Universidad de las Ciencias Informáticas.
Facultad # 9.

"Desarrollo de una Biblioteca de Estructuras de Datos Avanzadas (listas, pilas, colas, tablas hash y DCEL)".

Trabajo de diploma presentado por:

❖ Yulaine Arias Guerra.

Para optar por el título de Ingeniero en Ciencias Informáticas.

Dirigida por:

Tutor:

❖ Lic. José Ángel Lago Graverán.

Co-tutor(es):

❖ Ing. Alexey Díaz Domínguez.

❖ Ing. Yancy Martínez Pérez.

❖ Lic. Yusnier Valle Martínez.

Ciudad de la Habana, Julio del 2008.

"Año 50 de la Revolución".

"...El acceso al conocimiento y la cultura no significa por sí solo la adquisición de principios éticos; pero sin conocimiento y cultura no se puede acceder a la ética. Sin ambos no hay ni puede haber igualdad ni libertad. Sin educación y sin cultura no hay ni puede haber democracia".

"...La propia vida material futura de nuestro pueblo tendrá como base los conocimientos y la cultura. Con ellos nuestro país, en medio de una colosal crisis económica mundial, avanza en distintos frentes."

Fidel Castro (Discurso de sesión de clausura del Congreso Pedagogía 2003, en el teatro "Carlos Marx", el 7 febrero 2003)

Dedicatoria.

Dedicado a:

Fidel Castro Ruz, promotor principal del Proyecto Futuro, que nos ha aportado los conocimientos necesarios para servir a nuestra Patria…

Mis padres (Irma Guerra González y David Arias Pupo) y a mi hermanita (Yusel Arias Guerra) con todo mi amor y cariño por brindarme en todo momento su confianza y apoyo incondicional y ver realizado este sueño maravilloso…

Y en especial dedicado a mi Bebé, que sin su amor todo esto no hubiese tenido sentido, y aunque estemos lejos el uno del otro nuestro amor continúa vivo,.. Gracias por amarme a mí tanto como yo a ti…Te Amo.

Yula.

Agradecimientos.

» El presente trabajo es simplemente un camino para empezar y ver cristalizado nuestro sueño de ser unos profesionales de excelencia, y esto como es bien sabido por todos nosotros, se lo debemos en gran parte a la Revolución Cubana y a nuestro líder Fidel Castro, por darnos la oportunidad de convertirnos en profesionales, y por su visión futurista, para el beneficio y desarrollo de nuestro pueblo.

» A la Universidad de las Ciencias Informáticas, por ser mi casa y mi mundo durante cinco inolvidables años, donde nos formamos como hombres del presente y del futuro.

» A mi tutor José Ángel Lago Graverán por la dedicación brindada para que este trabajo saliera adelante, por ser tan exigente y preocupado, por sus sugerencias, en fin por su gran apoyo a lo largo de todos estos meses de trabajo.

» A mis padres Irma Guerra González y David Arias Pupo, a quienes les debo todo y es gracias a ellos que estoy aquí. Se que esto es muy poco por todo sus sacrificios, pero espero que este trabajo les devuelva en parte todo lo que me han brindado con tanto amor. Así mismo hago extensivo el agradecimiento a mi hermanita Yusel Arias Guerra que está estudiando también en esta universidad y que dentro de 3 años se graduará. Y en general a mi familia.

» A todas aquellas personas que en la UCI, me brindaron su amistad y ayuda siempre que las necesité: José Alberto Fernández Gómez, Alexey Díaz Domínguez, Yancy Martínez, Jorge Infante, Yaricel Alcántara Laurencio, Yulien Figueredo Guzmán, Víctor Frank Molina, Romanuel…

» A mi tribunal integrado por: Miriet Espinosa Ojeda, Aniuska Grotestan Columbie y Maykel López Oliva, por todas las sugerencias brindadas durante los cortes de tesis.

» A Dayana, Armando y Ayasel, por el gran esfuerzo que hicieron por imprimir la portada y contraportada de la tesis…

» También a todos aquellos que fueron siempre fuente de fuerza y alegría y me impulsaron a seguir adelante hasta alcanzar la meta.

