Universidad de las Ciencias Informáticas

Facultad 2

Título: Asistente Matemático para Teoría de Grafos en la asignatura Matemática Discreta.

Trabajo de Diploma para optar por el título de

Ingeniero Informático

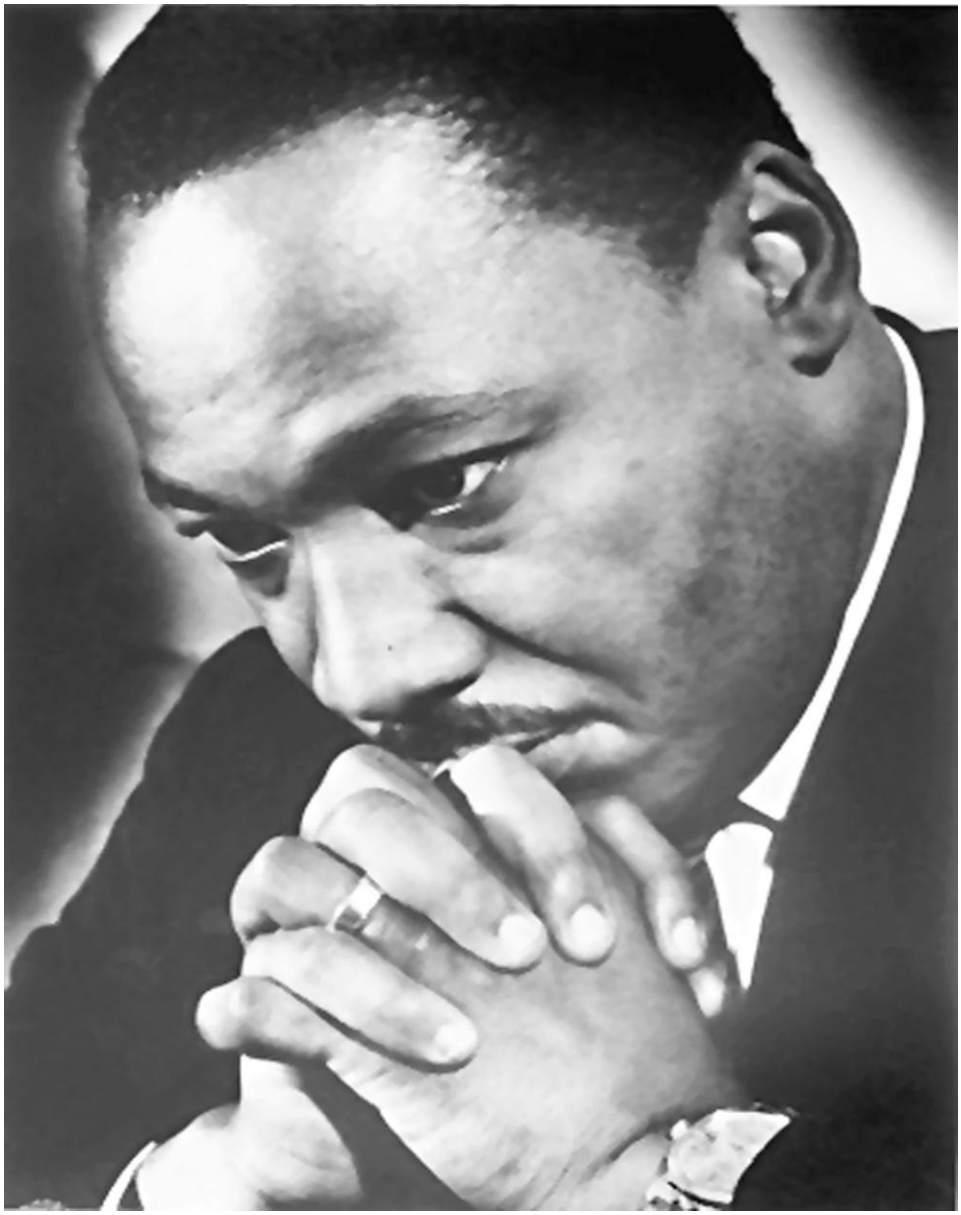
Autores: Dagoberto Pérez Hernández

Daniela Terrero Aroche

Tutor: Angélica María Díaz Valdivia

Co-tutor: Edgar González Blanco

La Habana, 2013



"Los filósofos no han hecho más que interpretar el mundo, lo que hace falta es transformarlo"

Martin Luther King

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 2 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los 19 días del mes de junio del año 2013

Daniela Terrero Aroche

_____

Dagoberto Pérez Hernández

_____

Dedicatoria

Daniela:

A la memoria de mi madre.

A mis abuelos queridos.

A mi papá.

A mi familia.

A mamita Margot y papá Guarín.

A Adrián.

A mis maestros.

A mis amigos.

A mis compañeras de cuarto.

A mis conocidos.

Dagoberto:

A muy especial a mi mamá Ángela por faro y guía, por saber encaminarme por el camino bueno y ser un gran ejemplo a seguir, por ser la maravillosa mujer y que es y la mejor madre del mundo.

A mis abuelos, a mi hermanito Dayan y muy especial para mi tío Pablo que sé que dios lo está cuidando muy bien, a mi papa Dagoberto y a toda mi familia.

Agradecimientos

Daniela:

Quiero agradecer a mis padres por darme la vida.

A mis abuelos por ser más que padres y amigos.

A mi familia por estar siempre ahí.

A mis amigos por darme apoyo.

A Adrián por darme amor cuando realmente lo necesito.

A todos los que han contribuido con mi formación.

A mi compañero de tesis, por aguantar mis malacrianzas.

A los tutores por ser la guía para alcanzar resultados.

A las personas buenas que he conocido.

Y a todos los que me han enseñado a cumplir mis sueños.

Dagoberto:

A mi mamá Ángela por ser el mayor regalo que me ha dado la vida, por ser guía y ejemplo, por su amor, cariño y confianza, por hacer de mí, con sus enseñanzas y consejos, todo lo que soy hoy, por apoyarme en todas las etapas de mi vida y en los momentos malos y buenos.

A mi hermano Dayan por estar siempre a mi lado, por apoyarme, aguantarme y compartir conmigo los mejores momentos de mi vida, por ser el mejor hermano del mundo.

A mis abuelos Pucha, Guango Lucila y Roberto por si apoyarme y aconsejarme, por estar siempre presentes en mi vida. A mi papa por ser un gran apoyo.

A mis tíos y primos por confiar siempre en mí, especialmente a mí tía Barby por su cariño y amor.

A toda mi familia por su constante confianza y preocupación, por ser mi primera escuela.

Agradecimientos

A Orlando por ser como un hermano para y siempre estar ahí dándome su apoyo incondicional.

A mi gran amigo Yunior que siempre me ha acompañado desde la vocacional.

A mis compañeros de apto por ser los mejores de la UCI y soportarme durante 3 años, Norey, Arian, Reinier y Olivio.

A todos mis compañeros de grupo durante los 5 años de Universidad, los que están y los que por algún motivo tuvieron que abandonar la escuela, especialmente al Bara, Yordo, Dayna, Aliannys, Dany Lili, Henry, Danima y Delmis.

A todos mis amistades especialmente a Yankiel, Bauza, Francis, Carlos, Adrián, Gabo, Chamizo, Souley y Iván Eddy .A todo el piquete de la recre y las fiestas.

A mis amigos de la zona y todas mis amistades de la vocacional, Manuel, Alberto, Miguel Ángel, Dayana y Malena, por demostrarme que las distancia nada puede contra una amistad verdadera.

Al grupo de danza "Rocanroleando" que este curso vivimos muchos momentos emotivos y felices que perduraran para siempre.

A todos mis compañeros del laboratorio 302 por soportarme durante todo 5to año.

A mi compañera de tesis Daniela por su empeño y dedicación. A mis compañeros del equipo del asistente matemático, Niorge, Ronny, Leanet y Osorio.

A mis tutores Angélica y Edgar por sus enseñanzas y apoyo durante el desarrollo de la tesis.

A todos: ¡GRACIAS!

RESUMEN

El avance tecnológico del mundo moderno ha permitido crear herramientas informáticas que permiten el apoyo al trabajo con elementos docentes, que permiten reformar el proceso de enseñanza – aprendizaje, y que se especializan en sintetizar temas de las matemáticas como *WinQSB* para Investigación de Operaciones, *SPSS* y *Statistic* para Estadística, Biometría y Econometría*, Interactive Physics* para simular fenómenos físicos*; y Analítica* y *Atom Visualizer* para Química*.*

La teoría de grafos es una temática perteneciente al campo de la Matemática Discreta que es necesaria para la formación de profesionales en las universidades. Particularmente cuenta en sí con herramientas que facilitan la comprensión y estudio de algoritmos dictados por la rama. Los mismos no abarcan en su totalidad todos los temas comprendidos de la teoría de grafos.

Este trabajo propone una herramienta de apoyo para el trabajo con la teoría de grafos, que cubra la mayoría de las temáticas no tratadas acerca de la teoría y que sirva de material de enseñanza-aprendizaje de la asignatura Matemática Discreta y de los estudiantes del primer año de la carrera de Ingeniería en Ciencias Informáticas de la UCI. Esta herramienta está desarrollada en el lenguaje de programación *Java* con *NetBeans IDE*, guiada por la metodología de desarrollo de *software XP*, utilizando *UML* para modelado y *Visual Paradigm* como herramienta *CASE.*

PALABRAS CLAVE

Teoría de Grafos, Objetos Discretos, Asistente Matemático, Matemática Discreta.

ÍNDICE

Índice

INTRODUCCIÓN

Los avances tecnológicos del mundo moderno han sido significativos convergiendo con sus aplicaciones en la mayoría de las áreas en la que se desenvuelve la humanidad. Los procesos docentes y educativos, particularmente en las matemáticas, no han sido ajenos al cambio aparejado a la introducción de la tecnología.

Los entornos computarizados de aprendizaje de las matemáticas caracterizados por el uso de herramientas matemáticas, Internet, el trabajo colaborativo, unido a otros medios audiovisuales propician que el profesor se concentre en su nuevo papel de estimulador y facilitador del aprendizaje y permite al estudiante explorar, inferir, hacer conjeturas, justificar, poner a prueba argumentos y de esta forma construir su propio conocimiento [1].

Basado en el poder de las nuevas tecnologías de la información como parte de propuestas metodológicas para la enseñanza–aprendizaje, se utilizan herramientas en diferentes asignaturas como: el *WinQSB* utilizado para la Investigación de Operaciones; *SPSS* y *Statistic* en la Estadística, Biometría y Econometría; para el apoyo de los temas de la Física existen diferentes programas como *Interactive Physics*, *Software Data Studio* y *Software VideoPoint Physics Fundamentals*. En Química *se* emplean *Química Analítica* y *Atom Visualizer*.

En los diferentes años de la carrera que se cursa en la Universidad de las Ciencias Informáticas adecuándose al objetivo de formar Ingenieros en Ciencias Informáticas, el plan de estudios incluye asignaturas de Ciencias Básicas como la Matemática Discreta. Como parte de los temas de la asignatura se incluye la Teoría de Grafos que es uno de los más extensos y que mayores problemas ocasionan a los estudiantes a la hora de su estudio, lo que dificulta el entendimiento y razonamiento lógico que conlleva este contenido, así como a la hora de la realización de ejercicios y pruebas de este tema y que es utilizado e impartido como estructura de datos en la Programación II.

A pesar de que la tecnología es utilizada como medio de enseñanza y aprendizaje en la mayoría de las asignaturas, para la instrucción de la Matemática Discreta los profesores y estudiantes no pueden auxiliarse de una herramienta informática, que logre una integración entre las conferencias, clases prácticas, trabajos independientes, laboratorios y seminarios incluyendo las evaluaciones, que permita un

resultado superior en el cumplimiento de los objetivos y una mayor motivación de los estudiantes por la asignatura.

Lo que conlleva a la siguiente **situación problemática**: En la asignatura Matemática Discreta no se cuenta con una herramienta matemática que presente funcionalidades capaces de trabajar exclusivamente con grafos, que se integre con los objetivos de la asignatura para lograr un mejor cumplimiento de los mismos, favoreciendo también la motivación del estudiante y que su utilización sea sencilla para el aprendizaje de los alumnos que provienen de la Enseñanza Media Superior y se enfrentan a un nuevo sistema de enseñanza más independiente y superior.

Teniendo en cuenta lo planteado anteriormente, se deriva el siguiente **problema a resolver**: ¿Cómo mejorar el trabajo con la teoría de grafos en la asignatura Matemática Discreta en el primer año de la carrera?

Siendo identificado como **objeto de estudio** el trabajo con la Teoría de Grafos, estableciéndose como **campo de acción**: el Trabajo de la Teoría de Grafos en la asignatura Matemática Discreta.

Para darle solución al problema se formula el **objetivo general** de este trabajo: Desarrollar una herramienta informática, que asista al trabajo con Teoría de Grafos, con un enfoque de apoyo al proceso de enseñanza-aprendizaje en la asignatura Matemática Discreta en la Universidad de las Ciencias Informáticas.

Para darle cumplimiento al Objetivo General se plantean las **Tareas de la Investigación** siguientes:

- Estudio sobre la teoría de grafos de las asignaturas Matemática Discreta I y II.
- Análisis de las herramientas informáticas dedicadas al trabajo con la teoría de grafos.
- Definición de las herramientas, lenguajes y metodología de desarrollo de *software* a utilizar, para dar soporte a los procesos de modelación, implementación y pruebas.
- Documentación de requerimientos funcionales y no funcionales que debe cumplir la herramienta de apoyo al trabajo con teoría de grafos.

- Desarrollo de la arquitectura de *software*[1] de la herramienta de apoyo al trabajo con teoría de grafos.
- Implementación de las funcionalidades para dar cumplimiento a los requerimientos que debe cumplir la herramienta de apoyo al trabajo con teoría de grafos.
- Elaboración de pruebas de funcionamiento a la herramienta para evaluar el cumplimiento de los requerimientos identificados.
- Ejecución de las pruebas a la herramienta para evaluar el cumplimiento de los requerimientos identificados.

Métodos de la Investigación:

Métodos Teóricos:

- Analítico – Sintético: Se examinarán bibliografías correspondientes a los contenidos de Matemática Discreta I y Matemática Discreta II, con el objetivo de utilizar la información para la realización de una herramienta para el apoyo en la enseñanza de la teoría de grafos.
- Histórico-lógico: En la primera parte de la investigación se desarrollará un estudio del estado del arte de la problemática; así como se analizarán los antecedentes y conceptos de la Teoría de Grafos.

Métodos Empíricos:

- Observación: Se utiliza para constatar las deficiencias existentes en herramientas matemáticos utilizados actualmente en la UCI, así como adquirir conocimientos sobre el funcionamiento, ventajas y desventajas de otras aplicaciones educativas similares a la que se desarrollar.

[1] *Software* (en inglés): término para definir programa informático.

