplataforma, también tiene una gran comunidad que lo respalda y su uso no está restringido por una licencia. Es un servidor altamente configurable y fácil de usar. Es muy extensible, permitiendo a cualquier usuario implementar módulos para alguna función específica, además de poseer abundante documentación para usuarios de poca experiencia. Se utilizará la versión 2.2.6.

1.12 Entornos de desarrollo integrado (IDE)

1.12.1 Netbeans v6.9.1

El IDE NetBeans es un reconocido entorno de desarrollo integrado disponible para Windows, Mac, Linux y Solaris. El proyecto NetBeans está formado por un IDE de código abierto y una plataforma de aplicación que permite a los desarrolladores crear con rapidez aplicaciones web, empresariales, de escritorio y móviles utilizando la plataforma Java. El proyecto de NetBeans está apoyado por una comunidad de

desarrolladores dinámica y ofrece documentación y recursos de formación exhaustivos, así como una amplia selección de complementos de terceros (24).

1.12.2 Eclipse

Eclipse es un entorno de desarrollo integrado, desarrollado por la firma IBM. Es actualmente desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto, multiplataforma, que incluye un conjunto de productos complementarios, capacidades y servicios. Es un IDE multiplataforma de código abierto, considerado como un armazón en el que se pueden montar herramientas de programación en cualquier lenguaje, mediante la implementación de los módulos adecuados (25).

1.12.3 Justificación del IDE seleccionado

Al realizarse un estudio sobre las potencialidades de los entornos integrados de desarrollo descritos, se decide el uso del IDE Netbeans 6.9.1, el cual posee una poderosa integración con el lenguaje Java ya que el mismo fue construido con ese lenguaje, además es libre, gratuito y sin restricciones de uso. El equipo de desarrollo posee un buen dominio del mismo, lo que contribuye al buen manejo y aprovechamiento óptimo de las características que el mismo brinda.

2. Capítulo 2. Características de la propuesta de solución. Exploración y Planificación

2.1 Propuesta del sistema

El presente trabajo de diploma propone el desarrollo de una herramienta web basada en la tecnología Applet usando el lenguaje Java, tomando el IDE NetBeans en su versión 6.9.1 como editor de código para la generación del código fuente. Esta herramienta permitirá el diseño de diagramas de flujos de datos y la generación de los códigos de los algoritmos diseñados en lenguajes Java, C++ y C# estudiados en la UCI, como apoyo al aprendizaje de temas relacionados con el diseño de algoritmos impartidos en la asignatura Introducción a la Programación.

2.1.1 Requerimientos funcionales de la propuesta de solución

Los requerimientos funcionales son aquellas capacidades que debe poseer una aplicación para dar solución a un problema dado. La herramienta propuesta para la solución contará, en su versión inicial, con las siguientes funcionalidades:

- Diseñar los DFD a través de los eventos del mouse o ratón.
- Visualizar los paneles que posibilitan la introducción de los datos que forman las diferentes estructuras de los algoritmos diseñados, con el objetivo de hacer más didáctica e instructiva la herramienta.
- Validar la correcta estructura de los diagramas diseñados (que no falte un estado inicial, un estado final y que no exista ningún componente sin conectar a través de los flujos correspondientes).
- Editar los datos de cada uno de los componentes del DFD.
- Generar código de un DFD en el lenguaje Java.
- Generar código de un DFD en el lenguaje C++.
- Generar código de un DFD en el lenguaje C#.

2.1.2 Requerimientos no funcionales de la propuesta de solución

Los requerimientos no funcionales son los que especifican criterios que pueden usarse para juzgar la operación de un sistema. Estos responden a las características del sistema en cuanto a funcionalidad, usabilidad, confiabilidad, compatibilidad con hardware y software, especificaciones del producto, etc.

- Interfaz de usuario: la aplicación debe presentar una interfaz sencilla, diseñada haciendo uso de colores claros (blanco, gris, azul) fácil de manipular por personas con pocos conocimientos de informática.
- Requerimientos de rendimiento: la eficiencia del producto estará determinada en gran medida por el aprovechamiento de los recursos que se disponen en el modelo cliente/servidor y la velocidad de la infraestructura de red. La aplicación propuesta debe ser rápida y el tiempo de respuesta debe ser inferior a los 10 segundos al cargar el applet en el navegador web de la computadora cliente.
- Requerimiento de portabilidad: la herramienta podrá ser utilizada en cualquier sistema operativo: Windows NT en adelante, Mac OS o cualquier distribución de Linux. El servidor web puede estar instalado en la misma computadora que el navegador donde se ejecuta la herramienta sin ocasionar problema alguno.
- Requerimientos de software: en las computadoras clientes se requiere de un navegador web con compatibilidad al lenguaje Java y deberá estar instalada la JVM que es la encargada de ejecutar el contenido del applet.
- Requerimientos de hardware:

  Las computadoras clientes deberán contar con 128 megabytes de RAM como mínimo.

  La computadora donde esté instalado el servidor web deberá tener como mínimo 256 megabytes de RAM y una capacidad de almacenamiento superior a los 10 gigabytes de disco duro.

2.2 Modelo de dominio o modelo conceptual

El modelo de dominio o modelo conceptual es la representación visual de los diferentes elementos significativos del mundo real y sus interacciones para resolver un problema determinado. En la presente investigación el modelo de dominio es utilizado como una herramienta de comunicación para permitir un

mejor entendimiento de las relaciones entre los elementos que de una forma u otra interactuarán en la herramienta.

Personal relacionado con el sistema

Se define como personal relacionado con el sistema a todo aquel individuo que de una forma u otra interactúa con la herramienta así como aquel que está implicado en el desarrollo de la misma.

- **Desarrollador:** es la persona encargada de realizar la implementación de las funcionalidades de la herramienta.
- **Usuario:** persona que interactúa con los componentes de la herramienta construida.