» A todos los que alguna vez me preguntaron: ¿y la tesis como va?

A todos, muchas gracias**…**

Yulaine Arias Guerra

Declaración de Autoría:

Declaro que soy la única autora de este trabajo y autorizo a la Facultad 9 y en general a la Universidad de las Ciencias Informáticas para que hagan el uso que consideren necesario con el mismo en su beneficio.

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

Yulaine Arias Guerra

_____

Lic. José Ángel Lago Graverán

_____

Datos de contacto:

Síntesis del Tutor:

Lic. José Ángel Lago Graverán:

Graduado con Título de Oro de Licenciado en Ciencias de la Computación, Universidad de la Habana año 2005. Durante el período de universidad integró el grupo UHSIS, en este grupo participó como desarrollador de cuatro productos, los cuales están registrados. Fue integrante del grupo de Geometría Computacional de la facultad de Matemática-Computación. Ha participado en varios eventos nacionales e internacionales, en algunos de ellos ha publicado trabajos. Actualmente trabaja como profesor en la UCI, en la cual ha impartido Matemática Numérica y Gráfico por Computadoras, además de estar vinculado a la producción.

Síntesis de los Co-tutores:

Lic. Yusnier Valle Martínez.
Graduado de Nivel Superior (Licenciatura) en la especialidad de Ciencia de la Computación el 13 de Julio de 2004 en la Universidad de la Habana (UH). Actualmente tiene la categoría docente de Profesor Instructor. Se desarrolla como Arquitecto de Software del Grupo Sistemas de Información Geográfica (GSIG), en la Facultad 9, en la Universidad de Ciencias Informáticas. Se desarrolla como arquitecto en el proyecto "Conceptualización de Soluciones Informáticas en Refinerías" [Información de Negocio de Refinación de Petróleo (áreas de Mantenimiento, Operaciones y Ambiente e Higiene Ocupacional), Ingeniería de Software (estudio de negocio)], con la empresa Petróleos de Venezuela S.A. (PDVSA). Es líder del proyecto MCP (modelación de negocio, asimilación de herramientas de desarrollo en software libre), con la empresa Petróleos de Venezuela S.A. (PDVSA). Labora también como Profesor Instructor de las asignaturas: Sistemas Operativos y Seguridad Informática a la carrera de Ingeniería en Informática en su Curso Regular Diurno (CRD) en la Universidad de las Ciencias Informáticas (UCI).

Datos de contacto:

Ing. Yancy Martínez Pérez:

Graduado como Ingeniero en Ciencias Informáticas en la UCI, en el curso 2005-2006. Actualmente se encuentra en su segundo año de adiestramiento. Tiene experiencias como tutor de trabajos de diploma, organizando, orientando y guiando el desarrollo de 3 ejercicios de culminación de estudios en el curso 2006–2007. Ha impartido las asignaturas de Ingeniería y Gestión de Software. Se desempeña como Analista Principal en el proyecto SACGIR.

Opiniones y Avales:

» **Opinión del Tutor:**

Título: Desarrollo de una Biblioteca de Estructura de Datos Avanzadas (Listas, Pilas, Colas, Tablas Hash y DCEL).

Autor: Yulaine Arias Guerra.

El tutor del presente Trabajo de Diploma considera que durante su ejecución la estudiante mostró las cualidades que a continuación se detallan:

- Asumió con un alto grado de responsabilidad todas las tareas trazadas.
- Su dedicación y esfuerzo permitió que se alcanzara el objetivo general de la investigación.
- En la documentación y herramienta obtenidas como resultado se puede apreciar originalidad, creatividad y un alto nivel de responsabilidad por parte de la autora.
- De forma general el Trabajo de Diploma presenta una alta calidad científico-técnica. Los resultados obtenidos presentan un elevado nivel de rigurosidad y formalismo, lo cual corrobora que el objetivo propuesto al inicio de la investigación fue alcanzado.

Por todo lo anteriormente expresado se considera que la estudiante está apta para ejercer como Ingeniero en Ciencias Informáticas, y se propone la calificación máxima de 5 puntos.