El documento estará estructurado de la siguiente manera:

Capítulo 1 "Fundamentación teórica": En este capítulo se realiza una descripción de los conceptos fundamentales asociados al dominio del problema. Además se ofrece una panorámica de las principales características de las herramientas informáticas que trabajen la Teoría de grafos. Se presenta una descripción, análisis y selección de las herramientas, lenguajes y metodología para el desarrollo de la aplicación.

Capítulo 2 "Características del sistema": Se realiza una detallada descripción de los requerimientos funcionales y no funcionales del sistema, el modelo de dominio en el que se enmarca la situación problemática, obteniéndose las Historias de usuario y el Plan de Entregas junto a la planificación de las iteraciones.

Capítulo 3 "Diseño del Sistema": Se describe la arquitectura que soportará al sistema, estilo, patrones y diseño arquitectónico a desarrollar.

Capítulo 4 "Implementación y Prueba": En este capítulo estará dedicado a la implementación del sistema partiendo de los elementos del diseño. Se obtienen los resultados de las pruebas unitarias y de aceptación por parte del cliente para medir la calidad y cumplimiento de los requerimientos funcionales.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

INTRODUCCIÓN

En el presente capítulo se ofrece una perspectiva acerca de los conceptos asociados con el problema y los resultados del análisis a las herramientas de apoyo a la enseñanza de las Matemáticas y al trabajo de la Teoría de Grafos existentes tanto en el mundo como en Cuba, y que son utilizadas en la docencia actualmente, así como un breve análisis de las herramientas, lenguajes y tecnologías seleccionadas para el desarrollo de la herramienta.

1.1 SÍNTESIS DE TEORÍA DE GRAFOS O GRÁFICAS

Aunque el primer artículo en la teoría de grafos apareció en 1736 y varios resultados se obtuvieron en el siglo XIX, solo fue a partir de la década de 1920 que hubo un interés sostenido, amplio e intenso. Sin duda, una de la razón de este reciente interés es su capacidad de aplicación en campos muy diversos, incluyendo las ciencias de la computación, la química, la investigación de operaciones, la ingeniería eléctrica, la lingüística y la economía [2].

1.1.1 HISTORIA

En 1736, el matemático suizo afincado en San Petersburgo Leonhard Euler publicó "*Solutio problematis ad geometriam situs pertinentis*[2]", un artículo en el que resolvía el problema de los siete puentes. La ciudad de *Königsberg* (llamada *Kaliningrado*, durante la época soviética) estaba dividida por el río *Pregel* en cuatro zonas. En aquel entonces había siete puentes comunicando las distintas zonas. Los ciudadanos de *Königsberg* buscaban un recorrido que travesase cada puente exactamente una vez. Fue Euler quien demostró que la existencia de un tal recorrido requeriría, en general, que a lo sumo dos de las zonas estuvieran unidas con el resto mediante un número impar de puentes, cada una de ellas. Este trabajo es considerado como el nacimiento de la Teoría de Grafos, utilizada hoy en día en un sinfín de aplicaciones,

[2] *Solutio problematis ad geometriam situs pertinentis* (en latín): Solución de un problema relacionado con la geometría de posición

y también uno de las primeras apariciones de una “nueva geometría” en la que importan sólo las propiedades estructurales de un objeto y no sus medidas[3] [3].

En 1852 Francis Guthrie planteó el problema de los cuatro colores, donde se cuestiona si es posible, utilizando solamente cuatro colores, dibujar cualquier mapa de países de tal forma que dos países vecinos nunca tengan el mismo color. Este problema, no fue resuelto hasta un siglo después por Kenneth Appel y Wolfgang Haken, los matemáticos intentando solucionarlo definieron términos y conceptos teóricos fundamentales de los grafos [4].

En 1857 Arthur Cayley estudió el problema de la enumeración de los isómeros de los hidrocarburos saturados C_nH_{2n+2} fijado el número n de átomos de carbono, es decir se trataba de determinar el número de compuestos químicos con idéntica composición (fórmula) pero distinta estructura molecular (disposición distinta de los enlaces). Para ello representó cada hidrocarburo mediante un árbol donde los vértices representaban los átomos (de grado uno los de hidrógeno y cuatro los de carbono) y donde las aristas indicaban la existencia de enlaces químicos. De este modo su problema equivalía a la enumeración (excepto isomorfismos) de los árboles con grados de sus vértices igual a 1 o 4. Antes de resolver su problema, consiguió hallar el número de árboles con raíz y el número de árboles con grado máximo ≤ 4 y finalmente cerró su problema inicial [3].

1.1.2 DEFINICIONES FUNDAMENTALES

Primeramente un **grafo** G puede definirse como un conjunto no vacío V (de vértices) y un conjunto A (de aristas) extraído de la colección de subconjuntos de dos elementos de V. Una arista de G es, pues, un subconjunto, con $a, b \in V$, $a \neq b$ [5] .

De manera simple puede decirse que los grafos son estructuras discretas que constan de vértices y aristas que conectan estos vértices [5]. Gráficamente pueden representarse los vértices como figuras geométricas, cuadrados o círculos y a las aristas como líneas o flechas.

[3] A esto se refieren las palabras “geometriam situs” en el título del artículo de Euler.

Los **vértices o nodos**, constituyen uno de los dos elementos que forman un grafo. Se define también como la unidad fundamental de la que están compuestos los grafos.

Las **aristas**, junto con los vértices, forman los elementos principales de un grafo. Se definen como las uniones entre nodos o vértices. Usualmente las aristas denotan relaciones entre los vértices, como el orden, la vecindad o la herencia. Estas aristas pueden tener o no dirección lo que indicaría si el grafo pertenece a la clasificación de dirigidos o no dirigidos.

El grafo **dirigido** o **dígrafo** es un tipo de grafo en el cual el conjunto de las aristas tiene una dirección definida**.** Las aristas dirigidas se indican mediante flecha o saeta (ver Fig.1). De manera contraria los grafos no dirigidos se indican con líneas sin fechas (ver Fig. 2).

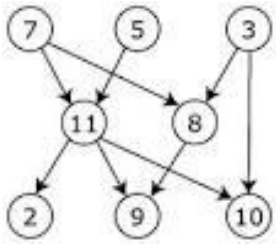


Fig. 1 Grafo dirigido o Dígrafo.

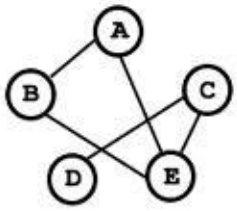


Fig. 2 Grafo no dirigido.

Una de las definiciones fundamentales que presenta la teoría de grafos consiste en los caminos o secuencias. Se define **camino** como una secuencia de vértices adyacentes. Es decir, cada par de vértices de la secuencia son adyacentes. Sean v_0 y v_n vértices de un grafo, un camino o ruta de v_0 a v_n de longitud n es una sucesión alternante de n+1 vértices y n aristas que comienza en v_0 y termina en v_n [5].

Se conoce como **ciclo** a la sucesión de aristas adyacentes, donde no se recorre dos veces la misma arista, y donde se regresa al punto inicial. Los grafos que poseen ciclos son grafos **cíclicos** por el contrario los que no presentan este tipo de secuencia se conocen como grafos **acíclicos**.

Los grafos pueden clasificarse dependiendo de diferentes condiciones, ya se la cantidad de aristas incidentes en un vértice, las particularidades de caminos, unión entre vértices, etc. A continuación se presentan algunos de los grafos más comunes y utilizados de la teoría de grafos.

Un grafo **simple** es un grafo que presenta aristas no dirigidas y no puede tener aristas múltiples ni lazos. Formalmente un grafo representado como *G* = (*V; E*) es simple si está formado por un conjunto finito de vértices *V* y un conjunto *E* de pares no ordenados de vértices distintos. A los elementos de *E* se les denomina aristas (no dirigidas o no orientadas). También puede decirse que en un grafo simple a lo sumo solo una arista une dos vértices cualesquiera [5].

Se conoce como grafo **conexo** (ver Fig.3) a cualquier grafo que entre cualquier par de sus vértices existe un camino que los une [5].

Un grafo **completo** (ver Fig. 4) se denota k_n donde n es la cantidad de vértices del grafo G que es un grafo simple con n vértices en la cual existe una arista para cada par de vértices distintos [5].

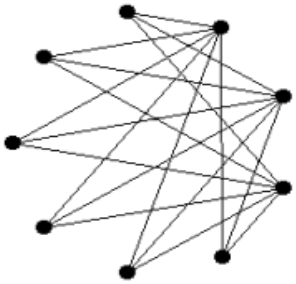


Fig. 3 Grafo Conexo.

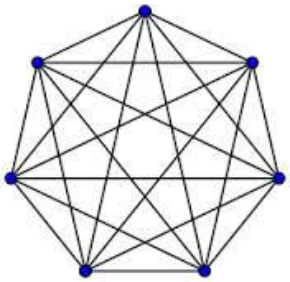


Fig. 4 Grafo Completo.

En la Fig. 5 se representa un grafo **ponderado** el cual tiene etiquetas sobre las aristas, se dice que el peso de la arista propiamente dicha es k y la longitud de los caminos en el grafo es la suma de los pesos que conforman a cada camino [5].

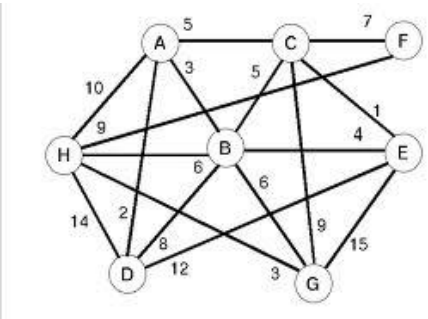


Fig. 5 Grafo ponderado.

Se dice que un grafo simple *G* = (*V;E*) es **bipartido** si su conjunto de vértices *V* se puede expresar como la unión de dos subconjuntos no vacíos disjuntos *u* y *v* de manera que cada arista del grafo conecta un vértice de *u* con un vértice de *v* [5]. En otras palabras, no existe ninguna arista entre dos vértices de *u* ni entre dos vértices de *v* (ver Fig. 6).

Un grafo **completo bipartito** (ver Fig. 7) se denota $k_{m,n}$ donde *m* y *n* son la cantidad de aristas pertenecientes a un grafo simple G, cuyo conjunto de vértices V puede dividirse en los subconjuntos V_1 con *m* vértices y V_2 con *n* vértices, en los cuales existe una arista entre cada par de vértices V_1 y V_2, donde v_1 está en V_1 y v_2 está en V_2 [5].

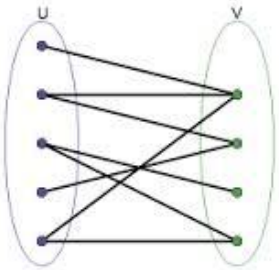


Fig. 6 Grafo bipartito.

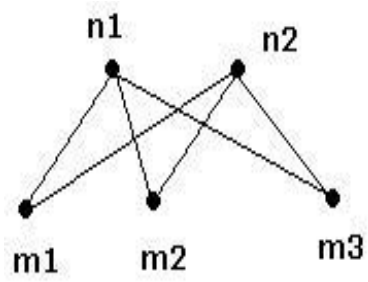


Fig. 7 Grafo Bipartito Completo.

Un grafo G es **nulo** cuando el conjunto E de aristas es nulo o vacío (ver Fig. 8).

El grafo de **Euler (**ver Fig. 9) es un grafo que contiene un ciclo de Euler, es decir que exista un camino que recorra todos los vértices transitando por las aristas una sola vez. *Corolario:* Si un grafo contiene un ciclo euleriano entonces es a su vez un grafo conexo y cada vértice es de grado par [5].

Un grafo de **Hamilton** (ver Fig. 10) es un grafo que contiene un ciclo de Hamilton lo que supone que existe un camino que contiene cada vértice exactamente una vez excepto el vértice inicial y final que aparece dos veces [5].

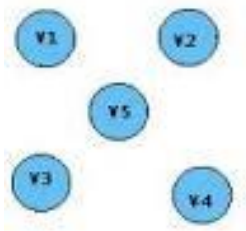


Fig. 8 Grafo Nulo.

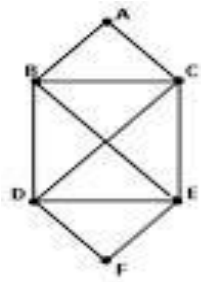


Fig. 9 Grafo de Euler.

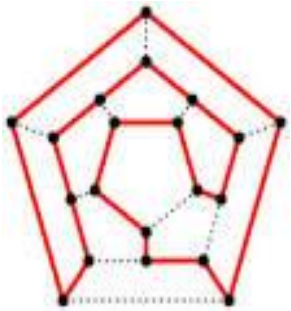


Fig. 10 Grafo de Hamilton.

Puede definirse como **Árbol** a un grafo simple T (ver Fig. 11) que satisface: Si v y w son vértices en T entonces existe un único camino simple de v a w, quiere decir cualesquiera dos vértices están conectados por exactamente un camino. *Corolario*: Todo árbol es un grafo no cíclico. El grafo **Bosque** es una unión disjunta de árboles (ver Fig. 12), de forma equivalente un bosque es un grafo acíclico [5].

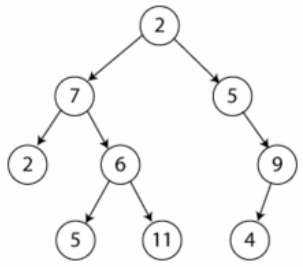


Fig. 11 Árbol.

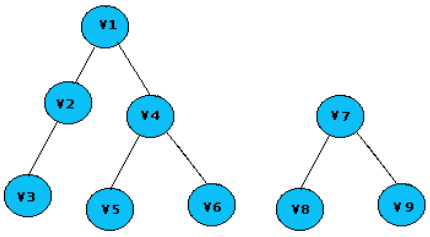


Fig. 12 Bosque.

1.2 TENDENCIAS DE LOS HERRAMIENTAS MATEMÁTICAS

Esta sección está dedicada al análisis de las herramientas matemáticas que se utilizan como parte del plan de estudio de la Universidad, junto a otras que se destacan por su uso en el trabajo con la teoría de grafos, con el objetivo de caracterizarlos para identificar las deficiencias o ventajas en su aplicación dentro del plan de estudios de la universidad de la asignatura Matemática Discreta.

1.2.1 MATLAB[4]

MATLAB es un *software* matemático con entorno de desarrollo integrado (en lo adelante IDE, por sus siglas en inglés) que tiene un lenguaje de programación propio (Lenguaje M) y es multiplataforma (Unix, Windows y Apple MacOs). Posee un lenguaje de alto nivel y ambiente interactivo para la computación numérica, visualización y programación. Se pueden analizar datos, desarrollar algoritmos y crear modelos y aplicaciones.[6].