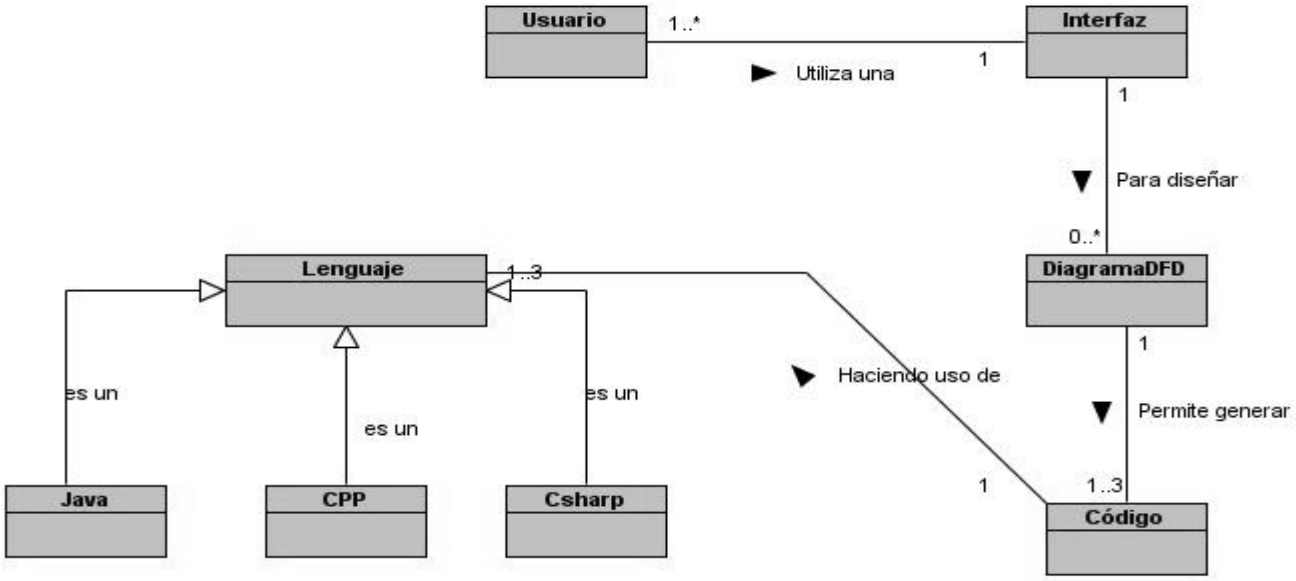


Ilustración 1. Modelo de dominio.

2.3 Fase de exploración

La metodología XP comienza con la fase de exploración, etapa en la cual se define el alcance general del proyecto, además los clientes exponen a grandes rasgos las posibles Historias de Usuarios (HU) de interés para la primera entrega del producto. El equipo de desarrollo, al mismo tiempo, se familiariza con

las herramientas, tecnologías y prácticas que se emplearán en el desarrollo del proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema. En esta fase, las estimaciones realizadas son primarias, y pueden variar cuando se analicen más en detalle en cada iteración. Esta fase toma de pocas semanas a pocos meses, dependiendo de la familiaridad que posean los programadores con las tecnologías empleadas (26).

2.3.1 Historias de usuarios

La metodología XP, emplea las historias de usuarios para representar una breve descripción del comportamiento del sistema, en estas se debe de emplear una terminología entendible para el cliente evitando el uso de un lenguaje técnico que dificulte su compresión. Se realiza una HU para cada funcionalidad del sistema. Se emplean además para realizar estimaciones de tiempo y para el plan de lanzamiento, reemplazan un gran documento de requisitos y rigen la creación de las pruebas de aceptación (26).

La principal diferencia entre los tradicionales documentos de especificación funcional y estas historias de usuarios se basa en el nivel de detalle requerido. Las HU deben poseer el nivel de detalle mínimo para que los programadores puedan realizar una estimación del tiempo que demorará su implementación. Deben poder ser programadas en un tiempo entre una y tres semanas, si la estimación es superior a las tres semanas, entonces la historia de usuario debe ser dividida en dos, y si la estimación es inferior a una semana, entonces debe combinarse con otra historia de usuario (26).

Los aspectos fundamentales de las historias de usuario son:

- Tarjeta**:** en ellas se almacena suficiente información para identificar y detallar la historia.
- Conversación: los programadores y los clientes deben discutir la historia para la ampliación de los detalles. (verbalmente cuando sea posible pero documentada si se requiere confirmación).
- Pruebas de aceptación: permiten verificar que la historia de usuario ha sido implementada correctamente.

Los datos de las HU son recogidos en tarjetas como la plantilla siguiente:

Historia de usuario	
No.: número sucesivo a partir de uno.	**Nombre:** identifica la historia de usuario en cuestión.
Usuario: quien ejecuta la historia de usuario.	
Prioridad del negocio: define la relevancia e impacto de la historia de usuario para el negocio de acuerdo con las necesidades del usuario.	**Nivel de complejidad:** define la complejidad técnica que supone desarrollar la historia de usuario desde el punto de vista del programador.
Puntos de estimación: permiten estimar duración de la implementación.	**Iteración asignada:** precisa la iteración a la que pertenece la historia de usuario.
Descripción: explica en qué consiste la historia de usuario, teniendo en cuenta las acciones realizadas por el usuario y la respuesta brindada por el sistema.	
Información adicional (Observaciones): información extra que se estime agregar para hacer más comprensible la historia de usuario. Por ejemplo: conceptos, post-condiciones, relaciones con otros requisitos, etc.	

Tabla 2. Plantilla de historia de usuario.

Como resultado de la fase de exploración se identificaron las siguientes HU:

Historia de usuario	
No. 1	**Nombre:** Dibujar componentes de los DFD.
Usuario: desarrollador	
Prioridad del negocio: alta	**Nivel de complejidad:** alta
Puntos de estimación: 2	**Iteración asignada:** 1
Descripción: en esta HU se implementarán las funcionalidades necesarias para posibilitar el diseño de los DFD haciendo uso de los componentes visuales empleados con tales fines.	
Información adicional (Observaciones): se implementarán las funcionalidades necesarias para dibujar componentes como Inicio, Asignación, Bifurcación, Repetición, Salida, Fin y Flujos.	

Tabla 3. HU Dibujar componentes de los DFD.

Historia de usuario	
No. 2	**Nombre:** Validar diseño de los DFD
Usuario: desarrollador	
Prioridad del negocio: alta	**Nivel de complejidad:** alta
Puntos de estimación: 1	**Iteración asignada:** 1
Descripción: en la presente HU se implementarán las funcionalidades necesarias para validar la correcta estructura de los diagramas diseñados.	
Información adicional (Observaciones): un diagrama correctamente diseñado es aquel que sólo contiene un estado inicial, un estado final, y en el que todos los componentes están conectados a través de los flujos (flechas).	

Tabla 4. HU Validar diseño de los DFD.

Historia de usuario	
No. 3	**Nombre:** Editar contenido de los componentes de los DFD.
Usuario: desarrollador	
Prioridad del negocio: alta	**Nivel de complejidad:** alta
Puntos de estimación: 1	**Iteración asignada:** 1
Descripción: en la presente HU se implementarán las funcionalidades necesarias para la edición de los contenidos de los componentes presentes en un DFD.	
Información adicional (Observaciones): una vez dibujado un componente en un DFD, es necesario tener la posibilidad de editar el contenido correspondiente a ese componente, lo que posibilita cambiar los datos de un algoritmo sin necesidad de rediseñar el DFD completo.	

Tabla 5. HU Editar contenido de los componentes de los DFD.

Historia de usuario	
No. 4	**Nombre:** Generar código en el lenguaje Java.
Usuario: desarrollador	

Prioridad del negocio: alta	**Nivel de complejidad:** alta
Puntos de estimación: 1	**Iteración asignada:** 2
Descripción: permite generar el código del algoritmo diseñado mediante el DFD en el lenguaje Java.	
Información adicional (Observaciones): tomando como referencia el DFD diseñado se debe generar el código del algoritmo en el lenguaje Java.	

Tabla 6. HU Generar código en el lenguaje Java.

Historia de usuario	
No. 5	**Nombre:** Generar código en el lenguaje C++.
Usuario: desarrollador	
Prioridad del negocio: alta	**Nivel de complejidad:** alta
Puntos de estimación: 1	**Iteración asignada:** 2
Descripción: permite generar el código del algoritmo diseñado mediante el DFD en el lenguaje C++.	
Información adicional (Observaciones): tomando como referencia el DFD diseñado se debe generar el código del algoritmo en el lenguaje C++.	