José Angel Lago Graverán

_____ _____

Firma del Tutor Fecha

Resumen:

El entorno donde coexiste el problema se da lugar en la Universidad de las Ciencias Informáticas donde es necesario utilizar estructuras de datos avanzadas para la solución de determinados problemas de la asignatura de Programación y en el desarrollo de los diferentes proyectos productivos. Pero desafortunadamente sobre este tipo de estructuras no se ha implementado aún una biblioteca que contenga agrupada una gran variedad de ellas y cuando es necesario utilizarlas, hay que implementar cada una por separado o usar las que existen aisladas.

Por tal motivo la tesis tiene como propósito el desarrollo de una Biblioteca de Estructuras de Datos Avanzadas elaborada en una plataforma de software libre, la cual utiliza UML como notación y sigue los pasos que propone el Proceso Unificado de Desarrollo de Software, utilizándose además otras herramientas complementarias.

El uso de estas estructuras de datos que se agrupan para conformar una biblioteca es una tendencia creciente del desarrollo del software con el fin de utilizarlas en la elaboración de diversas aplicaciones. Con el uso de esta biblioteca se espera optimizar el tiempo en la elaboración de los programas.

PALABRAS CLAVES:

- ✓ Biblioteca.
- ✓ Estructura de Datos Avanzadas.
- ✓ Software Libre.

Abstract:

The environment in which the problem coexists takes place in the University of Informatics Sciences where it becomes necessary to use advanced data structures in the solution of certain problems in the Programming subject and in the development of the different productive projects. But unfortunately, on this type of structures it has not been implemented yet a library that contains together a great variety of them, that's why when it is necessary to use them, they have to be implemented each one for separate or using those that are isolated.

For that reason this graduate paper has as a purpose the development of a Library of Advanced Data Structures elaborated on a free software platform, which uses UML like notation and it follows the steps proposed by the Unified Process of Software Development, being also used another complementary tools.

The use of these structures of data that are grouped to make up a library is a growing tendency of the software development with the purpose of using them in the elaboration of different applications. With the use of this library it is hoped to optimize the time in the elaboration of the programs.

KEY WORDS:

- ✓ Library.
- ✓ Advanced Data Structure.
- ✓ Free Software.

Índice.

Índice de Figuras.

Índice de Tablas.

Introducción.

El desarrollo del software es considerado como uno de los problemas técnicos cruciales en la actualidad y desde su aparición ha ganado gran complejidad.

El bloqueo económico, político y social impuesto a Cuba limita, entorpece y encarece la adquisición del software legal necesario para el desarrollo de la industria cubana del software, por tal motivo el Software Libre se abre como solución a esta problemática.

Sin duda alguna el Software Libre es sustentable para Cuba, por esto su aplicación como plataforma informática de trabajo adquiere una relevante importancia que puede verse reflejada en el ámbito económico, político y tecnológico.

Actualmente la Universidad de las Ciencias Informáticas lleva a cabo un significativo proceso de migración hacia el Software Libre, ya que muchas de las herramientas que se utilizan son propietarias y hay que pagar grandes sumas de dinero por la adquisición de sus licencias. La universidad juega un papel importantísimo en la producción de software para las diferentes instituciones y empresas, ya sean nacionales o extranjeras, como es el caso de PDVSA[1]; en la misma se utilizan junto con las tecnologías y herramientas, diferentes tipos de estructuras de datos para el manejo con grandes cantidades de información.

A nivel mundial existen bibliotecas de estructuras de datos implementadas en lenguajes de programación como C++, C# y Java que poseen estructuras básicas, tal es el caso de los registros que además son estructuras de datos estáticas, las cuales tienen como marcada limitación que el espacio ocupado en la memoria de la computadora se define en tiempo de compilación y no puede ser modificado durante la ejecución del programa. Para solucionar este problema están las estructuras de datos dinámicas, como las listas, las pilas, las colas, etc., que pueden almacenar grandes cantidades de elementos de diferentes tipos.