MATLAB trabaja profunda y profesionalmente la mayoría de los temas relacionados con grafos y las operaciones pueden realizarse con sus representaciones matriciales [7]. Esto tiene como ventaja que el análisis para grafos sea preciso. Los algoritmos aplicados en teoría de grafos en su mayoría descomponen el grafo desde su matriz, lo cual brinda precisión en los cálculos y rapidez en la ejecución, debido a que recorrer una matriz es sencillo de lograr utilizando ciclos anidados.

Sin embargo la desventaja más impactante es que, al solamente enfocarse en la representación matricial quedan eximidas otras formas de representación, que son igual de importantes para el cumplimiento de objetivos de la asignatura, tal como, dominar todas las representaciones de un grafo, como son la representación en forma de conjuntos de vértices y aristas; y la representación gráfica.

Otro inconveniente es la complejidad del lenguaje M de Matlab. Si se tratara de un conocimiento profesional que se requiere para dominar la herramienta, fácilmente puede resolverse con una preparación o curso. En este caso se trata de brindar un conocimiento de la forma que motive más al

[4] MATrix LABoratory, “Laboratorio de Matrices”.

estudiante, en la que el profesor pueda explicar con un mayor alcance para lograr el entendimiento en los alumnos.

Demorar el proceso de aprendizaje de un tema determinado (teoría de grafos), para dar paso a otro proceso de aprendizaje y adaptación con una herramienta (Matlab) teniendo como base que se necesita aprender a dominarla por su complejidad, para empezar a desarrollar el tema principal, podría resultar poco o nada beneficioso. Teniendo en cuenta además que no sea exitosa la preparación, solo traería más desconocimiento e incertidumbre que ventajas en el razonamiento lógico del estudiante.

Otro aspecto a valorar es que la Matemática Discreta se imparte en el primer año de la carrera, cuando el estudiante aún no ha entendido en su totalidad la lógica de programación, ni domina las sintaxis de un lenguaje. Sería entonces una sobrecarga en sus estudios imponerle el aprendizaje de un lenguaje de programación nuevo, para entender un tema de otra asignatura. Esto podría crear conflicto con la asignatura programación y la matemática.

1.2.2 R GUI

R inicialmente fue escrito por Robert Gentleman y Ross Ihaka del Departamento de Estadísticas de la Universidad de Auckland. Desde mediados de 1997 el código fuente de R es conducido por un grupo de desarrollo (Development Core Team). R es un sistema para la estadística y gráfica computacional. Provee un lenguaje de programación de alto nivel de gráficos, interfaces y otros lenguajes y facilidades de depuración [8]

Es un programa que maneja el trabajo con gráficas de todo tipo, no es especializada en el trabajo con la teoría de grafos específicamente, está diseñada principalmente para la creación de gráficos como los diagramas de procesos Bernoulli y otros gráficos de estados de procesos. No permite el análisis, ni la aplicación de algoritmos propios de la teoría de grafos.

1.2.3 ALGRAF (ALGORITMOS EN GRAFOS)

Es una herramienta diseñada en Visual Basic, teniendo de autores a Abraham Fernández, Julián Ramírez, Fátima Rico y José Luis Santisteban, que puede ser utilizada para el estudio de los algoritmos sobre grafos. El objetivo de la aplicación es fundamentalmente didáctico, para ayudar a la mejor comprensión de

algunos conceptos y para visualizar de forma animada algoritmos sobre grafos. El programa responde a numerosas cuestiones sobre un grafo o dígrafo (con o sin pesos en las aristas):

- Sucesión de grados, matriz de adyacencia.
- Conectividad, vértices corte, aristas puente, componentes conexas y bloques.
- Operaciones sobre grafos: Grafo complementario y grafo de aristas.
- Árboles: algoritmos de búsqueda, algoritmos de construcción del árbol generador de peso mínimo (Prim, Kruskal y Boruvka) y código de Prüfer de un árbol etiquetado.
- Caminos en un grafo o dígrafo sin pesos o con pesos (algoritmo de Dijkstra).
- Recorridos eulerianos: existencia, construcción con los algoritmos de Hierholzer, Fleury y Tucker.
- Problema del cartero.
- Coloración: algoritmos secuenciales, variantes y algoritmo de Brelaz.

Luego instalado de la dirección de la página *web* de los autores[5], puede llegarse a varias conclusiones acerca de sus funcionalidades.

- Es un programa interactivo y didáctico, que trata varios conceptos de grafos como se describen anteriormente.
- Es sencillo de manejar porque cuenta con una interfaz simple y una buena organización.
- Permite trabajar visualmente con los grafos.

La principal desventaja que presenta esta herramienta es que se enfoca demasiado en las representaciones gráficas, obviando por completo temas como la representación por conjunto y la creación de grafos por matriz de adyacencia. Tampoco se refiere a la matriz de incidencia asociada a un grafo ni a los conceptos como secuencia de vértices, cadenas, ciclos o circuitos.

[5] www.dma.fi.upm.es/gregorio/grafos/ALGRAF.html

1.3 SELECCIÓN DE METODOLOGÍA, LENGUAJES Y HERRAMIENTAS

La Ingeniería de *Software* puede analizarse como "una tecnología multicapa", ilustrada en la Fig. 13.

Fig. 13 Capas de la Ingeniería de Software.

Dichas capas se describen a continuación [9]:

- La gestión total de la **calidad** y las filosofías similares fomentan una cultura continua de mejoras de procesos que conduce al desarrollo de enfoques cada vez más robustos para la ingeniería del *software*.
- El fundamento de la ingeniería de *software* es la capa **proceso**. El proceso define un marco de trabajo para un conjunto de áreas clave, las cuales forman la base del control de gestión de proyectos de *software* y establecen el contexto en el cual se aplican los métodos técnicos, se producen resultados de trabajo, se establecen hitos, se asegura la calidad y el cambio se gestiona adecuadamente.
- Los **métodos** de la ingeniería de *software* indican cómo construir técnicamente el *software*. Los métodos abarcan una gran gama de tareas que incluyen análisis de requerimientos, diseño, construcción de programas, pruebas y mantenimiento. Estos métodos dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.
- Las **herramientas** de la ingeniería del *software* proporcionan un soporte automático o semi-automático para el proceso y los métodos, a estas herramientas se les llama herramientas CASE (por sus siglas en inglés).

Dado lo anterior, el objetivo de la ingeniería de *software* es lograr productos de *software* de calidad (tanto en su forma final como durante su elaboración), mediante un proceso apoyado por métodos y

herramientas. A continuación se realiza la selección de la metodología de desarrollo, las herramientas y los lenguajes.

1.3.1 METODOLOGÍA DE DESARROLLO DE SOFTWARE

Las metodologías se basan en una combinación de los modelos de proceso genéricos (cascada, evolutivo, incremental). Adicionalmente una metodología debería definir con precisión los artefactos, roles y actividades involucrados. Como parte de las metodologías que se proponen para guiar un proceso de desarrollo de *software*, están definidos dos enfoques muy utilizados:

- Las metodologías tradicionales o pesadas que centran su atención en llevar una documentación exhaustiva de todo el proyecto y en cumplir con un plan de proyecto, en la fase inicial del desarrollo del proyecto y que se focalizan en documentación, planificación y procesos [10].
- Las metodologías ágiles que ponen de relevancia de que la capacidad de respuesta a un cambio es más importante que el seguimiento estricto de un plan" [10].

Metodología Ágil	Metodología Tradicional
Pocos Artefactos. El modelado es prescindible, modelos desechables.	Más Artefactos. El modelado es esencial, mantenimiento de modelos
Pocos Roles, más genéricos y flexibles	Más Roles, más específicos
No existe un contrato tradicional, debe ser bastante flexible	Existe un contrato prefijado
Cliente es parte del equipo de desarrollo (además in-situ)	El cliente interactúa con el equipo de desarrollo mediante reuniones
Orientada a proyectos pequeños. Corta duración (o entregas frecuentes), equipos pequeños (< 10 integrantes) y trabajando en el mismo sitio	Aplicables a proyectos de cualquier tamaño, pero suelen ser especialmente efectivas/usadas en proyectos grandes y con equipos posiblemente dispersos
La arquitectura se va definiendo y mejorando a lo largo del proyecto	Se promueve que la arquitectura se defina tempranamente en el proyecto
Énfasis en los aspectos humanos: el individuo y el trabajo en equipo	Énfasis en la definición del proceso: roles, actividades y artefactos
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Se esperan cambios durante el proyecto	Se espera que no ocurran cambios de gran impacto durante el proyecto

Fig. 14 Comparación entre metodologías.

Teniendo en cuenta lo representado en la Fig. 14 puede decirse que las metodologías tradicionales o pesadas se acentúan por un riguroso control en la definición de roles, actividades y artefactos, generando una documentación pormenorizada. Esta perspectiva ha demostrado ser efectiva y necesaria en el desarrollo de *software* para proyectos de gran tamaño (respecto a tiempo y recursos), sin embargo no es la más adecuada donde es necesario reducir el tiempo de desarrollo pero mantener una alta calidad.

Este punto de vista no es el más importante para elegir una metodología ágil sobre una tradicional, sin embargo existen dos aspectos muy significativos que poseen las primeras sobre las últimas, por una parte no tratan de predecir como será el proceso sino adaptarse a los cambios que puedan surgir en las etapas del desarrollo. Aceptar el cambio es una ventaja que permite crecer ante los riesgos y de esta manera evitar que impacten negativamente sobre el proyecto, principalmente porque los requerimientos pueden ser bastante imprecisos y cambiar continuamente, lo que quiere decir que si no se tienen requisitos estables entonces no se está en la posición de tener un plan estable y seguir un proceso planeado.

Por otro lado resulta interesante también que las metodologías ágiles enfoquen su orientación a las personas y no al proceso, esta perspectiva tiene como resultado que el desarrollo de *software* sea una actividad agradable, al mantenerse a favor de la naturaleza humana en lugar de en su contra se crea un ambiente de trabajo que motiva a los integrantes del grupo. Otro criterio para escoger la metodología son las características particulares del equipo:

Solo existen dos desarrolladores y los clientes, los cuales pertenece al equipo de desarrollo (tutores del trabajo), esto permite que sean los mayores interesados en el proceso de desarrollo, ya que se desea que la herramienta cumpla con las condiciones metodológicas para su inclusión en el plan de la asignatura Matemática Discreta. Otro aspecto son los requerimientos, estos pueden cambiar en dependencia de los objetivos de la asignatura, aunque no se debería tomar como que pueden representar como un cambio rápido ni en gran escala.

Ante el impedimento de utilizar estas metodologías tradicionales, las ágiles constituyen una solución para el proceso de desarrollo de la herramienta por estar especialmente orientadas para proyectos pequeños, aportando un reajuste de tiempo sin renunciar a la calidad del producto, donde el cliente forma parte del equipo de desarrollo y existe buena comunicación entre sus integrantes.

Dentro del enfoque ágil existen varias metodologías que han demostrado su eficacia y éxito dependiendo de las características del proyecto. SCRUM puede ser efectiva en proyectos con un rápido cambio de requerimientos; CRYSTAL se enfoca en las personas que componen el equipo y en desarrollar pocos artefactos, así como definir políticas y colores dependiendo del tamaño del mismo; el método Desarrollo

de Software Adaptativo (ASD por sus siglas en inglés) es orientado a los componentes software más que a las tareas y tolerante a los cambios.

A pesar de que estas metodología ágiles tiene valores en común se destaca sobre las demás la Programación Extrema[6] (XP por sus siglas en inglés), formulada por Kent Beck en 1999 utilizando como paradigma de desarrollo el enfoque Orientado a Objetos. Esto se debe a varias características particulares descritas por su creador en "Extreme Programming. Installed", las cuales son:

- Requiere mucha más disciplina para ejecutarse que otras metodologías, esto se ha logrado por la habilidad de sus líderes para construir una docena de prácticas que los proyectos deben seguir. Muchas de estas son técnicas que ya existían y que fueron probadas anteriormente pero que a menudo se olvidaron, incluyendo la mayoría de los procesos planeados. Además de resucitar estas técnicas, XP las teje en un todo sinérgico donde cada una refuerza a las demás.

- Propone que los desarrolladores sean los encargados de estimar el tiempo de las tareas mediante una planeación realizada a fondo y dejándoles la toma de todas las decisiones técnicas. Tal acercamiento requiere compartir una responsabilidad donde desarrolladores y líderes tienen un mismo lugar en la dirección del proyecto.

- Mientras todos los procesos mencionan las pruebas, la mayoría lo hace con muy poco énfasis. Sin embargo XP pone la comprobación como el fundamento del desarrollo, con cada programador escribiendo pruebas cuando escriben su código de producción. Las pruebas se integran en el proceso de integración continua y construcción lo que rinde una plataforma altamente estable para el desarrollo futuro.

- XP construye un proceso de diseño evolutivo que se basa en refactorizar un sistema simple en cada iteración. Todo el diseño se centra en la iteración actual y no se hace nada anticipadamente para necesidades futuras. El resultado es un proceso de diseño disciplinado, combina la disciplina

[6] XP: eXtreme Programming.

con la adaptabilidad de una manera que indiscutiblemente la hace la más desarrollada de entre todas las metodologías adaptables.

Roles

Los roles de acuerdo con la propuesta original de Beck son:

- **Programador:** escribe las pruebas unitarias y produce el código del sistema.
- **Cliente:** escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.
- **Encargado de pruebas (Tester):** Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
- **Encargado de seguimiento (Tracker):** Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.
- **Entrenador (Coach):** Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.
- **Consultor:** es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.
- **Gestor (Big Boss):** es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación

Ciclo de vida

En su ciclo de vida XP define 4 etapas para el desarrollo de un proyecto. A continuación se caracterizan dichas etapas tomadas del libro de Addison- Wesley, escrito por Beck, titulado “Extreme Programming. Explained”:

1. Planificación.

En esta etapa se propone que el cliente realice las siguientes actividades:

- Definir las Historias del Usuario.
- Decidir qué valor de negocio tiene cada Historia. El valor de negocio indica qué importancia tiene esta historia para el negocio.
- Decidir que HU construir en la iteración o entrega.

El programador posee liderazgo técnico para realizar las tareas de:

- Estimar el tiempo de implementación.
- Advertir al cliente sobre los riesgos técnicos significativos.
- Medir el progreso del equipo para una perspectiva del cliente sobre el proceso global del proyecto.

En esta etapa se obtienen como artefactos principales para utilizarse en el diseño: las HU como especificaciones de los requerimientos funcionales del *software*, el Plan de entregas y la planificación de las iteraciones definidas por el equipo de desarrollo.

2. Diseño.

El Diseño XP estimula el uso de las tarjetas CRC[7] para identificar y organizar las clases orientadas a objetos que son relevantes para el incremento del *software*. También se promueve la refactorización, refinamiento iterativo interno del diseño del programa, sin alterar el comportamiento externo del código

Las tarjetas CRC son fichas, una por cada clase, en las que se escriben brevemente las responsabilidades de la clase, y una lista de los objetos con los que colabora para llevar a cabo esas responsabilidades. Se desarrollan normalmente en una sesión de trabajo en grupo pequeño.