Tabla 7. HU Generar código en el lenguaje C++.

Historia de usuario	
No. 6	**Nombre:** Generar código en el lenguaje C#.
Usuario: desarrollador	
Prioridad del negocio: alta	**Nivel de complejidad:** alta
Puntos de estimación: 1	**Iteración asignada:** 3
Descripción: permite generar el código del algoritmo diseñado mediante el DFD en el lenguaje C#.	

Información adicional (Observaciones): tomando como referencia el DFD diseñado se debe generar el código del algoritmo en el lenguaje C#.

Tabla 8. HU Generar código en el lenguaje C#.

2.4 Fase planificación

En la fase de planificación el cliente establece una prioridad para cada historia de usuario, luego los programadores realizan una estimación de esfuerzo para desarrollar cada una de ellas. Aquí se toman los acuerdos correspondientes a la primera entrega y se pacta un cronograma en conjunto con el cliente. Esta fase dura unos pocos días. Los programadores establecen las estimaciones de esfuerzo asociado a la implementación de las HU tomando como medida el punto. Un punto equivale a una semana ideal de programación, y las HU generalmente valen de uno a tres puntos. El equipo de desarrollo mantiene un registro de la velocidad de desarrollo, que se establece por los puntos por iteración, basándose principalmente en la suma de los puntos de las historias de usuario que fueron implementadas en la última iteración (26).

La planificación puede realizarse basándose en el tiempo o el alcance, la velocidad del proyecto se utiliza para saber cuántas historias de usuario se pueden implementar antes de una fecha determinada o cuánto tiempo tomaría implementar un conjunto de historias de usuario. Para realizar la planificación por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden desarrollar. Si se planifica por el alcance del sistema, se divide la suma de los puntos de las historias de usuarios seleccionadas entre la velocidad del proyecto, y se obtiene el número de iteraciones necesarias para completar la implementación (26).

2.4.1 Estimación de esfuerzo por HU

Tomando como referencia las HU generadas en la fase de exploración, se realizó la confección de la siguiente estimación de esfuerzo por HU:

Historias de Usuario	**Puntos de estimación.**

Dibujar componentes de los DFD	2 semanas
Validar diseño de los DFD	1 semana
Editar contenidos de los componentes de los DFD	1 semana
Generar código en el lenguaje Java	1 semana
Generar código en el lenguaje C++	1 semana
Generar código en el lenguaje C#	1 semana

Tabla 9. Estimación de esfuerzo por HU.

2.4.2 Plan de iteraciones

En la presente fase se incluyen varias iteraciones sobre el sistema antes de ser liberado, 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 sirva de base durante todo el desarrollo del proyecto. Esto podría lograrse escogiendo las historias de usuario que fuercen el establecimiento de esta arquitectura, sin embargo, esto no siempre se logra puesto que es el cliente quien decide qué historias de usuario se implementarán en cada iteración con el objetivo de maximizar el valor del negocio. Al finalizar la última iteración el sistema estará listo para entregar en producción. Los elementos que deben tenerse en cuenta durante la elaboración del plan de iteraciones son las historias de usuarios. Todo el trabajo a realizar en una iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable (26).

Una vez definidas las historias de usuarios y realizada la estimación del esfuerzo requerido para la realización de cada una de ellas, se decide realizar el desarrollo del sistema en tres iteraciones, las cuales se describen con más detalle a continuación.

Iteración 1

Esta iteración se tiene como objetivo darle cumplimiento a las historias de usuario que se consideraron como las de mayor importancia para el sistema. Al finalizar esta iteración, se contará con todas las funcionalidades especificadas en las historias de usuario 1 y 2, referentes al diseño y validación de los DFD.

Iteración 2

Esta iteración tiene como objetivo el desarrollo de las funcionalidades descritas en las historias de usuario 3 y 4, en las cuales se describe la necesidad de editar los contenidos de los componentes del DFD y generar el código de los algoritmos diseñados en los lenguajes Java.

Iteración 3

En la tercera iteración, se implementarán las funcionalidades descritas en las historias de usuarios números 5 y 6, que se refieren a la generación de código de los algoritmos diseñados en los lenguajes de programación C++ y C#. Al finalizar dicha iteración, el usuario podrá seleccionar estos lenguajes para realizar la generación del código al igual que con el lenguaje Java.

2.4.3 Plan de duración de las iteraciones

Como parte del ciclo de vida del desarrollo de un proyecto guiado por la metodología de desarrollo XP, se crea el plan de duración de las iteraciones. Este plan tiene como finalidad mostrar la duración de cada iteración, así como mostrar el orden en el que se desarrollarán las HU pertenecientes a cada una de las iteraciones.

Iteración	Historias de Usuario	Duración
Iteración 1	Dibujar componentes de los DFD.	3 Semanas
	Validar diseño de los DFD.	
Iteración 2	Editar contenido de los componentes de los DFD	2 Semanas
	Generar código en el lenguaje Java.	
Iteración 3	Generar código en el lenguaje C++. Generar código en el lenguaje C#.	2 Semanas

Tabla 10. Plan de duración de las iteraciones.

3. Construcción y validación de la propuesta de solución

3.1 Diseño de la propuesta de solución

Para el diseño de las aplicaciones, XP no requiere de la presentación de los sistemas mediante diagramas de clases utilizando notación UML, en su lugar se emplean otras técnicas como las tarjetas CRC (Clase, Responsabilidad, Colaboración). No obstante, los diagramas pueden emplearse siempre y cuando influyan en el mejoramiento de la comunicación entre los desarrolladores, que no sea una carga más su mantenimiento, no sean extensos y se enfoquen en la información importante.

3.1.1 Tarjetas CRC

Para la realización del diseño de un sistema como un equipo, se deben tener en cuenta tres principios: Cargo o Clase, Responsabilidad y Colaboración (CRC). Las tarjetas CRC permiten desprenderse de los métodos de trabajo basados en procedimientos y trabajar en una metodología basada en objetos, además posibilitan que todo el equipo trabaje en las labores del diseño.

Cada tarjeta CRC representa un objeto, el nombre de la clase se coloca a modo de título de la tarjeta, las responsabilidades se colocan a la izquierda, y las clases que se implican en cada responsabilidad a la derecha, en la misma línea que su requerimiento correspondiente. Partiendo de que la metodología XP plantea la programación por encima de cualquier documentación, propone para el diseño y modelación de las clases el uso de las tarjetas CRC o los diagramas de clases UML, se decide el uso de las tarjetas CRC debido a que estas son muy fáciles de elaborar y entender, se considera que estas son suficientes para describir las clases involucradas en el diseño de la propuesta de solución.

A continuación se muestra una plantilla de tarjeta CRC, la que será empleada en la presente investigación.

Tarjeta CRC
Clase: nombre de la clase que se está modelando.
Súper clase: nombre de la clase de la que se deriva en la herencia.

Sub clase(s): nombre de la(s) clase(s) hija en la herencia.	
Responsabilidades: descripción de alto nivel del propósito de la clase.	**Colaboración:** otras clases con las que tiene relación para el cumplimiento de las responsabilidades.

Tabla 11. Plantilla de Tarjeta CRC.