3. Código.

[7] CRC: Class Responsibility Collaborator o Clase, Responsabilidad y Colaborador.

En la etapa de Código, XP recomienda la construcción de Pruebas Unitarias antes de que comience la implementación, alentando la programación en parejas (dos personas compartiendo la misma estación de trabajo), teniendo como meta de trabajo 40 horas a la semana, para de esta manera tener tiempo dedicado a la implementación, ya que debe ser una tarea priorizada.

En esta etapa se crean las Tarjetas de Ingeniería para conducir la implementación de las funcionalidades del sistema. Las tarjetas de Ingeniería pueden definirse como las actividades en las que se puede dividir una HU para ser implementada. Estas son asignadas al programador el cual estima los puntos de estimación y la fecha de inicio y fin. Esto permite organización y responsabilidad en la programación como con la utilización de los estándares de programación, que también son buenas prácticas que XP propone para la organización del código y acentuar la comunicación.

4. Prueba

Las pruebas son procesos que permiten verificar y revelar la calidad de un producto *software*, siendo estas uno de los pilares de la metodología XP. Son utilizadas para identificar posibles fallos de implementación, calidad, usabilidad, etc., de un programa o *software*.

XP divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas por el cliente final.

1.3.2 LENGUAJE DE PROGRAMACIÓN

Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos *hardware*[8] y *software* existentes [11].

Antes de escoger un lenguaje de programación sobre el cual implementar la aplicación se deben tener en cuenta varios elementos. Según las necesidades del problema se puede decir que se desea crear una aplicación de escritorio sin servicios de base de datos, que sea pequeña y a la que puedan acceder los

[8] *Hardware* (en inglés): término para referirse a los componentes tangibles de una herramienta tecnológica.

estudiantes y profesores de la asignatura. Se espera que esté instalada en los laboratorios de la universidad sin necesidad de autenticación. Tampoco se requiere del uso de servidores para su funcionamiento ni recursos de red u otras estaciones de trabajo.

Es necesario también escoger el tipo de lenguaje de programación, en este caso hay que tomar en cuenta los lenguajes que pertenecen a las siguientes clasificaciones:

- Lenguajes de alto nivel: por ser más parecidos al lenguaje humano. Además manejan conceptos de una manera cercana al pensamiento humano abstrayéndose del funcionamiento de la máquina.
- Lenguajes orientados a objetos: El concepto de encapsulamiento de la información permite utilizar un objeto, método o procedimiento desde muchas partes del código sin necesidad de repetirlo. Esto ahorra tiempo al programador y le permite facilidad a la hora de programar el código.

Existen varios lenguajes que además de ser de alto nivel, permiten utilizar la Programación Orientada a Objetos, entre ellos se encuentran Java, C# y C++.

Java es desarrollado por *Sun Microsystems* en el año 1995, C# fue desarrollado y estandarizado por *Microsoft* en el año 2000, mientras que C++ fue la creación de Bjarne Stroustrup a mediados de los años 80 [12].

Se escoge el lenguaje Java en su versión 7 por diferentes razones que lo hacen más adecuado al proyecto, primeramente por ser libre, para desarrollar en Java no se necesita comprar licencias, esto permite eximir las ataduras tanto económicas como éticas que el *software* privativo (C# o C++) representa para los países pobres o en vías de desarrollo como lo es Cuba.

Java es de código abierto, esta ventaja permite que ofrezca una colección de librerías también de código abierto, para implementar estructuras de datos como listas, grafos y árboles. Esto permite el trabajo con la teoría de grafos, ya que es una estructura de datos abstracta en programación.

Al tener el código fuente disponible se puede expandir o acoplar a la aplicación. Esto permite heredar la funcionalidad de una clase ya existente y agregar procedimientos adicionales, posteriormente empaquetar todo y distribuir el producto de manera libre.

Tiene una gran variedad de Entornos de Desarrollo Integrados (IDE por sus siglas en inglés) muy potentes y gratuitos como el NetBeans y Eclipse para la implementación del código.

Mantiene un proceso continuado ya que se le sigue dando soporte y actualizando constantemente. Java cuenta con varias comunidades dedicadas a otorgar soporte, algunas están divididas por países o idiomas. Asimismo, la documentación que Oracle pone a disposición sobre cada una de las clases, métodos y componentes del lenguaje es bastante útil y muy completa.

1.3.3 ENTORNO DE DESARROLLO INTEGRADO

Un IDE[9] es un ambiente de programación que ha sido empaquetado como un programa de aplicación, o sea, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDE's pueden ser aplicaciones por sí solos o pueden ser parte de aplicaciones existentes [13].

Luego de escogido el lenguaje de programación es necesario el ambiente en el que se pueda programar que sea compatible con el mismo. Para ello se necesita conocer los IDE's disponibles para utilizar el lenguaje Java en su versión 7:

- Eclipse.
- NetBeans.
- BlueJ.
- JBuilder.
- JCreator**.**

No existe un entorno mejor que otro simplemente, es necesario escoger cual es el más conveniente para el proyecto. Según el sitio oficial de la Comunidad de Desarrollo que lo promueve **Netbeans.org**, NetBeans es: "un entorno de desarrollo integrado, modular, de base estándar (normalizado), escrito en el lenguaje de programación Java". Utilizando código abierto y una plataforma de aplicación se estructura el soporte general o *framework,* que presenta las siguientes características:

[9] IDE: por sus siglas en ingles Integrated Development Enviroment.

- Es el primer IDE suministrando soporte para las tecnologías más actuales de Java como JDK 7, Java EE 6, and JavaFX 2.0.
- Presenta edición de código con completamiento de código y corrección de errores sintácticos y semánticos, plantillas y herramientas de refactorización.
- Posee una interfaz de usuario con corrección del espacio y alineación, donde el *builder*[10] es intuitivo.
- Mantiene soporte para múltiples lenguajes.

Dichas características le permiten a NetBeans ser el más completo, estable y fiable entre otros IDE's como Eclipse e IntelliJ IDEA.

La gestión de paquetes y sus avanzadas detecciones de errores (incluso antes de compilar) resultan más cómodas e intuitivas que en los otros entornos. Incorpora el autocompletado de código que permite programar deprisa, y funciones como el formateado automático del texto y el autocompletado de código ahorran al programador tiempo y errores.

Se escoge NetBeans 7.3 como IDE ya que es ideal para trabajar con el lenguaje de desarrollo JAVA y todos sus derivados. La IDE de NetBeans es perfecta y muy cómoda para los programadores. Tiene un excelente balance entre una interfaz con múltiples opciones y un aceptable completamiento de código.

Un aspecto importante para la elección de este IDE es la condición de producto libre y gratuito sin restricciones de uso, además permite la integración con las siguientes tecnologías libres que se hacen necesario su uso en el proyecto, tales como:

- ***Framework* jUnit 4.10:** el objetivo del uso de jUnit con NetBeans IDE es escribir y ejecutar pruebas repetibles. jUnit es una instancia de la arquitectura xUnit para *frameworks* encargados de

[10] Builder: en español "constructor"

realizar pruebas unitarias, orientado al programador para la realización de pruebas unitarias en lenguaje Java [14].

- **Librería jGraph 5.8.3.1:** el uso de esta librería permite la visualización, creación y análisis de grafos. jGraph es una librería de código abierto para la visualización de grafos escrita en lenguaje Java, compatible con componentes *Swing* tanto visualmente como en el diseño de sus arquitectura [15].

1.3.4 LENGUAJE DE MODELADO

El lenguaje de modelado es la notación (principalmente gráfica) que usan los métodos para expresar un diseño. El modelado de procesos constituye la base para el análisis, a partir del cual se identificarán los aspectos que tienen problemas y por tanto deben ser mejorados [16].

Los lenguajes y notaciones utilizados en la modelación de procesos juegan un papel de suma importancia, pues son los encargados de hacer posible que los procesos propiamente dichos sean entendidos por todas las personas que intervienen, desde los diseñadores del modelo, los especialistas de las tecnologías de la información, los que los ejecutan y los que los controlan y dirigen [17].

Específicamente para la herramienta a desarrollar se necesita un lenguaje que sea gráfico, a fin de especificar y documentar el sistema de un modo estándar, incluyendo aspectos conceptuales tales como funciones del sistema, dado que el desarrollo de todo *software* orientado a objetos requiere que cada una de las partes que comprende a los mismos se visualice, especifique y documente con lenguaje común.

Como no se modelan los procesos del negocio debido a que no se existe un negocio definido claramente no es posible utilizar la notación de modelado para procesos de negocio (BPMN por sus siglas en inglés), que es una agrupación que tiene dentro de sus objetivos principales el crear una notación estándar para el modelado de procesos de negocio [18].

Aunque la metodología en sí no promueve el uso de diagramas, debido a que refuerza su interés en el código, en ocasiones es necesario aclarar un determinado proceso que puede ser difícil de entender de manera teórica, con un diagrama para la mejor visualización y especificación de todas las partes del grupo de desarrollo.

Uno de los lenguajes de modelado mas conocidos es el Lenguaje Unificado de Modelado (*UML*, por sus siglas en inglés), fue originalmente concebido por la Corporación *Rational Software* y tres de los más prominentes metodólogos en la industria de la tecnología y sistemas de información: Grady Booch, James Rumbaugh, e Ivar Jacobson ("*The Three Amigos*"), presentado al *Object Management Group* (OMG) y aprobado por éste como un estándar en noviembre 17 de 1997 [19].

Se escoge este lenguaje en sus versión 2.1 por las siguientes características [20]:

- Es un lenguaje para visualizar, especificar, construir y documentar un sistema de *software*, física y conceptualmente.
- Es independiente del lenguaje de implementación y procesos de desarrollo, por lo que sus diseños se pueden implementar en cualquiera de ellos que soporte las posibilidades de *UML*.
- Como un lenguaje para modelamiento industrialmente estandarizado, no es propiedad de alguien sino un lenguaje libre y totalmente extensible

1.3.5 HERRAMIENTA CASE

Las herramientas *CASE* se definen como diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de *software*, reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del *software* en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores, entre otras [21]

Escogido el lenguaje de modelado *UML*, se necesita una herramienta *CASE* que soporte este lenguaje y permita el manejo y la modelación de diagramas en esta notación. Para ello es necesario un análisis de las mejores herramientas *CASE* para *UML,* y tener en cuenta además que no existe una modelación de la Base de Datos, porque no existe un acceso a datos. Esto descarta las herramientas *PLATINUM ERwin*, (herramienta de diseño de base de datos), *EasyCASE Profesional* (producto para la generación de esquemas de base de datos e ingeniería reversa), *Oracle Designer* (para aplicaciones cliente/servidor), entre otras herramientas conocidas.

Visual Paradigm para *UML* por su parte es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de *software* y consta de módulos que facilitan el trabajo en la construcción del *software* [9] .

Visual Paradigm versión 8.0 se selecciona por su estabilidad de ejecución en diferentes sistemas operativos y la facilidad de abrir y trabajar con un modelo *UML* utilizando el mismo programa sin importar el sistema operativo y sin afectar en absoluto el trabajo hecho. Además guarda todo el modelo en un solo fichero, de manera muy simple para salvarlo y cargarlo posteriormente.

Se integra perfectamente con NetBeans IDE, elemento muy importante para su selección en este trabajo. Además de que la UCI posee la licencia para su utilización.

1.3 CONCLUSIONES

Luego del análisis de las herramientas matemáticas para apoyar el trabajo con la Teoría de Grafos se puede concluir que se necesita construir una herramienta que:

- Se ajuste al proceso de enseñanza aprendizaje de la teoría de grafos en la matemática discreta
- La representación gráfica debe ser primordial
- Aplicar algoritmos, visualizar o trabajar no debe ser complicado para un estudiante que nunca se ha enfrentado con la herramienta.

Como parte del capítulo se escogió el soporte tecnológico que dará continuidad a las etapas del desarrollo de *software*:

- Metodología de *software*: eXtreme Programming (XP).
- Lenguaje de Programación e IDE: NetBeans para el lenguaje Java.
- Lenguaje de Modelado y herramienta CASE: Visual Paradigm para *UML*.

Los resultados obtenidos en estos análisis permitieron obtener conocimientos más profundos de la teoría de grafos, ratificar la necesidad de una nueva herramienta y la selección de las tecnologías que permitan dar comienzo al desarrollo de *software*.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

INTRODUCCIÓN

En el presente capítulo tomando como base el objeto de automatización se realiza la propuesta solución que incluye el modelo de arquitectura según las características de la situación en la que se desenvuelve el problema. Se comienza la etapa de planificación de XP donde se incluye el levantamiento de requerimientos y su especificación en las Historias del Usuario, se planifican las iteraciones en las que se agrupan estas historias para su implementación y entrega, en una escala de tiempo que permita la optimización para diseñar e implementar, valorando el riesgo, mitigando el impacto en el sistema y los posibles cambios que pueden ocurrir en el proceso de desarrollo del *software*.

2.1 DESCRIPCIÓN DEL DOMINIO

En los primeros años de la carrera que se cursa en la UCI, el plan de estudios incluye asignaturas de Ciencias Exactas adecuándose al objetivo de formar Ingenieros en Ciencias Informáticas. A pesar de que en estas asignaturas se utiliza la tecnología como medio de enseñanza y aprendizaje, para la instrucción de la Matemática Discreta los profesores y estudiantes no pueden auxiliarse de una herramienta, que permita un resultado superior en el cumplimiento de los objetivos y una mayor motivación de los estudiantes por la asignatura.

Las herramientas matemáticas que se emplean no presentan funcionalidades capaces de trabajar con la teoría de grafos de forma exclusiva, o su estructura es compleja de entender, lo cual no permite que su utilización sea sencilla para el aprendizaje por parte de estudiantes que provienen de la Enseñanza Media Superior y que se enfrentan a un nuevo sistema de enseñanza más independiente y superior.

2.2 PROPUESTA DE SISTEMA

La propuesta del sistema debe ser una aplicación de escritorio donde las funcionalidades principales sean basadas en los objetivos de la asignatura para el tema de Teoría de Grafos.

Posibilitando un espacio donde se pueda comprobar el nivel de aprendizaje que los estudiantes han adquirido en las clases.

Debe contar con una ayuda para facilitar el entendimiento y un menú sencillo y escrito en un lenguaje simple, con fácil localización de las funcionalidades y operaciones.

Concretamente para satisfacer el objetivo de este trabajo se debe implementar una herramienta que:

- La base fundamental de la misma sea la representación visual de las operaciones, conceptos y algoritmos tratados en la Teoría de Grafos, incluyendo temas que no solo son objetivo de la asignatura sino que son importantes como conocimiento integral,
- Permita la intuición, el análisis, la creación de argumentos y la resolución de inquietudes que los estudiantes formulen durante el proceso de aprendizaje.
- Permita el aumento en sus funcionalidades para llegar a conformar un asistente matemático potente y capaz de resolver todas las funcionalidades que la Matemática Discreta plantea

.En la Fig.16 se representa la propuesta solución descrita de manera general.