Las clases involucradas en la construcción de la herramienta propuesta para la solución son las registradas en las siguientes tarjetas CRC:

Tarjeta CRC	
Clase: JInterfazPrincipalDFD	
Súper clase: javax.swing.JApplet.	
Sub clase(s): -.	
Responsabilidades: Contenedora principal de todos los componentes que brindan la posibilidad al usuario de interactuar con el sistema.	**Colaboración:** java.awt.*; java.io.*; java.net.URL; java.util.*; javax.imageio.ImageIO; javax.swing.*;

Tabla 12. Tarjeta CRC JInterfazPrincipalDFD.

Tarjeta CRC
Clase: JComponenteDFD
Súper clase: -.
Sub clase(s):

JCompAuxDFD	
Responsabilidades: A través de ella se pueden generar los flujos y permite que los demás componentes hereden las características comunes.	**Colaboración:** -.

Tabla 13. Tarjeta CRC JComponenteDFD.

Tarjeta CRC	
Clase: JCompAuxDFD	
Súper clase: JComponenteDFD	
Sub clase(s):	
Responsabilidades: Declara las características comunes a los componentes Inicio, Asignación, Bifurcación, Repetición, Salida y Fin, con los cuales de diseñan los DFD.	**Colaboración:**

Tabla 14. Tarjeta CRC JCompAuxDFD.

3.2 Arquitectura del sistema

Denominada Arquitectura Lógica, la Arquitectura de Software consiste en un conjunto de patrones y abstracciones coherentes que proporcionan un marco de referencia necesario para la construcción de un sistema de información. Establece los fundamentos para que los miembros del equipo de desarrollo trabajen sobre una línea común que permita alcanzar los objetivos del producto que se quiere construir cubriendo todas las necesidades. Define además, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos (28).

La arquitectura de software debe ser seleccionada y diseñada sobre la base de los objetivos y las restricciones del sistema que se desea construir. Los objetivos son prefijados para el sistema de

información, no sólo los de tipo funcional, sino también los de mantenimiento, auditoría, flexibilidad e interacción con otros sistemas de información. Las restricciones son aquellas limitaciones derivadas de las tecnologías disponibles para realizar la implementación del sistema. La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema, aplicación o programa y tiene la responsabilidad de definir los módulos principales, las responsabilidades e interacción entre cada uno de estos módulos en cuanto a: control y flujo de datos, secuencia de la información, protocolos de interacción y comunicación y a la ubicación en hardware (28).

Para el desarrollo de la herramienta propuesta para la solución se decide el uso de la arquitectura cliente-servidor. Esta arquitectura utiliza un conjunto de estándares, reglas y procesos, que permite integrar un amplio conjunto de aplicaciones informáticas. Esta arquitectura debe su nombre a que la lógica de su funcionamiento está compuesta por un cliente que es el encargado de realizar las peticiones y órdenes que debe realizar el servidor. El servidor subordinándose al diseño e implementación de su programación interna, resuelve y responde a las peticiones del cliente. Esta arquitectura responde a las necesidades de esta investigación permitiendo que un servidor (en el caso particular de esta investigación el servidor contiene el applet, responsable de las funciones principales) pueda responder peticiones de múltiples clientes (la interfaz de usuario) concurrentemente. Además, permite realizar cambios en la programación del servidor de manera que sea transparente al cliente. Esta arquitectura permite que se realicen actualizaciones o reemplazo tecnológico sin que influya directamente en el usuario. La arquitectura cliente-servidor se caracteriza por ser flexible y escalable permitiendo en un futuro realizar cambios y modificaciones de forma sencilla.

3.3 Diagrama de componentes

A través de un diagrama de componentes se muestran las dependencias lógicas entre los componentes de un software, sean éstos fuentes, binarios o ejecutables. Ilustran las diferentes piezas del software, los controladores embebidos, etc. Los diagramas de componentes prevalecen en el campo de la arquitectura de software, pero pueden ser utilizados para modelar y documentar cualquier arquitectura de sistemas. Estos diagramas se relacionan con los diagramas de clases, ya que un componente normalmente se corresponde con una o más clases, interfaces o colaboraciones. Un diagrama de componentes tiene un nivel de abstracción más alto que un diagrama de clases, usualmente un componente se implementa por una o más clases (u objetos) en tiempo de ejecución (29).

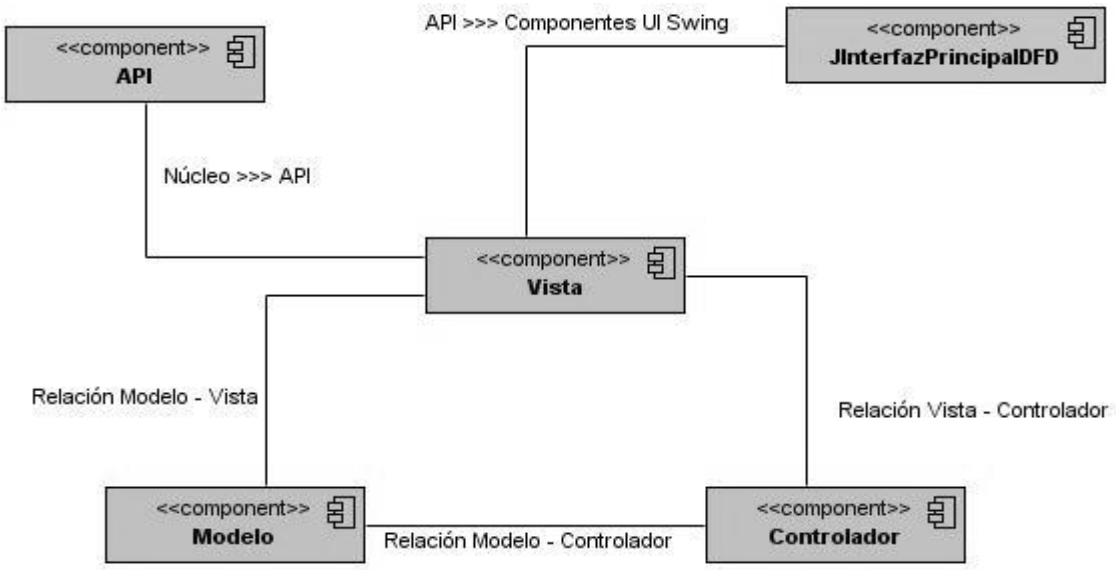


Ilustración 2. Diagrama de componentes.

3.4 Patrones de diseño

Los patrones de diseño nombran, abstraen e identifican los aspectos clave de un diseño estructurado común, que lo hace útil para la creación de diseños orientados a objetos reutilizables. Definen una posible solución correcta para un problema de diseño dentro de un contexto dado, describiendo las cualidades invariantes de todas las soluciones (29).

Algunas características de los patrones de diseño son:

- Solución estándar para un problema común de programación.
- Técnica para flexibilizar el código satisfaciendo ciertos criterios.
- Proyecto o estructura de implementación que logra una finalidad determinada.
- Lenguaje de programación de alto nivel.
- Manera práctica de describir aspectos de la organización de un programa.
- Conexiones entre componentes de programas.
- Establecen la forma de un diagrama de objeto o modelo de objeto.

3.4.1 Modelo-Vista-Controlador (MVC)

El patrón de arquitectura MVC, conocido por sus siglas en inglés del Model-View-Controller, permite realizar la programación multicapa, separando en tres componentes distintos los datos de una aplicación, la interfaz del usuario y la lógica de control. Este patrón se ve usualmente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo representa la información con la que se trabaja en la aplicación y el controlador representa la lógica del negocio, los niveles que lo conforman son los siguientes (29):

- Modelo: lo constituyen las colecciones de datos que almacenan la información referente a los componentes de los DFD.
- Vista: conformada por el formulario principal del applet, en el que se encuentran los componentes visuales pertenecientes al paquete Swing de Java.
- Controlador: es la clase principal de la aplicación, donde se encuentran las funcionalidades principales del applet, aquí se implementan las funcionalidades que dan respuesta a los requisitos funcionales de la herramienta, así como el control y respuesta a las acciones realizadas a través de los componentes de la interfaz.

3.5 Estándares de codificación

Los estándares de codificación, también conocidos como estilos de programación o convenciones de código, no son más que convenios que se toman para escribir el código fuente de un proyecto en ciertos lenguajes de programación. Estos estándares elevan la capacidad de mantenimiento del código, sirven como punto de referencia a los programadores, mantienen un estilo de programación y ayudan a mejorar el proceso de codificación, haciéndolo mucho más eficiente.

El uso de técnicas de codificación sólidas y realizar buenas prácticas de programación con el objetivo de generar código fuente eficiente es de vital importancia para obtener un producto de alta calidad así como para obtener un buen rendimiento del mismo. Si un estándar de codificación bien definido y técnicas de programación adecuadas, se aplican sobre un proyecto de software, es posible que este se convierta en un proyecto fácil de mantener y comprender (30).

Declaraciones

- **Cantidad de declaraciones por línea**

Se utilizó una declaración por línea usando un espacio entre el tipo y el identificador ya que esto facilita los comentarios. No se emplearon declaraciones de diferentes tipos en la misma línea. La estructura empleada fue la siguiente:

- **Inicialización**

Las variables locales se inicializaron donde mismo fueron declaradas. La única razón para no hacerlo es si el valor inicial depende de algunos cálculos que se realicen.

- **Colocación**

Las declaraciones se colocan sólo al inicio de cada bloque (cualquier código encerrado por llaves “{” y “}”.

```
void myMethod() {
    int int1 = 0;          // comienzo del bloque del método

    if (condition) {
        int int2 = 0;      // comienzo del bloque del "if"
        ...
    }
}
```

Las excepciones serán los índices de los bucles **for**, que en Java, C++ y C# se pueden declarar en la propia sentencia **for**.

```
for (int i = 0; i < maximoVueltas; i++) { ... }
```

Se evitarán las declaraciones locales que oculten declaraciones de niveles superiores, por ejemplo no declarar una variable ya declarada en un bloque externo.

Sentencias

- **Sentencias simples**

Las sentencias son declaradas a lo sumo una por línea. Ejemplificado quedaría de la siguiente manera:

```
argv++;                 // Correcto
argc--;                 // Correcto
argv++; argc--;         // EVITAR!
```

- **Sentencias compuestas**

Las sentencias compuestas son aquellas que contienen listas de sentencias encerradas entre llaves "{" sentencias "}". Para su estructuración se tienen en cuenta las siguientes reglas.

- ✓ Las sentencias encerradas se comienzan un nivel más adentro que las sentencias compuestas.
- ✓ La llave de apertura se coloca al final de la línea donde se declara la sentencia compuesta, la llave de cierre se coloca en una nueva línea al mismo nivel que al principio de la sentencia compuesta.
- ✓ Las llaves se usan en todas las sentencias, incluyendo las simples, cuando forman parte de una estructura de control. Esto hace más sencillo el adicionar sentencias sin incluir errores accidentalmente por no colocar las llaves.

- **Sentencias if, if-else**

La codificación de las estructuras de control condicionales son las siguientes:

```
if (condicion) {
    sentencias;
}

if (condicion) {
    sentencias;
} else {
    sentencias;
}
```

- **Sentencias for**

La estructura seleccionada para las sentencias **for** es la siguiente:

```
for (inicializacion; condicion; actualizacion) {
    sentencias;
}
```

3.6 Desarrollo de las iteraciones

3.6.1 Primera iteración

En esta iteración se realizaron las acciones necesarias para implementar las HU 1 y 2, que tienen el objetivo de obtener las funcionalidades necesarias para realizar el diseño de los DFD y comprobar que los mismos fueron diseñados correctamente.

Historia de Usuario	**Tiempo de Implementación(semanas)**	
	Estimación	**Real**
Dibujar componentes de los DFD	2	3
Validar diseño de los DFD	1	1

Tabla 15. HU abordadas en la primera iteración.

Tarea de Ingeniería	
No. de la Tarea: 1	**No. de la HU:** 1
Nombre de la tarea: Dibujar componentes de los DFD a través de eventos del mouse o ratón.	
Tipo de tarea: desarrollo	**Puntos estimados:** 2
Fecha inicio:22/03/2011	**Fecha fin:** 11/04/2011

Programador responsable: Tomás López Fernández.
Descripción: se implementan las funcionalidades necesarias para realizar el dibujo de los componentes que representan el inicio, fin, las asignaciones, bifurcaciones, repeticiones y salida de datos dentro de un DFD.

Tabla 16. Tarea de ingeniería de la HU 1.

Tarea de Ingeniería	
No. de la Tarea: 1	**No. de la HU:** 2
Nombre de la tarea: Validar que los DFD fueron diseñados correctamente.	
Tipo de tarea: desarrollo	**Puntos estimados:**1
Fecha inicio: 11/04/2011	**Fecha fin:** 18/04/2011
Programador responsable: Tomás López Fernández.	
Descripción: se realiza la implementación de las funcionalidades que permiten advertir si los DFD fueron diseñados correctamente, teniendo en cuenta que no existan componentes sin conectar o flujos (flechas) sobrantes.	

Tabla 17. Tarea de ingeniería de la HU 2.

3.6.2 Segunda iteración

En la segunda iteración se abordarán las HU 3 y 4, que se refieren a la edición de los contenidos de los componentes de los DFD y la generación de código en el lenguaje Java.

Historia de Usuario	**Tiempo de Implementación(semanas)**

	Estimación	Real
Editar contenido de los componentes de los DFD	1	1
Generar código en el lenguaje Java	1	2

Tabla 18. HU abordadas en la segunda iteración.

Tarea de Ingeniería	
No. de la Tarea: 1	**No. de la HU:** 3
Nombre de la tarea: Permitir edición de los contenidos de los componentes de los DFD.	
Tipo de tarea: desarrollo	**Puntos estimados:**1
Fecha inicio: 19/04/2011	**Fecha fin:** 26/04/2011
Programador responsable: Tomás López Fernández.	
Descripción: una vez diseñado un DFD es necesario permitir que los contenidos de cada uno de sus componentes se puedan editar sin necesidad de generar un nuevo diagrama o eliminar algún componente.	

Tabla 19. Tarea de ingeniería de la HU 3.

Tarea de Ingeniería	
No. de la Tarea: 1	**No. de la HU:** 4
Nombre de la tarea: Implementar funcionalidades necesarias para la generación del código en el lenguaje Java.	