Fig. 15 Propuesta solución.

En esta figura se representan las características principales de la propuesta solución, a la cual accederán desde una PC en la cual estará instalada, los usuarios que pueden ser los profesores y estudiantes que lo deseen, sin la necesidad de autenticarse, enmarcado en el dominio de la teoría de grafos (ver Fig. 17).

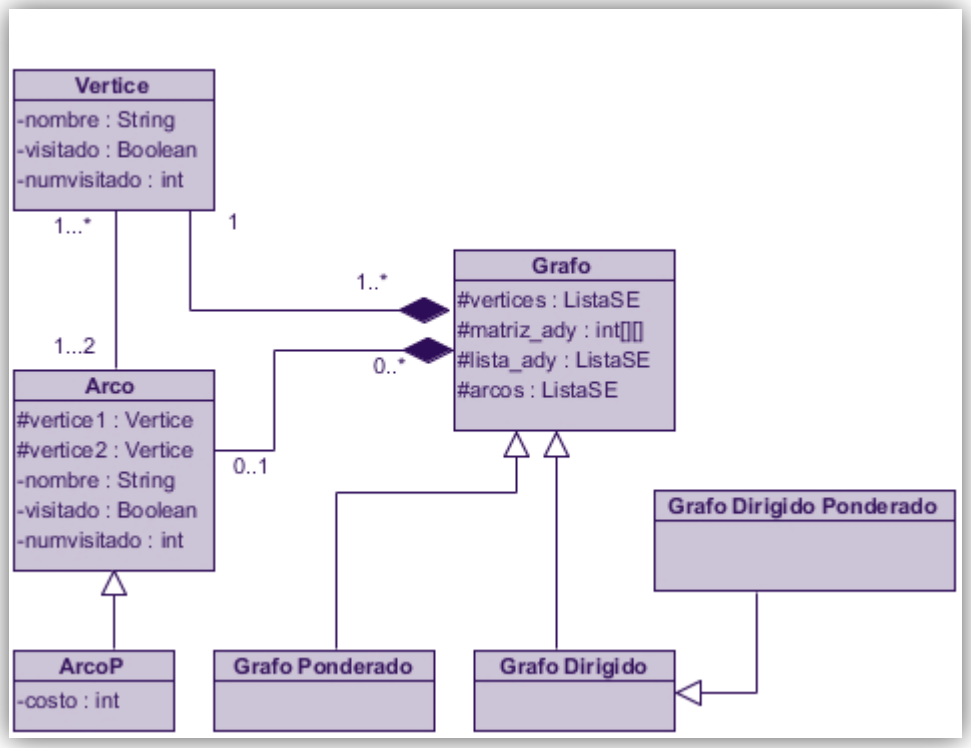


Fig. 16 Modelo de dominio de la aplicación.

2.3 ETAPA DE PLANIFICACIÓN

Fase de Exploración

Esta fase tiene como meta dar una apreciación de lo que el sistema eventualmente debe hacer. Para ellos se definen los requerimientos de software.

2.3.1 Requerimientos funcionales

El cliente especificó 48 requerimientos funcionales de *software*:

R1 Crear grafo analíticamente.

R2 Crear grafo por matriz de adyacencia.

R3 Gestionar grafo.

R3.1 Insertar vértice.

R3.2 Insertar arista.

R3.3 Eliminar.

R3.4 Limpiar pantalla.

R4 Mostrar matriz de adyacencia.

R5 Mostrar matriz de incidencia.

R6 Mostrar estadísticas de un grafo.

R7 Localizar vértice

R8 Localizar arista.

R9 Determinar si dos vértices son adyacentes.

R10 Determinar vértices adyacentes.

R11 Determinar distancia entre vértices.

R12 Determinar los vértices aislados.

R13 Determinar los vértices colgantes.

R14 Determinar grado de un vértice.

R15 Determinar caminos entre un par de vértices.

R16 Determinar cantidad de caminos simples entre un par de vértices.

R17 Determinar cantidad de cadenas entre par de vértices.

R18 Determinar cantidad de caminos simples de un grafo.

R19 Determinar camino entre par de vértices.

R20 Determinar si una secuencia de vértices es un camino.

R21 Determinar si una secuencia de vértices es un camino simple.

R22 Determinar si una secuencia de vértices es una cadena.

R23 Determinar si una secuencia de vértices es un ciclo.

R24 Determinar si una secuencia de vértices es un ciclo simple.

R25 Determinar si una secuencia de vértices es un circuito.

R26 Determinar si un par de secuencias de vértices son disjuntas.

R27 Determinar cadena de Euler.

R28 Determinar cadena cerrada de Euler (Grafo de Euler).

R29 Determinar camino de Hamilton.

R30 Determinar ciclo de Hamilton (Grafo de Hamilton).

R31 Determinar si un grafo es simple

R32 Determinar si un grafo es multígrafo.

R33 Determinar si un grafo es pseudografo.

R34 Determinar si un grafo es nulo.

R35 Determinar si un grafo es un árbol.

R36 Determinar si un grafo es un bosque.

R37 Determinar si un grafo es bipartito.

R38 Determinar si un grafo es conexo.

R39 Determinar componentes conexas.

R40. Crear grafo completo.

R41. Crear grafo completo bipartito.

R42. Crear grafo rejilla.

R43. Crear grafo platónico.

R44. Crear grafo n- cubo.

R45. Crear grafo ciclo.

R46. Crear grafo rueda.

R47. Crear grafo estrella.

R48 Gestionar archivo.

R48.1 Crear nuevo archivo.

R48.2 Abrir archivo.

R48.3 Salvar archivo.

2.3.2 REQUERIMIENTOS NO FUNCIONALES

✓ Requerimientos de usabilidad

- RNF 1 Se necesitará un nivel bajo o medio en computación, ya que el manejo de la aplicación es sencillo, permitiendo al usuario una fácil comprensión y trabajo con la misma.
- RNF 2 El sistema podrá ser utilizado por cualquier persona con conocimientos básicos tanto en sistema operativo Windows como el sistema operativo Linux.
- RNF 3 La aplicación estará y será distribuido en lenguaje de español.

✓ Requerimientos de soporte

- RNF 4 El sistema permitirá la modificación o agregarle nuevas funcionalidades cuando sea necesario, asegurando su extensibilidad y lograr mejores prestaciones en un futuro.

✓ Requerimientos de eficiencia

- RNF 5 La aplicación debe mantener un tiempo de respuesta aceptable en cuanto a la realización de operaciones, graficar, construir los objetos, etc.

✓ Requerimientos de hardware

- RNF 6 Memoria RAM: Mínimo recomendada 1 GB.

- RNF 7 Procesador: Mínimo recomendado Intel Pentium 3.
- RNF 8 Capacidad de almacenamiento en disco duro: Mínimo 5 GB.
- RNF 9 PC Mínimo recomendado Pentium V.
- RNF 10 CPU 133 MHZ

✓ Requerimientos de apariencia o interfaz de usuario

- RNF 11 Las interfaces de la aplicación contarán con los componentes visuales necesarios para las operaciones correspondientes.
- RNF 12 El diseño debe permitir al usuario interactuar con facilidad con la aplicación.

Escribir Historias de Usuario

Definidas por el cliente los requerimientos funcionales y no funcionales del *software*, la fase de exploración propone escribir las HU para una especificación de los requerimientos, en este caso la aplicación posee 48 HU, a continuación se muestran las funcionalidades principales.

Tabla 1. "HU Crear grafo analíticamente"

Historia de Usuario	
Número: 1	**Usuario:** usuario.
Nombre historia: Construir grafo según conjunto	
Valor de Negocio: Alto	**Riesgo:** Alto
Puntos estimados: 5	
Programador responsable: Dagoberto Pérez Hernández	
Descripción: Permite insertar un conjunto de vértices, posteriormente se inserta el conjunto de aristas en el formato especificado por el sistema. Se crea un grafo según los datos	

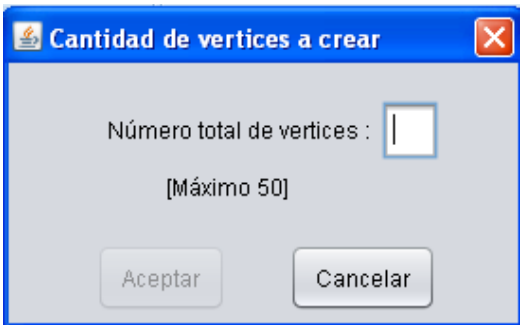
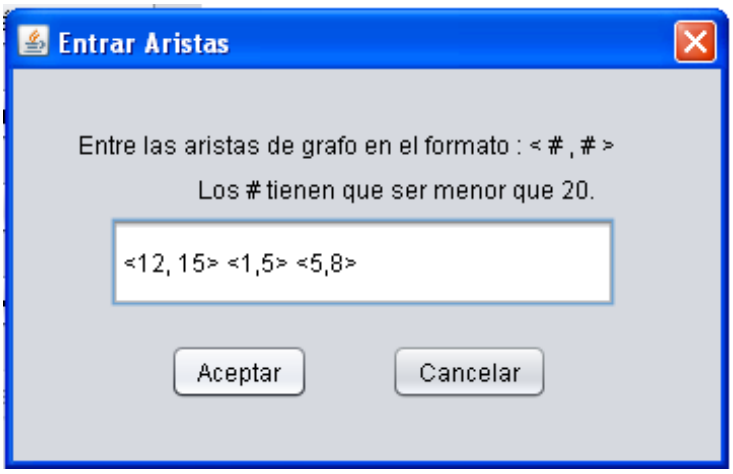
introducidos y se muestra de forma visual en el área de trabajo.
Observaciones: El usuario selecciona la cantidad de vértices del conjunto de vértice. El formato de una arista debe corresponder con: <v1,v2>. Donde v1 y v2 son vértices que pertenecen al conjunto de vértices. Para tipos de grafos dirigidos debe especificarse el orden de los vértices para el conjunto de aristas.
Prototipo de interfaz    

Tabla 2. "HU Crear grafo por matriz"

Historia de Usuario	
Número: 2	**Usuario:** usuario.
Nombre historia: Construir grafo según matriz	
Valor de Negocio: Alto	**Riesgo:** Alto
Puntos estimados: 5	
Programador responsable: Dagoberto Pérez Hernández	
Descripción: Se introduce los datos pertenecientes a la matriz de adyacencia. En esta funcionalidad se debe construir el grafo a partir de estos datos y graficar el resultado en el área. de trabajo	
Observaciones: Para grafos simples la matriz debe ser simétrica, con valores de 0 en la diagonal principal y solo se aceptan valores binarios.	
Prototipo de Interfaz	

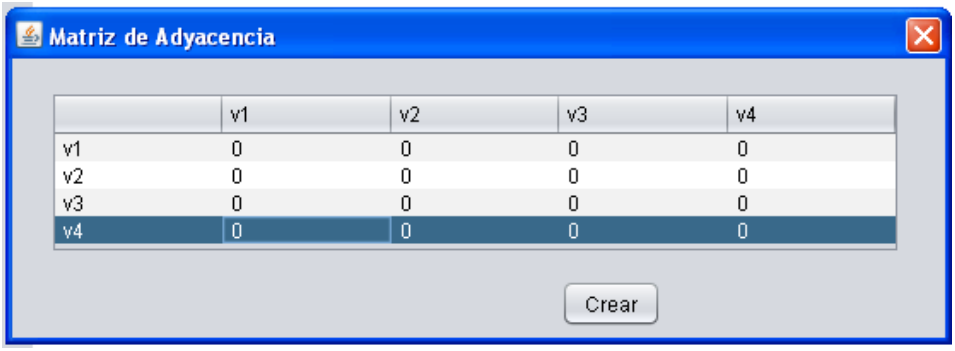
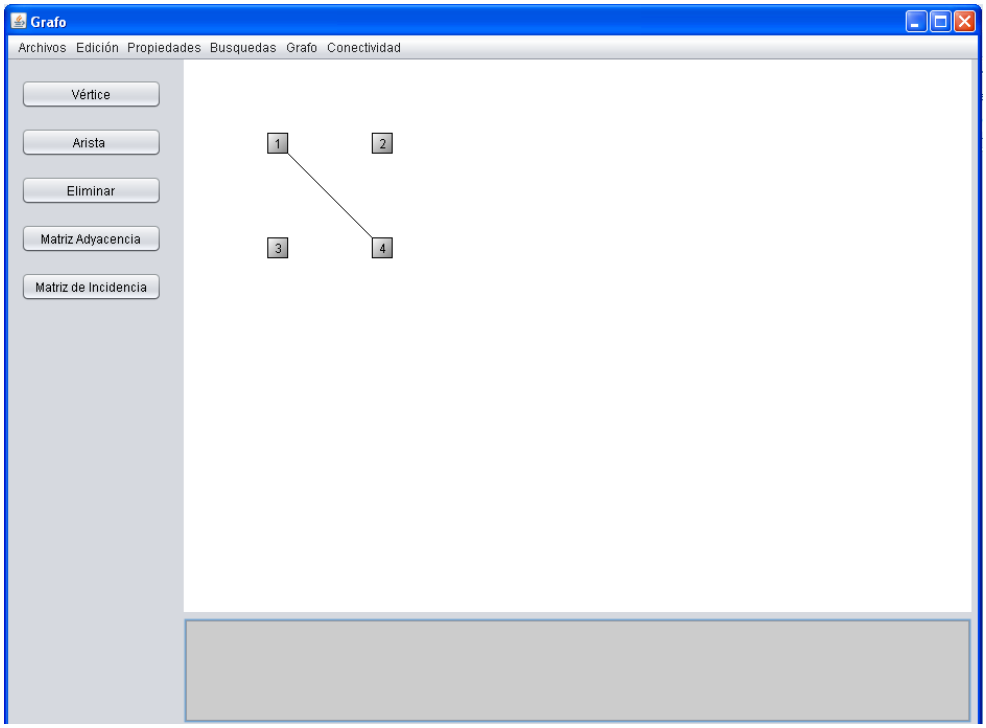


Tabla 3. "HU Gestionar grafo"

Historia de Usuario	
Número: 3	**Usuario:** usuario.
Nombre historia: Gestionar grafo	
Valor de Negocio: Alto	**Riesgo:** Alto
Puntos estimados: 5	
Programador responsable: Dagoberto Pérez Hernández	
Descripción: **Insertar vértice:** Esta opción permite que se inserte en el área de trabajo un nuevo vértice al grafo. **Insertar arista:** Al seleccionar esta opción se construyen las aristas conectando un par de vértices a la vez. **Eliminar:** Para eliminar un elemento del grafo a la vez (arista o vértice) se elige la opción eliminar y posteriormente se selecciona el elemento. **Limpiar pantalla:** Esta opción permite reiniciar el área de trabajo. Con ella se elimina cualquier trabajo que este activo y la pantalla queda limpia.	
Observaciones:	

Al insertar una arista es necesario que existan al menos dos vértices. No se debe elegir un vértice de inicio o final de la arista si no existen en el grafo.
Prototipo de Interfaz 

Para las HU restantes ver Anexos del **¡Error! No se encuentra el origen de la referencia.**.