Tipo de tarea: desarrollo	**Puntos estimados:**1
Fecha inicio: 27/04/2011	**Fecha fin:** 11/05/2011
Programador responsable: Tomás López Fernández.	
Descripción: tiene como objetivo obtener las funcionalidades necesarias para la generación del código del algoritmo diseñado en el lenguaje Java.	

Tabla 20. Tarea de ingeniería de la HU 4.

3.6.3 Tercera iteración

En esta iteración se abordan las HU 5 y 6, las que hacen referencia a la generación del código del algoritmo diseñado a través del DFD en los lenguajes C++ y C#, funcionalidades que completan las requeridas para la primera versión de la aplicación.

Historia de Usuario	**Tiempo de Implementación(semanas)**	
	Estimación	**Real**
Generar código en el lenguaje C++.	1	2
Generar código en el lenguaje C#	1	1

Tabla 21. HU abordadas en la tercera iteración.

Tarea de Ingeniería	
No. de la Tarea: 1	**No. de la HU:** 5
Nombre de la tarea: Implementar funcionalidades para generar el código en el lenguaje C++.	

Tipo de tarea: Desarrollo	**Puntos estimados:**1
Fecha inicio: 11/05/2011	**Fecha fin:** 25/05/2011
Programador responsable: Tomás López Fernández.	
Descripción: tiene como objetivo obtener las funcionalidades necesarias para la generación del código del algoritmo diseñado en el lenguaje C++.	

Tabla 22. Tarea de ingeniería de la HU 5.

Tarea de Ingeniería	
No. de la Tarea: 1	**No. de la HU:** 6
Nombre de la tarea: Implementar funcionalidades para la generar el código en el lenguaje C#.	
Tipo de tarea: desarrollo	**Puntos estimados:**1
Fecha inicio: 26/05/2011	**Fecha fin:** 3/06/2011
Programador responsable: Tomás López Fernández.	
Descripción: tiene como objetivo obtener las funcionalidades necesarias para la generación del código del algoritmo diseñado en el lenguaje C#.	

Tabla 23. Tarea de ingeniería de la HU 6.

3.7 Pruebas

La metodología XP entre las prácticas que propone se encuentra el uso de pruebas para comprobar la funcionalidad de los códigos que se van implementando. Esto permite medir la calidad de los productos finales reduciendo el número de errores no detectados y disminuye el tiempo entre la detección de un

error y su eliminación. También permite aumentar la seguridad evitando efectos colaterales no deseados a la hora de realizar modificaciones o refactorizaciones (31).

XP divide las pruebas en dos grupos, pruebas unitarias y pruebas de aceptación. Las pruebas unitarias son desarrolladas por los programadores y se encargan de verificar el código automáticamente y las pruebas de aceptación están destinadas a verificar que al finalizar cada iteración las historias de usuario cumplen con las funcionalidades asignadas y satisfagan las necesidades del cliente (31).

Las pruebas de sistema tienen como objetivo verificar las funcionalidades de los sistemas a través de sus interfaces externas comprobando que estas cumplan con las necesidades del cliente. Generalmente, estas pruebas son llevadas a cabo por los desarrolladores que verifican que su sistema se comporta de la manera deseada, por lo que podrían encajar en la definición de pruebas unitarias propuestas por la metodología XP. Las pruebas de sistema también tienen como objetivo verificar si se cumple con los requisitos establecidos por el cliente, por lo que también encajan en la definición de pruebas de aceptación definida por la mencionada metodología (31).

Se puede decir que las pruebas de aceptación son más importantes que las pruebas unitarias, ya que estas representan la satisfacción del cliente con el producto desarrollado y el final de una iteración y el comienzo de la siguiente.

3.7.1 Desarrollo dirigido por pruebas

El desarrollo dirigido por pruebas, (TDD) por sus siglas en inglés, es una práctica de programación definida por la metodología XP que involucra otras dos prácticas: escribir primero las pruebas (Test First Development) y refactorización (Refactoring). Para escribir estas pruebas se utiliza generalmente la prueba unitaria (PU).

Para la realización de estas pruebas el primer paso es escribir las pruebas y verificar que estas fallen, luego se implementa un código que haga que la prueba pase satisfactoriamente y luego se realiza la refactorización del código escrito. El propósito principal del desarrollo dirigido por pruebas es la obtención de un código limpio que funcione. La idea a tener presente es que los requisitos sean traducidos a pruebas, de esta forma, cuando estas pruebas pasen satisfactoriamente se estará garantizando que los requerimientos hayan sido implementados correctamente.

Este modelo de desarrollo dirigido por pruebas posibilita el uso de un diseño más robusto, tal es el caso, que a menudo se piensa como diseño dirigido por pruebas (Test Driven Desing). En consecuencia, el TDD facilita un diseño más fácil de mantener a través de la noción de pruebas, las que obliguen a reflexionar sobre el comportamiento del código y la forma de garantizar que funcione según lo previsto (31).

Reglas en las que puede resumirse la práctica del TDD

- No puede escribirse código productivo, a menos que sea para hacer pasar un test fallido.
- No se puede escribir más de lo necesario para hacer que falle un test unitario. Los errores de compilación son considerados fallos.
- No se puede escribir más código productivo que el estrictamente necesario para hacer pasar un test.

Ciclo de desarrollo del TDD

- **Escribir la prueba**. Para escribir la prueba, el desarrollador debe entender claramente las especificaciones y los requisitos. El diseño del documento deberá cubrir todos los escenarios de prueba y condición de excepciones.

- **Escribir el código haciendo que pase la prueba**. Este paso fuerza al programador a tomar la perspectiva de un cliente considerando el código a través de sus interfaces. Ésta es la parte conducida por el diseño del TDD. Como parte de la calibración de la prueba, el código debe fallar la prueba significativamente las primeras veces.

- **Ejecutar las pruebas automatizadas**. Si pasan, el programador puede garantizar que el código resuelve los casos de prueba escritos. Si hay fallos, el código no resolvió los casos de prueba.

- **Refactorización y limpieza en el código**. Después se vuelven a efectuar los casos de prueba y se observan los resultados.

- **Repetición**. Después se repetirá el ciclo y se comenzará a agregar las funcionalidades adicionales o a arreglar cualquier error.

3.7.2 Pruebas de aceptación

Las pruebas de aceptación son creadas en base a las historias de usuario en cada ciclo de iteración del desarrollo. El cliente debe realizar la especificación de uno o diversos escenarios para comprobar que una historia de usuario fue implementada correctamente.

Estas pruebas son consideradas pruebas de caja negra (Black Box System Test). Los clientes son los responsables de verificar que los resultados de estas pruebas sean los esperados. En caso de que fallen estas pruebas, los propios clientes son los encargados de indicar el orden de prioridad para la resolución de los errores. Las historias de usuario no se pueden considerar como terminadas hasta tanto no pasen todas las pruebas de aceptación. Debido a que la responsabilidad es compartida por todos los miembros del equipo de desarrollo, es recomendable que los resultados de las pruebas sean publicados para que todos estén al tanto de estas informaciones (31).

En las tablas de las Pruebas de Aceptación, se incluyen los siguientes campos: **Código caso de prueba**, que contiene el identificador de caso de prueba (en el caso de las presentes, se utiliza el identificador de la HU, al que se le adiciona '-P' y un número consecutivo); **Nombre historia de usuario**, que contiene el nombre de la HU correspondiente a este caso de prueba; **Descripción de la prueba**, que contiene una breve descripción de la prueba realizada; **Condiciones de ejecución**, se incluyen las condiciones necesarias para que se pueda realizar la prueba; **Entrada/pasos de ejecución**, contiene varios pasos enumerados para lograr realizar la prueba de esta HU; **Resultado esperado**, contiene la descripción de lo que se espera luego de realizar la prueba (cumplimiento de las restricciones del producto); y **Evaluación de la prueba**, muestra si la prueba fue satisfactoria o insatisfactoria.

Caso de prueba de aceptación	
Código: P1_HU1	**Historia de usuario:** 1
Nombre: Realizar dibujo de los componentes de los DFD.	
Descripción: revisar los componentes de los DFD sean dibujados correctamente, haciendo uso de los eventos del mouse o ratón.	

Condiciones de Ejecución: la aplicación se ejecuta correctamente en un navegador web.
Entrada/ pasos de ejecución: seleccionar uno a uno los componentes, realizar el diseño de los DFD: 1 – Realizar el diseño del componente Inicio. 2 – Realizar el diseño del componente Asignación. 3 – Realizar el diseño del componente Bifurcación. 4 – Realizar el diseño del componente Repetición. 5 – Realizar el diseño del componente Salida. 6 – Realizar el diseño del componente Fin. 7 – Realizar el diseño del componente Flujo.
Resultado Esperado: los componentes se dibujan correctamente.
Evaluación de la Prueba: satisfactoria.

Tabla 24. Prueba de aceptación para la HU 1.

Caso de prueba de aceptación	
Código: P1_HU2	**Historia de usuario:** 2
Nombre: Validar que los DFD sean diseñados correctamente.	
Descripción: evaluar que los DFD sean diseñados correctamente, que no existan componentes sin conectar a través de flujos (flechas) y que no falten los estados inicial y final.	
Condiciones de ejecución: haber diseñado u DFD para validar el diseño del mismo.	
Entrada / pasos de ejecución: se diseña un DFD y se trata de generar el código del correspondiente en cualquiera de los lenguajes Java, C++ o C#.	

Resultado esperado: si el DFD no está diseñado correctamente se lanza un mensaje advirtiendo que el DFD no es válido.
Evaluación de la prueba: satisfactoria.

Tabla 25. Caso de prueba para la HU 2.

Caso de prueba de aceptación	
Código: P1_HU3	**Historia de usuario:** 3
Nombre: Editar contenido de los componentes de los DFD diseñados.	
Descripción: una vez diseñados los DFD se edita el contenido de los componentes para realizar modificaciones al código correspondiente al algoritmo diseñado.	
Condiciones de ejecución: haber diseñado correctamente un DFD.	
Entrada/ pasos de ejecución: Una vez diseñado un DFD, se selecciona la opción Editar, se arrastra el mouse o ratón hasta el componente deseado.	
Resultado esperado: la edición de los componentes de los DFD se realiza correctamente.	
Evaluación de la prueba: satisfactoria.	

Tabla 26. Caso de prueba para la HU 3.

Caso de prueba de aceptación	
Código: P1_HU4	**Historia de usuario:** 4
Nombre: Realizar la generación de código en el lenguaje Java.	

Descripción: generar el código correspondiente a un DFD en el lenguaje Java.
Condiciones de ejecución: realizar el correcto diseño de un DFD.
Entrada/ pasos de ejecución: una vez terminado el diseño de un DFD, se selecciona la opción para generar el código en el lenguaje Java.
Resultado esperado: el código es generado y mostrado correctamente.
Evaluación de la prueba: satisfactoria.

Tabla 27. Caso de prueba para la HU 4.

Caso de prueba de aceptación	
Código: P1_HU5	**Historia de usuario:** 5
Nombre: Realizar la generación de código en el lenguaje C++.	
Descripción: generar el código correspondiente a un DFD en el lenguaje C++.	
Condiciones de ejecución: realizar el correcto diseño de un DFD.	
Entrada/ pasos de ejecución: una vez terminado el diseño de un DFD, se selecciona la opción para generar el código en el lenguaje C++.	
Resultado esperado: el código es generado y mostrado correctamente.	
Evaluación de la prueba: satisfactoria.	

Tabla 28. Caso de prueba para la HU 5.

Caso de prueba de aceptación

Código: P1_HU6	**Historia de usuario:** 6
Nombre: Realizar la generación de código en el lenguaje C#.	
Descripción: generar el código correspondiente a un DFD en el lenguaje C#.	
Condiciones de ejecución: realizar el correcto diseño de un DFD.	
Entrada/ pasos de ejecución: una vez terminado el diseño de un DFD, se selecciona la opción para generar el código en el lenguaje C#.	
Resultado esperado: el código es generado y mostrado correctamente.	
Evaluación de la prueba: satisfactoria.	

Tabla 29. Caso de prueba para la HU 6.

Para un desarrollo satisfactorio del proceso de pruebas se definieron 6 casos de pruebas, se realizaron un total de 3 iteraciones utilizando el método de caja negra para comprobar las funcionalidades con las que debe cumplir la herramienta. Fueron detectadas 16 no conformidades, 7 críticas y 9 no críticas. Estas no conformidades tenían un impacto negativo sobre el funcionamiento de la herramienta, por lo que su erradicación contribuyó a que se pudieran mejorar y completar las funcionalidades que brinda la aplicación.

Conclusiones generales

La investigación desarrollada y los resultados obtenidos le permiten al autor de la presente investigación arribar a las siguientes conclusiones:

- La búsqueda y análisis de sistemas semejantes demostró que aunque existen en el mundo herramientas que permiten el diseño de los DFD, la mayoría son aplicaciones de escritorio lo que limita su uso sobre plataformas web desaprovechando las ventajas tecnológicas que brinda la UCI.
- El estudio de las características y aplicaciones de los DFD, posibilitó el diseño de una herramienta web que permite el diseño de estos diagramas y la generación de código de los algoritmos correspondientes en los lenguajes Java, C++ y C# que sirve de apoyo a los procesos de aprendizaje de temas relacionados con el diseño de algoritmos que se imparten en la asignatura Introducción a la Programación perteneciente a la disciplina Programación.
- El uso de la metodología ágil de desarrollo de software XP logró una buena efectividad en la ingeniería de software y la gestión del proyecto durante todo su desarrollo.
- Se obtuvo una aplicación web funcional (probada durante el proceso de desarrollo y validada a través de pruebas de aceptación) desarrollada con tecnologías libres basada en la tecnología Applet de Java, provista de una interfaz sencilla y fácil de utilizar.

Recomendaciones

Gracias a los resultados obtenidos y a la experiencia adquirida durante la realización de este trabajo de diploma, con el propósito de asegurar la ampliación, modificación y mejora de la solución propuesta, se expresan las siguientes recomendaciones:

- Continuar el proceso de desarrollo de la aplicación con la implementación de nuevas funcionalidades que posibiliten: realizar la ejecución paso a paso del algoritmo, comprobar la validez del DFD a través de la obtención del resultado de la ejecución del algoritmo diseñado.
- Realizar la firma digital del applet para la incorporación de funcionalidades que permitan la carga y descarga de ficheros en las computadoras clientes.