Estimar Historias

Posteriormente se estiman las tareas (ver Tabla 4), donde los programadores realizan una valoración de cada historia del usuario y especifican un valor de esfuerzo a realizar por cada una. Este valor es escogido a partir de los días ideales de ingeniería, lo cual requiere la opinión del programador acerca de cuanto se estima demorar implementando las historias según el calendario. Cada punto representa un día.

Tabla 4. "Estimación por HU"

No.	Historia de Usuario	Pts. de Estim.
1.	Crear grafo analíticamente.	5
2.	Crear grafo por matriz de adyacencia.	5
3.	Gestionar grafo.	5
4.	Mostrar matriz de adyacencia.	2
5.	Mostrar matriz de incidencia.	2
6.	Mostrar estadísticas de un grafo.	1
7.	Localizar vértice	1
8.	Localizar arista.	1
9.	Determinar si dos vértices son adyacentes.	1
10.	Determinar vértices adyacentes.	1
11.	Determinar distancia entre vértices.	1
12.	Determinar los vértices aislados.	1
13.	Determinar los vértices colgantes.	1
14.	Determinar grado de un vértice.	1
15.	Determinar caminos entre un par de vértices.	1
16.	Determinar cantidad de caminos simples entre un par de vértices.	1
17.	Determinar cantidad de cadenas entre par de vértices.	1

18.	Determinar cantidad de caminos simples de un grafo.	1
19.	Determinar camino entre par de vértices.	1
20.	Determinar si una secuencia de vértices es un camino.	2
21.	Determinar si una secuencia de vértices es un camino simple.	2
22.	Determinar si una secuencia de vértices es una cadena.	2
23.	Determinar si una secuencia de vértices es un ciclo.	3
24.	Determinar si una secuencia de vértices es un ciclo simple.	3
25.	Determinar si una secuencia de vértices es un circuito.	3
26.	Determinar si un par de secuencias de vértices son disjuntas.	1
27.	Determinar cadena de Euler.	3
28.	Determinar cadena cerrada de Euler (Grafo de Euler).	3
29.	Determinar camino de Hamilton.	3
30.	Determinar ciclo de Hamilton (Grafo de Hamilton).	3
31.	Determinar si un grafo es simple	2
32.	Determinar si un grafo es multígrafo.	2
33.	Determinar si un grafo es pseudografo.	2
34.	Determinar si un grafo es nulo.	1
35.	Determinar si un grafo es un árbol.	2
36.	Determinar si un grafo es un bosque.	2
37.	Determinar si un grafo es bipartito.	1

38.	Determinar si un grafo es conexo.	1
39.	Determinar componentes conexas.	3
40.	Crear grafo completo.	3
41.	Crear grafo completo bipartito.	3
42.	Crear grafo rejilla.	2
43.	Crear grafo platónico.	3
44.	Crear grafo n- cubo.	2
45.	Crear grafo ciclo.	2
46.	Crear grafo rueda.	2
47.	Crear grafo estrella.	2
48.	Gestionar archivo.	5

Fase de Compromiso

El objetivo de esta fase es escoger el alcance y la fecha de las entregas. Para ello se definen cuatro actividades:

Ordenar las HU por valor

El cliente define el Valor en Negocio y el programador agrega además a estas HU el riesgo el cual es Alto, Medio y Bajo, luego en el primer movimiento de la fase las HU se ordenan en tres grupos los cuales pueden ser:

(1) aquellas sin las que el sistema no funcionaría,

(2) aquellas que no son esenciales pero proveen un valor significativo de negocio y

(3) aquellas que son agradables de tener.

Tabla 5. "Ordenamiento de HU por valor de negocio"

Grupo	Historia de Usuario	Valor	Riesgo
1.	Crear grafo analíticamente.	1	Alto
	Crear grafo por matriz de adyacencia.	1	Alto
	Gestionar grafo.	1	Alto
	Mostrar matriz de adyacencia.	1	Alto
	Mostrar matriz de incidencia.	1	Alto
	Determinar cadena de Euler.	1	Alto
	Determinar cadena cerrada de Euler (Grafo de Euler).	1	Alto
	Determinar camino de Hamilton.	1	Alto
	Determinar ciclo de Hamilton (Grafo de Hamilton).	1	Alto
	Crear grafo completo.	1	Alto
	Crear grafo completo bipartito.	1	Alto
	Crear grafo rejilla.	1	Alto
	Crear grafo platónico.	1	Alto
	Crear grafo n- cubo.	1	Alto
	Crear grafo ciclo.	1	Alto
	Crear grafo rueda.	1	Alto
	Crear grafo estrella.	1	Alto
2.	Determinar si dos vértices son adyacentes.	2	Medio

	Determinar vértices adyacentes.	2	Medio
	Determinar distancia entre vértices.	2	Medio
	Determinar los vértices aislados.	2	Medio
	Determinar los vértices colgantes.	2	Medio
	Determinar grado de un vértice.	2	Medio
	Determinar caminos entre un par de vértices.	2	Medio
	Determinar cantidad de caminos simples entre un par de vértices.	2	Medio
	Determinar cantidad de cadenas entre par de vértices.	2	Medio
	Determinar cantidad de caminos simples de un grafo.	2	Medio
	Determinar camino entre par de vértices.	2	Medio
	Determinar si una secuencia de vértices es un camino.	2	Medio
	Determinar si una secuencia de vértices es un camino simple.	2	Medio
	Determinar si una secuencia de vértices es una cadena.	2	Medio
	Determinar si una secuencia de vértices es un ciclo.	2	Medio
	Determinar si una secuencia de vértices es un ciclo simple.	2	Medio
	Determinar si una secuencia de vértices es un circuito.	2	Medio
	Determinar si un par de secuencias de vértices son disjuntas.	2	Medio
	Determinar si un grafo es simple	2	Medio
	Determinar si un grafo es multígrafo.	2	Medio
	Determinar si un grafo es pseudografo.	2	Medio

	Determinar si un grafo es nulo.	2	Medio
	Determinar si un grafo es un árbol.	2	Medio
	Determinar si un grafo es un bosque.	2	Medio
	Determinar si un grafo es bipartito.	2	Medio
	Determinar si un grafo es conexo.	2	Medio
	Determinar componentes conexas.	2	Medio
3.	Mostrar estadísticas de un grafo.	3	Bajo
	Localizar vértice	3	Bajo
	Localizar arista.	3	Bajo
	Gestionar archivo.	3	Bajo

Definir la velocidad y el alcance del proyecto

En esta actividad se determina la velocidad del equipo de desarrollo programando en la escala tiempo ideal de ingeniería por mes, este valor es calculado tomando los días que el equipo desarrolla en el mes, teniendo como 5 días ideales de ingeniería a la semana, 4 semanas al mes se puede calcular la velocidad del proyecto como 20 días por mes.

Finalmente el cliente escoge las tarjetas a implementar en la entrega, así como precisa la fecha de entrega basándose en la prioridad, riesgo y estimación de cada tarjeta. El estimado de las fechas es calculado por los puntos de estimación de cada HU y la velocidad del proyecto. Teniendo lo anterior en cuanta se fija la fecha de inicio de la implementación 15 de marzo.

En la tabla 6 puede observarse la planificación de las entregas progresivas de la aplicación.

Tabla 6. "Planificación de Entregas"

Sistema	Entrega 1	Entrega 2	Entrega 3
Herramienta de Apoyo al Trabajo con la Teoría de Grafos.	Fecha: 5/04/2013 Versión 1.0 con las HU de valor 1 y prioridad alta del grupo 1 (Tabla 6)	Fecha 29/05/2013 Versión 1.1 con las HU de valor 2 y prioridad media que pertenecen al grupo 2 (Tabla 6)	Fecha 8/05/2013 Versión final HU de Valor y Prioridad Bajas del grupo 3 (Tabla 6).

2.3.3 PLANIFICACIÓN DE LAS ITERACIONES

Esta actividad tiene la importancia de dividir las entregas en iteraciones, de esta manera se logra que la implementación de las HU sean programadas en iteraciones que conformen una entrega, logrando que se tenga una mejor organización en el cumplimiento de en las tareas.

Las HU se dividieron en 8 iteraciones, debido a que XP propone iteraciones cortas y con pocas tareas. El origen de los requerimientos que definió el usuario permite que se agrupen en conceptos definidos por la teoría de grafos, así van liberándose funcionalidades en dependencia del tema que traten, además de tener en cuenta su prioridad y riesgo para el sistema.

- **Iteración 1**

Esta interacción comprende las HU imprescindibles para el funcionamiento de la aplicación, que presentan el mayor riesgo y complejidad de programación:

- ✓ HU Crear grafo analíticamente.
- ✓ HU Crear grafo por matriz de adyacencia.
- ✓ HU Gestionar grafo.
- ✓ HU Mostrar matriz de adyacencia.
- ✓ HU Mostrar matriz de incidencia.

- **Iteración 2**

Aunque las HU que se insertan dentro de esta iteración son de prioridad alta, se agrupan en una segunda evolución de las iteraciones debido a que en ellas se determinan los grafos de Euler y Hamilton.

- ✓ HU Determinar cadena de Euler.
- ✓ HU Determinar cadena cerrada de Euler (Grafo de Euler).
- ✓ HU Determinar camino de Hamilton.
- ✓ HU Determinar ciclo de Hamilton (Grafo de Hamilton).

- **Iteración 3**

Las HU que se implementan en esta iteración también pertenecen al grupo que presentan alta prioridad en el sistema y con ellas pueden visualizarse grafos predefinidos por diferentes características como la forma y la disposición de sus vértices.

- ✓ HU Crear grafo completo.
- ✓ HU Crear grafo completo bipartito.
- ✓ HU Crear grafo rejilla.
- ✓ HU Crear grafo platónico.
- ✓ HU Crear grafo n- cubo.
- ✓ HU Crear grafo ciclo.
- ✓ HU Crear grafo rueda.
- ✓ HU Crear grafo estrella.

- **Iteración 4**

Al inicio de esta iteración se realiza la primera entrega de la aplicación con las principales funcionalidades definidas por el usuario para ello. Las HU que pertenecen al grupo que presentan prioridad media y que son funcionalidades para determinar propiedades de vértices, se implementan en la cuarta iteración.

- ✓ HU Determinar si dos vértices son adyacentes.
- ✓ HU Determinar vértices adyacentes.
- ✓ HU Determinar distancia entre vértices.

- ✓ HU Determinar los vértices aislados.
- ✓ HU Determinar los vértices colgantes.
- ✓ HU Determinar grado de un vértice.

- **Iteración 5**

En esta iteración se implementan las funcionalidades referidas en las HU de prioridad media que tratan temas de caminos, y cadenas.

- ✓ HU Determinar caminos entre un par de vértices.
- ✓ HU Determinar cantidad de caminos simples entre un par de vértices.
- ✓ HU Determinar cantidad de cadenas entre par de vértices.
- ✓ HU Determinar cantidad de caminos simples de un grafo.
- ✓ HU Determinar camino entre par de vértices.

- **Iteración 6**

Esta iteración es una continuación de la iteración anterior, pero en ella se implementan las HU que presentan funcionalidades de secuencias de vértices.

- ✓ HU Determinar si una secuencia de vértices es un camino.
- ✓ HU Determinar si una secuencia de vértices es un camino simple.
- ✓ HU Determinar si una secuencia de vértices es una cadena.
- ✓ HU Determinar si una secuencia de vértices es un ciclo.
- ✓ HU Determinar si una secuencia de vértices es un ciclo simple.
- ✓ HU Determinar si una secuencia de vértices es un circuito.
- ✓ HU Determinar si un par de secuencias de vértices son disjuntas.

- **Iteración 7**

Para implementar en esta iteración se seleccionan las HU de prioridad media que muestran otras clasificaciones de grafos, que dependen de las aristas incidentes en los vértices, conectividad, etc.

- ✓ HU Determinar si un grafo es simple
- ✓ HU Determinar si un grafo es multígrafo.
- ✓ HU Determinar si un grafo es pseudografo.
- ✓ HU Determinar si un grafo es nulo.
- ✓ HU Determinar si un grafo es un árbol.
- ✓ HU Determinar si un grafo es un bosque.
- ✓ HU Determinar si un grafo es bipartito.
- ✓ HU Determinar si un grafo es conexo.
- ✓ HU Determinar componentes conexas.

- **Iteración 8**

Las restantes HU que son de prioridad baja y aportan algún valor al negocio se modelan en esta iteración, luego de que se entregue la segunda versión del producto.

- ✓ HU Mostrar estadísticas de un grafo.
- ✓ HU Localizar vértice
- ✓ HU Localizar arista.
- ✓ HU Gestionar archivo.

Terminada esta iteración se entrega la tercera versión del producto y permite la realización de las pruebas de aceptación por parte del cliente.

2.4 CONCLUSIONES PARCIALES

En este capítulo se obtuvieron resultados importantes que permiten la continuidad en el proceso de desarrollo de *software*.

- La propuesta solución que permita satisfacer el objetivo general del trabajo y fijar una meta de cómo debe cualitativamente funcionar la aplicación, enmarcar el problema y trazarse los objetivos para solventarlos.
- La definición y diseño de las HU para determinar las funcionalidades del sistema, de esta manera sentar las bases del diseño simple que propone XP.
- La planificación de las entregas e iteraciones que permite una estimación y rigor en el cumplimiento de las fechas para los *releases*[11].

[11] Releases: en español entregas.

CAPÍTULO 3: DISEÑO DEL SISTEMA

INTRODUCCIÓN

En todo desarrollo de sistemas de *software* es necesario seguir especificaciones que permita a los desarrolladores mantener una disciplina para obtener la coherencia en todas sus etapas. El diseño de *software* provee una guía para construir una arquitectura sólida y una implementación lo más eficiente posible. En este capítulo se especifica el diseño arquitectónico así como todo lo comprendido en él, basándose en la metodología XP y su propuesta, para que la próxima etapa de implementación sea análoga con el objetivo del proyecto.

3.1 DISEÑO ARQUITECTÓNICO

El diseño es la etapa de desarrollo de *software* que tiene como objetivo producir un modelo o representación que posteriormente se va a implementar, para ello es necesario definir los requerimientos funcionales y no funcionales que puedan ser traducidos y orientados a una representación para el *software* [22].

En todo diseño debe cumplirse que los requerimientos obtenidos con anterioridad en otra etapa del desarrollo deben quedar explícitos en la modelación. Debe poder ser leída y entendida con claridad por los programadores, los probadores y por las personas que se encargan del mantenimiento del *software*, es decir que proporcione una idea completa del sistema.

En la descripción de un diseño orientado a objetos, la tarea principal es la asignación cuidadosa de responsabilidades a los componentes de *software*, puesto que es una actividad que debe efectuarse mientras se dibuja un diagrama *UML* o programando, ya que influye fuertemente sobre la robustez, mantenimiento y reutilización de los componentes *software*. Por lo tanto este epígrafe se centra en esta tarea.

3.2 PATRONES DE DISEÑO

Los patrones GRASP[12] constituyen un enfoque para la comprensión y utilización de los principios de diseño, se basa en los patrones de asignación de responsabilidades [23]. Dentro de estos patrones se encuentran los que se refieren a cuestiones muy básicas, comunes y a aspectos fundamentales del diseño tales como:

- ✓ **Experto en Información.**

  Al aplicar este patrón se busca como objetivo asignar las responsabilidades a las clases expertas en información. Se debe crear un diseño que seleccionadas las clases que contengan información para determinada acción o método, se le asigne la responsabilidad a la misma. Con esto se logra que se mantenga el encapsulamiento de la información, puesto que los objetos utilizan su propia información para llevar a cabo las tareas. Normalmente, esto conlleva un bajo acoplamiento, lo que da lugar a sistemas más robustos y más fáciles de mantener. Se distribuye el comportamiento entre las clases que contienen la información requerida, por tanto, se estimula las definiciones de clases más cohesivas y ligeras que son más fáciles de entender y mantener. Se soporta normalmente una alta cohesión.

- ✓ **Creador.**

  Con este patrón se define como debe funcionar la creación de instancias, actividad que es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia, es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien, el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización. Al aplicar este patrón se logra que desde el diseño se le asigne la responsabilidad de creación de instancias a las clases que necesiten utilizar el objeto creado en alguna situación, con esto se favorece el bajo acoplamiento, lo que implica menos dependencias de mantenimiento y mayores oportunidades para reutilizar.

[12] *General Responsability Assignation Software Patterns* (en inglés): Patrones Generales de Software para Asignaciones de Responsabilidades.

✓ **Alta Cohesión.**

Definir las relaciones entre componentes de un sistema es importante para delimita las funciones de cada uno y permite que las clases que no son de manera directa encargadas de la responsabilidad, colaboren con las realmente responsables de la acción a no estar sobrecargadas, permitiendo la alta cohesión entre ellas. Si en el diseño se logra asignar responsabilidades de manera correcta es posible lograr una alta cohesión entre clases, subsistemas, sistemas, etc. Una mala cohesión causa, normalmente, un mal acoplamiento, y viceversa.

✓ **Bajo Acoplamiento.**

Un bajo acoplamiento entre clases significa simplemente que la dependencia de una clase con otra sea la mínima, y queda a decisión del desarrollador si este es perjudicial o beneficioso para su sistema. Es necesario lograr que el diseño permita que el acoplamiento entre clases sea el mínimo necesario para que el sistema funcione correctamente. El bajo acoplamiento soporta el diseño de clases que son más independientes, lo que reduce el impacto del cambio.

No se puede considerar de manera aislada a otros patrones como el Experto o el de Alta Cohesión, sino que necesita incluirse como uno de los diferentes principios de diseño que influyen en la elección para asignar las responsabilidades. El objetivo del diseño es lograr un sistema donde cada uno de sus componentes sea cohesivo y débilmente acoplado para lograr la modularidad. Esto se alcanza diseñando cada método con un único y claro objetivo, agrupando un conjunto de aspectos relacionados en una clase.

✓ **Controlador.**

Este patrón se utiliza cuando es necesario que un componente sea el encargado de dirigir cada uno de los componentes restantes. Si las clases a las que no se le asignan responsabilidades globales las delegan a otras que realmente controlen todo el sistema, permite:

- operar sobre los eventos externos que influyen en él.
- eliminar o suavizar la sobrecarga de cada uno de los componentes restante que no tienen fundamento para tener métodos innecesarios

- agrupar las funciones elementales, generales e imprescindibles para dirigir el sistema, evitando que las clases estén altamente acopladas y pobremente cohesionadas.

3.3 PATRONES DE ARQUITECTURA

Un patrón arquitectónico describe a una familia de sistemas informáticos en términos de su organización estructural, describe además componentes y las relaciones entre ellos con las restricciones de la aplicación, la composición asociada y el diseño para su construcción [22].

Arquitectura N capas

El objetivo primordial es la separación de la lógica de los componentes en capas. Este estilo arquitectónico aplicado presenta una ventaja principal, dividir la aplicación en niveles y, si durante la implementación de sus módulos ocurren cambios, solo se ataca el nivel requerido sin tener que revisar el resto del código asociado, esto permite un bajo acoplamiento entre los componentes que no se relacionan directamente.

Los componentes del *software* se organizan en tres capas lógicas, las cuales están definidas de esta forma:

1era Capa: Presentación, esta capa se encarga de proveer una interfaz entre el sistema y el usuario. Primordialmente, se responsabiliza de que se le comunique información al usuario por parte del sistema y viceversa.

2da Capa: Negocio, es la capa encargada de mantener la comunicación entre la capa superior y la inferior. Permitiendo así que no existan violaciones en los accesos de una capa a otra.

3ra Capa: Dominio, que es la capa que contiene los procesos a realizar con la información recibida a través de la capa de presentación, las peticiones que el usuario realiza, y responsabilizándose de que se le envíen las respuestas adecuadas.

La Fig. 18 representa la organización lógica del sistema.

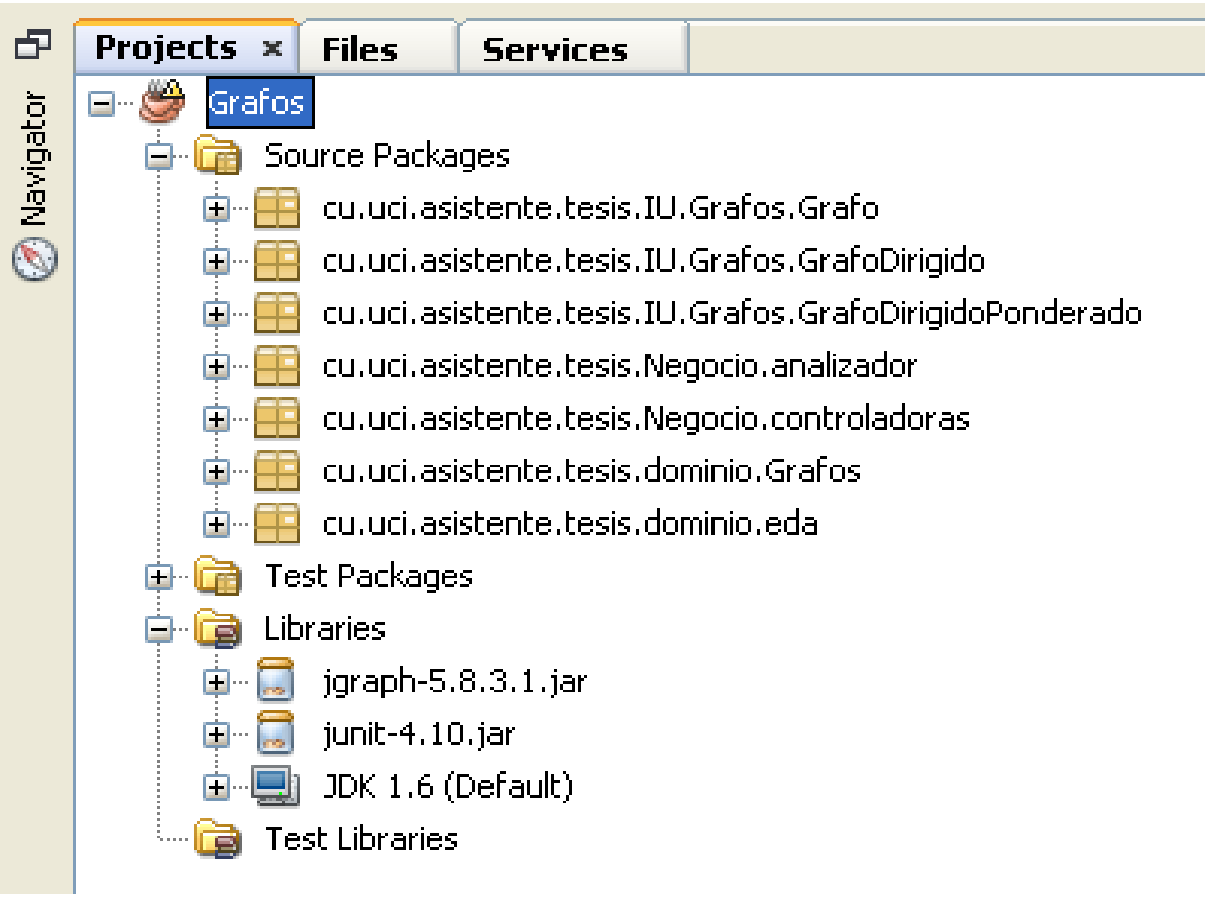


Fig. 17 Arquitectura del Sistema.

3.4 TARJETAS CRC

Clase Grafo	
Description: Esta clase es la encargada de tener la información de los grafos no dirigidos, de ella heredan las demás clases Grafo dirigido, ponderado y ponderado dirigido.	
Responsibilities:	
Name	Collaborator
Insertar Vértice	Controlador CGrafos
Eliminar Vértice	
InsertarArco	
EliminarArco	
Son Adyacentes	
Adyacentes A	
Grado Vértice	
Distancia de Vértices	
Es simple	
Es multigrafo	
Es pseudografo	
Es conexo	
Es nulo	
Es regular	
Es completo	
Es árbol	
Es bipartito	
Vértices aislados	
Vértices colgantes	
Matriz de Incidencia	
Matriz de Adyacencia	
Es camino	
Es camino simple	
Es cadena	
Es ciclo	
Es ciclo simple	
Es circuito	
Determinar caminos simples	
Cantidad caminos simples	
Son disjuntos	

Fig. 18 Tarjeta CRC Clase Grafo.

Clase CGrafos	
Description: Controla las operaciones entre la Interfaz Grafo y la clase Grafo	
Responsibilities:	
Name	Collaborator
Crear grafo desde matriz	Clase Grafo, Interaz Grafo
Crear grafo desde conjunto	Clase Grafo, Interaz Grafo

Fig. 19 Tarjeta CRC Clase CGrafos.

Clase GrafoD	
Description: Clase experta en la información de los grafos dirigidos. Implementa todos los métodos de la clase grafo pero especializados para los dirigidos.	
Responsibilities:	
Name	Collaborator
Crear arista dirigida	Controlador CGrafosD
Es débilmente conexo	Controlador CGrafosD
Grado negativo	Controlador CGrafosD
Grado positivo	Controlador CGrafosD

Fig. 20 Tarjeta CRC clase GrafoD.

Clase GrafoP	
Description: Clase experta en la información de los grafos ponderados, hereda todas los métodos de la clase padre Grafo y los especializa para los grafos ponderados	
Responsibilities:	
Name	Collaborator
Insertar Arista Ponderada	Clase CGrafoP
Modificar Peso	Clase CGrafoP

Fig. 21 Tarjeta CRC Clase Grafo Ponderado

3.5 CONCLUSIONES

Luego de diseñada la arquitectura para dar paso a la implementación del sistema se puede concluir que:

- Con la construcción de las tarjetas CRC y la aplicación de los patrones GRASP, las clases obtuvieron su responsabilidad y se pudieron especificar las clases colaboradoras en las funcionalidades. Esto permite determinar la relación entre los componentes del sistema.
- El diagrama de paquetes permite obtener una idea general de la organización de los componentes del sistema.
- Con la obtención de la arquitectura se facilita la tarea de implementar el código, ya que define el comportamiento del sistema.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

INTRODUCCIÓN

La última etapa en el ciclo de vida de XP es la etapa de Pruebas, que se realiza luego de la implementación, entre ellas se encuentran las pruebas unitarias efectuadas al código, las cuales son diseñadas y ejecutadas por el equipo de desarrollo, utilizando el *framework* Junit. También XP propone las pruebas de aceptación del cliente, realizadas a las funcionalidades para comprobar su correcto funcionamiento. De esta manera se comprueba que el producto final aprueba los requerimientos propuestos por el cliente. Por su importancia este capítulo se dedica a la implementación y las pruebas propiamente dichas.

4.1 IMPLEMENTACIÓN CON XP

4.1.1 ESTÁNDARES DE CODIFICACIÓN

XP promueve la programación basada en estándares, de manera que sea fácilmente entendible por todo el equipo, y que facilite la recodificación.

- Escritura en forma de “camello”, para la declaración de los nombres de clases, métodos e interfaces. Esto se refiere al primer carácter de cada palabra en mayúscula seguida de las demás letras que la componen en minúscula, sin separación de las palabras que compongan el nombre ni la utilización de los *underscore*.
- Todas las declaraciones de los nombres de variables se representaran en minúscula.
- El nombre de los paquetes tendrán la misma estructura: cu.uci.asistente.tesis. “nombre de la capa”. “nombre del paquete”, escrito todo en minúscula.
- Todos los métodos serán comentados.

4.1.2 TARJETAS DE TAREAS DE INGENIERÍA

A continuación se muestran las tarjetas de tareas de ingeniería principales por cada una de las iteraciones planificadas para la implementación de la aplicación.

Iteración 1

Tabla 7. "Tarea de Ingeniería Crear Grafo según matriz"

Tarea de Ingeniería	
Numero de Tarea: 1	**Historia de Usuario:** HU1 Crear grafo analíticamente
Nombre de Tarea: Crear Grafo según matriz	
Tipo de Tarea: Desarrollo	**Puntos de Estimación:** 5
Fecha Inicio: 15/03/2013	**Fecha Fin:** 21/03/2013
Programador Responsable: Dagoberto Pérez Hernández	
Descripción: Permite insertar un conjunto de vértices, posteriormente se inserta el conjunto de aristas en el formato especificado por el sistema. Se crea un grafo según los datos introducidos y se muestra de forma visual en el área de trabajo.	

Tabla 8. "Tarea de Ingeniería Crear grafo por matriz"

Tarea de Ingeniería	
Numero de Tarea: 2	**Historia de Usuario:** HU2 Crear grafo por matriz de adyacencia
Nombre de Tarea: Crear grafo por matriz	
Tipo de Tarea: Desarrollo	**Puntos de Estimación:** 5
Fecha Inicio: 15/03/2013	**Fecha Fin:** 21/03/2013
Programador Responsable: Dagoberto Pérez Hernández	
Descripción: Se introduce los datos pertenecientes a la matriz de adyacencia. En esta funcionalidad se debe construir el grafo a partir de estos datos y graficar el resultado en el área de trabajo.	