Referencias bibliográficas

1. **Vitturini, María Mercedez.** *Análisis y Diseño de Sistemas.* [PDF] Buenos Aires : s.n, 2010.

2. **Córdova Neri, Teodoro.** *Diagramas de Flujos de Datos.* [PDF] Lima, Perú : s.n, 2005.

3. **SGV Sarc Inc.** Sitio de descarga de software. [En línea] 15 de febrero de 2007. http://www.freedownloadmanager.org/es/downloads/FLUJO_de_cristal_para_C_23497_p/.

4. **SGV Sarc Inc**. Sitio de descarga de software. [En línea] 31 de enero de 2008. http://www.freedownloadmanager.org/es/downloads/Revs_Crystal_para_C%2B%2B_52717_p/.

5. **Origramy Inc.** Sitio de descarga de software. [En línea] 18 de febrero de 2009. http://www.freedownloadmanager.org/es/downloads/Origramy_55992_p/.

6. **López Berrios, C.** ETSIT-UPM. [En línea] 25 de noviembre de 2005. http://www-lsi.die.upm.es/~carreras/ISSE/programacion_extrema_1.x2.pdf. [Accedido Abril, 2011].

7. **Eumednet.** Grupo de investigación eumednet (SEJ-309) de la Universidad de Málaga. [En línea] agosto de 2009. http://www.eumed.net.

8. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El proceso unificado de desarrollo de software.* [PDF] Madrid : Addison Wesley, Addison Wesley, 2000.

9. **Canós, José H, Letelier, Patricio y Penadés, María Carmen.** *Métodologías Ágiles en el Desarrollo de Software.* [PDF] Valencia : s.n, s.a.

10. **Álvarez, Itzcoalt.** SG Software Guru. [En línea] s.a. http://www.sg.com.mx/sg07/presentaciones/Mejora%20de%20procesos/SG07.P02.Scrum.pdf.

11. **Calderón, Anyelín.** Sistematizando Aprendizajes. [En línea] abril de 2010. http://anyelincalderon.blogspot.com/2010/04/algo-sobre-metodologias-agiles.html.

12. **Álvarez, Boris Luis y Rojas Güemes, Marlon.** *Software Educativo como soporte tecnológico del aprendizaje técnico-táctico del Futsal para los estudiantes de la Universidad de las Ciencias Informáticas.* [PDF] La Habana : s.n, 2009.

13. **Soluciones Racionales.** Soluciones Racionales. [En línea] s.a. http://www.solucionesracionales.com/xde_dev.html. [Accedido Abril, 2011].

14. **Visual Paradigm.** Visual Paradigm. [En línea] 2004. http://www.visual-paradigm.com/product/vpuml/.

15. **Garrido Vargas, Yoennis.** *Arquitectura para la creación de aplicaciones multimedia. MAPri.* [PDF] La Habana : s.n., 2009.

16. **López, Ángel.** *Java, la programación del futuro.* [PDF] Buenos Aires : s.n, 2005.

17. **ULPGC.** *Introducción a Javascript.* [En línea] s.a. http://www.ulpgc.es/otros/tutoriales/JavaScript/cap1.htm. [Accedido Abril, 2011].

18. **Heredia Santos, Herminio.** Maestros del web. [En línea] s.a. http://www.maestrosdelweb.com/editorial/phpintro/.

19. **Cattell, Rick.** *J2EE in practice.* [PDF] 2005.

20. **Prado Ajona, Guillermo.** Página Web sobre Html y CSS. [En línea] Universidad de la Rioja, s.a.

21. **Stroustrup, Bjarne.** *El lenguaje de programación C++.* [PDF] Madrid : Addison Wesley, 2000.

22. **González Seco, José Antonio.** *El lenguaje de programación C#.* [Ebook] 2002.

23. **The Apache Software Foundation.** Apache HTTP Server Project. *Information on the Apache HTTP Server Project.* [En línea] 2011. http://httpd.apache.org/dev/.

24. **Netbeans.** Netbeans. [En línea] 2011. http://netbeans.org/.

25. **The Eclipse Foundation.** Eclipse. [En línea] 2011. http://www.eclipse.org/.

26. **Joskowicz, José.** *Reglas y prácticas en eXtreme Programming.* [PDF] Uruguay: 2008.

27. **Carballo, Carlos David.** *La programación y la arquitectura de software.* [PDF] España : s.n, 2008.

28. **Erizaca Ramírez, Elisa.** *Análisis y diseño de sistemas.* [PDF] La Paz, Bolivia : s.n, 2009.

29. **EcuRed.** *Patrones de diseño y arquitectura*. [En línea] s.a. http://www.ecured.cu/index.php/Patrones_de_dise%C3%B1o_y_arquitectura. [Accedido Abril, 2011].

30. **González Cornejo, José Enrique.** *Acerca del estilo en programación*. [En línea] 18 de abril de 2009. http://www.docirs.cl/acerca_del_estilo_programacion.htm. [Accedido Abril, 2011].

31. **Gutiérrez, J. J.** *Pruebas del sistema en programación extrema.* [PDF] Sevilla : s.n, 2004.

Glosario de términos

Software: equipamiento lógico o soporte lógico de un ordenador. Comprende el conjunto de los componentes lógicos necesarios para hacer posible la realización de una tarea específica, en contraposición a los componentes físicos del sistema (hardware). Suele ser utilizado para referirse a los programas y aplicaciones informáticas.

Tecnologías de la Información y las Comunicaciones: conjunto de servicios, redes, software y dispositivos que tienen como fin la mejora de la calidad de vida de las personas dentro de un entorno, y que se integran a un sistema de información interconectado y complementario.

Metodología: métodos de investigación que se siguen para alcanzar una gama de objetivos en una ciencia.

Metodología de Desarrollo: marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información.

Lenguaje de Programación: conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

Interfaz de Programación de Aplicaciones (API): conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

dirección IP: es una etiqueta numérica que identifica, de manera lógica y jerárquica, a una interfaz de un dispositivo (habitualmente una computadora) dentro de una red que utilice el protocolo IP (Internet Protocol), que corresponde al nivel de red del protocolo TCP/IP.

GUI: acrónimo de Graphical User Interface (en español: Interfaz Gráfica de Usuario), es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso consiste en proporcionar

un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

Multiplataforma: es un término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de *software*, que puedan funcionar en diversas plataformas. Por ejemplo, una aplicación multiplataforma podría ejecutarse en los sistemas operativos Windows, en GNU/Linux y en Mac OS.

Orientado a objetos: hace referencia a la Programación Orientada a Objetos, un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas de ordenador. Está basado en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento.

Redirect: directiva del servidor HTTP Apache, que redirecciona una URL vieja a una nueva.

Planeamiento Blizt: establece que se escriben las funciones del programa en tarjetas y los tiempos para cada una son estimados por los programadores de forma independiente entre las mismas.

Estimación Delphi: empleado para estimaciones de pericia. Los expertos se reúnen y definen el tamaño del proyecto, fecha de entrega, etc.