Tabla 9. “Tarea de Ingeniería Gestionar grafo”

Tarea de Ingeniería	
Numero de Tarea: 3	**Historia de Usuario:** HU3 Gestionar grafo
Nombre de Tarea: Gestionar grafo	
Tipo de Tarea: Desarrollo	**Puntos de Estimación:** 5
Fecha Inicio: 15/03/2013	**Fecha Fin:** 21/03/2013
Programador Responsable: Dagoberto Pérez Hernández	
Descripción: **Insertar vértice:** Esta opción permite que se inserte en el área de trabajo un nuevo vértice al grafo. **Insertar arista:** Al seleccionar esta opción se construyen las aristas conectando un par de vértices a la vez. **Eliminar:** Para eliminar un elemento del grafo a la vez (arista o vértice) se elige la opción eliminar y posteriormente se selecciona el elemento. **Limpiar pantalla:** Esta opción permite reiniciar el área de trabajo. Con ella se elimina cualquier trabajo que este activo y la pantalla queda limpia.	

Iteración 2

Tabla 10. "Tarea de Ingeniería Determinar cadena de Euler"

Tarea de Ingeniería	
Numero de Tarea: 6	**Historia de Usuario:** HU27 Determinar cadena de Euler
Nombre de Tarea: Euler	
Tipo de Tarea: Desarrollo	**Puntos de Estimación:** 3
Fecha Inicio: 25/03/2013	**Fecha Fin:** 28/03/2013
Programador Responsable: Dagoberto Pérez Hernández	
Descripción: En un grafo construido en el área de trabajo se determina si existe una cadena de Euler.	

Iteración 3

Tabla 11. "Tarea de Ingeniería Crear grafo completo"

Tarea de Ingeniería	
Numero de Tarea: 10	**Historia de Usuario:** HU40 Crear grafo completo
Nombre de Tarea: Completo	
Tipo de Tarea: Desarrollo	**Puntos de Estimación:** 3
Fecha Inicio: 28/03/2013	**Fecha Fin:** 1/04/2013
Programador Responsable: Dagoberto Pérez Hernández	
Descripción: En función del número de vértices introducidos por el usuario se crea un grafo completo.	

Iteración 4

Tabla 12. "Tarea de Ingeniería Determinar si dos vértices son adyacentes"

Tarea de Ingeniería	
Numero de Tarea: 18	**Historia de Usuario:** HU9 Determinar si dos vértices son adyacentes
Nombre de Tarea: Vértices adyacentes	
Tipo de Tarea: Desarrollo	**Puntos de Estimación:** 1
Fecha Inicio: 3/04/2013	**Fecha Fin:** 4/04/2013
Programador Responsable: Dagoberto Pérez Hernández	
Descripción: Creado un grafo en el área de trabajo, se selecciona un par de vértices. El sistema determina si son adyacentes.	

Iteración 5

Tabla 13. "Tarea de Ingeniería Determinar caminos entre un par de vértices"

Tarea de Ingeniería	
Numero de Tarea: 24	**Historia de Usuario:** HU15 Determinar caminos entre un par de vértices
Nombre de Tarea: Caminos entre par de vértices	
Tipo de Tarea: Desarrollo	**Puntos de Estimación:** 1
Fecha Inicio: 4/04/2013	**Fecha Fin:** 5/04/2013
Programador Responsable: Dagoberto Pérez Hernández	
Descripción: Construido un grafo permite determinar la cantidad de caminos que existen entre un par de vértices.	

Iteración 6

Tabla 14. "Tarea de Ingeniería Determinar si un par de secuencias de vértices son disjuntas"

Tarea de Ingeniería	
Numero de Tarea: 35	**Historia de Usuario:** HU26 Determinar si un par de secuencias de vértices son disjuntas
Nombre de Tarea: Disjuntas	
Tipo de Tarea: Desarrollo	**Puntos de Estimación:** 1
Fecha Inicio: 13/04/2013	**Fecha Fin:** 14/04/2013
Programador Responsable: Dagoberto Pérez Hernández	
Descripción: Permite verificar si dos secuencias de vértices son disjuntas, en un grafo construido en el área de trabajo.	

Iteración 7

Tabla 15. "Tarea de Ingeniería Determinar si un grafo es simple"

Tarea de Ingeniería	
Numero de Tarea: 36	**Historia de Usuario:** HU31 Determinar si un grafo es simple
Nombre de Tarea: Simple	
Tipo de Tarea: Desarrollo	**Puntos de Estimación:** 2
Fecha Inicio: 14/04/2013	**Fecha Fin:** 16/04/2013
Programador Responsable: Dagoberto Pérez Hernández	
Descripción: Construido un grafo en el área de trabajo, permite seleccionar la opción que determina si el grafo es simple.	

Las tarjetas restantes a las tareas de ingeniería pueden encontrarse en los Anexos **¡Error! No se ncuentra el origen de la referencia.**.

4.2 PRUEBAS

4.2.1 PRUEBAS UNITARIAS

Las pruebas unitarias (Fig. 24) permiten probar el correcto funcionamiento del código de un componente para asegurarse del correcto funcionamiento de cada uno por separado. El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Las mismas son realizadas por el equipo de trabajo [24].

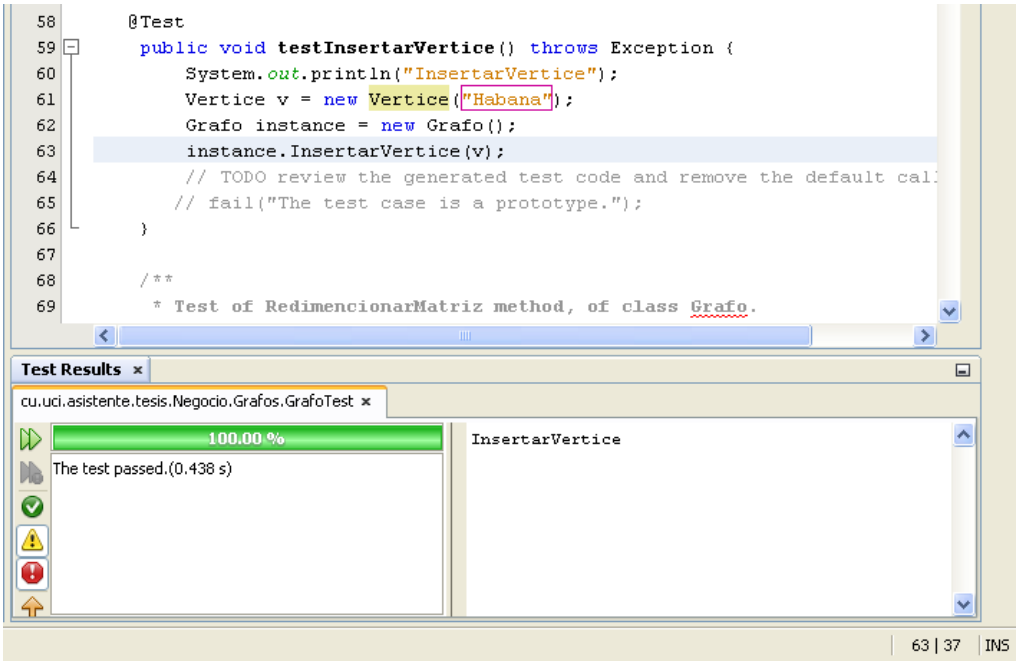


Fig. 22 Prueba Unitaria Insertar vértice.

En los Anexos del **¡Error! No se encuentra el origen de la referencia.** se documentan las pruebas unitarias realizadas a funcionalidades básicas.

4.2.2 PRUEBAS DE ACEPTACIÓN

Las pruebas de aceptación son creadas a partir de las historias de usuario. Durante una iteración la historia de usuario seleccionada en la planificación de iteraciones se convertirá en una prueba de aceptación. El cliente o usuario especifica los aspectos a testear cuando una historia de usuario ha sido correctamente implementada [24].

Tabla 16. "Resultados de las pruebas de aceptación del cliente"

No conformidad	Iteración	Estado
La matriz de incidencia no mostraba los vértices correctamente.	1ra	Resuelta
Validar la matriz de adyacencia para que el usuario pudiese insertar valores solamente en la mitad superior, ya que la matriz es simétrica para grafos no dirigidos y ponderados.	1ra	Resuelta
Al dibujar múltiples aristas entre par de vértices, se muestran superpuestas.	1ra	Pendiente
El campo para introducir la cantidad de vértices para crear la matriz de adyacencia no validaba correctamente	1ra	Resuelta
Al crear el grafo analíticamente no se activaban las funcionalidades de Hamilton.	2da	Resuelta
No se coloreaba el camino entre dos pares de vértices.	3ra	Resuelta
Para grafos dirigidos no coloreaba los vértices y aristas en la funcionalidad determinar distancia entre par de vértices.	3ra	Resuelta
Para grafos ponderados dirigidos calculaba erróneamente la longitud de un camino entre par de vértices.	3ra	Resuelta
Las funcionalidades en el menú para mostrar grafos de formas, se activan o desactivan incorrectamente.	4ta	Resuelto
Algunas funcionalidades quedaban activas al mostrar un resultado de una funcionalidad que no permitía que estuviesen activas las restantes hasta presionar el botón	4ta	Resuelta

“Aceptar”.		
No determinaba en su totalidad cuando dos caminos eran disjuntos y no existía diferencia al colorearlos.	4ta	Resuelta
Calculaba erróneamente la cantidad de caminos simples entre par de vértices.	5ta	Resuelta
Al ejecutar la funcionalidad cantidad de caminos entre par de vértices, el resultado que mostraba era el grado negativo	5ta	Resuelta
No estaban activadas las funcionalidades “Abrir archivo” y “Salir del Sistema”.	8va	Resuelta

4.3 CONCLUSIONES

En este capítulo quedó conformado el modelo de implementación, basada en el diseño descrito, estableciendo casos de pruebas que le permite al usuario validar el funcionamiento de las funcionalidades. Se obtuvieron los artefactos propuestos por la metodología XP para la etapa de producción, demostrando el desarrollo satisfactorio del *software* en el período de tiempo establecido. Se definieron los estándares de codificación puestos en práctica en la implementación Además, se pudo constatar que el *software* no presentaba errores al ser satisfactorias las pruebas aceptación del cliente arrojando excelentes resultados por cada una de las pruebas.

CONCLUSIONES GENERALES

Concluidos la investigación y todas las etapas del proceso de desarrollo del *software* se pueden arribar a las siguientes conclusiones:

- Los análisis de las herramientas matemáticas concluyeron que no satisfacían la situación problemática y problema a resolver. Con esto se comienza el proceso de desarrollo de una nueva aplicación que lo resuelva.
- El estudio de la Teoría de Grafos con el objetivo de aplicar los conocimientos a la herramienta proporcionaron un aumento y profundización por parte de los autores en el tema.
- La selección de las herramientas y la metodología de *software* establecieron las piedras angulares del proceso de desarrollo. Permitiendo obtener los útiles que más se adecuaran a la situación, con las cuales realizar un óptimo trabajo.
- El proceso de desarrollo permitió obtener primeramente una planificación de las actividades y tareas de los desarrolladores, el diseño arquitectónico brindó una arquitectura adecuada al tipo de aplicación y las características principales, dando paso a la implementación y finalmente comprobando la calidad y funcionamiento con las pruebas unitarias y de aceptación.
- Se obtuvo la última versión del producto, resultado esperado que permitió cumplir con el objetivo general, que los estudiantes de primer año que cursan la asignatura Matemática Discreta y los profesores que imparten esta asignatura contarán con una herramienta que servirá de apoyo al proceso de enseñanza-aprendizaje, permitiendo fomentar, ejercitar y consolidar los conocimientos acerca de la Teoría de Grafos.

RECOMENDACIONES

Terminada la aplicación cumpliendo con el objetivo general se recomienda:

- ✓ Diseñar una ayuda integrada a la aplicación que permita un acercamiento y manejo de la aplicación.
- ✓ Que el asistente sea utilizado en la asignatura Matemática Discreta o los interesados para comprobar la influencia del mismo en el proceso de enseñanza aprendizaje.
- ✓ Ampliar la aplicación con módulos de otros temas de la Matemática Discreta para formar un asistente matemático que ayude a todos los temas de la asignatura.

REFERENCIAS BIBLIOGRÁFICAS

1. Francisco Alberto Fernández Nodarse, S.L.M.y.J.S.I.R., *Experiencias en la estructuración de clases de matemáticas empleando asistentes matemáticos y colección de tutoriales hipermediales*.
2. Johnsonbaugh, R., *Matematicas Discretas*. 4ta ed. Vol. 2. 1999, México.
3. Gimbert, J., *Los grafos como modelos matemáticos: ejemplos y aplicaciones*. 1999: Universidad Autónoma de Guerrero, México.
4. Rodríguez, J.M., *"COLORACIÓN DE GRAFOS"*, in *Departamento de Informática*. 2010, Universidad Politécnica de Madrid: Madrid.
5. Johnsonbaugh, R., *Matemáticas Discretas*. 4ta ed. Vol. 2. 2002.
6. *Sitio Oficial de MathWorks.inc*. 2012 [cited 2012 Diciembre]; Available from: http://www.mathworks.com/help/matlab/functionlist.html)
7. Moler, C., *Ayuda de Matlab*. 1984.
8. Richard A. Becker, J.M.C.a.A.R.W., *The New S Language*. 1988, Chapman & Hall, New York.
9. Pressman, R.S., *Ingeniería de Software, un enfoque práctico*. Quinta edición ed. 2002: McGraw-Hill Companies.
10. López Quesada, P.J.A. *Fundamentos de Ingeniería del Software*. . 2004-2005; Available from: http://dis.um.es/~lopezquesada/documentos/FIS_0405/Tema7.ppt.
11. *Definition.org*. Available from: http://www.definicion.org/lenguaje-de-programacion. .
12. Marañón, D.G.Á. *Departamento de Tratamiento de la Información y Codificación*. 1997-1999; Available from: http://www.iec.csic.es/criptonomicon/java/quesjava.html.
13. *PC Magazine Encyclopedia*. 2013; Available from: http://www.pcmag.com/encyclopedia/term/44707/ide.
14. *junit.org*. Available from: http://www.junit.org
15. *JGraph User Manual for JGraph Version 5.13.0.0*. 2009.
16. García, J., *'UML: Diagramas UML. ¿Qué es UML?'*. 2005.
17. Davis, R., *Business Process Modeling with ARIS: A Practical Guide*. 4ta Edition, 2005 ed. 2001: Springer, London. 531.
18. White, S.A., *'Business Process Modeling Notation'*. 2003. 189.
19. O'Reilly & Associates, I., *UML en Resumen: Una Rápida Referencia de Escritorio*. 1998.
20. Fowler, M., *UML Distilled: A Brief Guide to the Standard Object Modeling Language*.
21. Dictionary.com. 2013; Available from: http://dictionary.reference.com/browse/computer+aided+software+engineering.
22. Pressman, R.S., *Ingeniería del Software*. Sexta Edición ed.

23. Erich Gamma, R.H., Ralph Johnson, John Vlissides. , *Design Patterns (Elements of Reusable Object-Oriented Software)*.
24. Jeffries, R., Anderson, A., Hendrickson, C, *Extreme Programming Installed*. 2001: Addison-Wesley.