[1] Petróleos de Venezuela Sociedad Anónima

En la carrera de Ingeniería Informática o similares, debido a su importancia, es prácticamente obligatorio el estudio de las estructuras de datos pero en su versión más simple (listas, pilas, colas, árboles y grafos sencillos), lo cual sirve para sumergirse en esta importante materia y darle solución a determinados problemas.

Para las estructuras de datos se han realizado varios trabajos en el mundo y en Cuba, principalmente en la Universidad de las Ciencias Informáticas, pero lo que se ha desarrollado respecto a este tema en la universidad es para contribuir con el desarrollo de los diversos proyectos productivos o para resolver los problemas puntuales de la docencia, por ejemplo, que sirvan de apoyo en las clases y en las tareas finales de la asignatura de Programación o en la pruebas de nivel de la misma asignatura.

Luego, ¿Qué sucedería si los problemas a resolver, fueran tan complejos como representar terrenos o implementar algoritmos para Gráficos por Computadora, que necesitan manejar grandes cantidades de información en la memoria de la computadora? Para estos casos es necesario utilizar estructuras de datos avanzadas, que son óptimas para el trabajo con grandes volúmenes de datos, además minimizan los recursos de la computadora en cuanto a tiempo de ejecución y consumo de la memoria.

Pero desafortunadamente, según las investigaciones realizadas en la universidad, no se ha implementado aún una biblioteca que contenga agrupada una gran variedad de estructuras de datos y cuando hace falta utilizarlas hay que implementarlas por separado o usar las que se encuentran aisladas. Por tal motivo el **problema científico** es precisamente: “la poca disponibilidad de bibliotecas que contengan agrupadas una gran variedad de estructuras de datos avanzadas”, con el fin de ser usadas en cualquier solución que lo requiera.

El **objeto de estudio** de la investigación lo constituyen las *estructuras de datos*. El **campo de acción** es la parte del objeto de estudio que se va a investigar, que serían las Estructuras de Datos Avanzadas, específicamente: las Listas, las Pilas, las Colas, las Tablas Hash y DCEL.

El **objetivo general** es el desarrollo en una plataforma de Software Libre de una Biblioteca de Estructuras de Datos Avanzadas, que permita reducir en costo y tiempo la solución de diversos problemas, facilitando además la gestión y aprovechamiento de la memoria del sistema y brindando una mayor flexibilidad frente a los cambios que puedan existir.

Conjuntamente con la biblioteca se confecciona un documento de ayuda que puede ser utilizado como guía de estudio, aún cuando no se utilicen las implementaciones de la misma, ya que en el documento se detallan diversos aspectos de interés de cada una de las estructuras de datos avanzadas. Lo mencionado anteriormente permite tener un mejor conocimiento de estas estructuras y saber en un problema determinado cual es la más óptima para darle solución al mismo.

Para el desarrollo y la culminación satisfactoria del presente trabajo de diploma, las **tareas de la investigación** se desglosaron de la siguiente manera:

1. Estudiar el estado del arte.
2. Escoger las herramientas para el desarrollo de la biblioteca.
3. Realizar el análisis y el diseño de la biblioteca.
4. Elaborar el diagrama del diseño de las estructuras de datos.
5. Realizar las distintas implementaciones de las diferentes estructuras de datos.
6. Realizar las pruebas a la biblioteca.
7. Elaborar la documentación de la biblioteca.
8. Revisar el informe de la tesis y la presentación de la defensa.

El producto final tendría gran utilidad principalmente en la esfera productiva y docente de la universidad así como en cualquier parte del mundo, ya que como se ha planteado anteriormente, nunca se ha logrado agrupar a las variantes de implementación de estas estructuras de datos en una biblioteca, además la documentación que la acompaña puede emplearse como una herramienta independiente para el estudio de estas estructuras de datos avanzadas.

El hecho de que la Biblioteca de Estructuras de Datos Avanzadas estará implementada para una plataforma de Software Libre, anotará un punto más a favor en su desarrollo, y contará para su mejoramiento y perfección con la colaboración de todos los interesados.

Para alcanzar un mejor entendimiento y obtener la mayor cantidad de información, en la investigación se pusieron en práctica los siguientes **métodos científicos**:

- ***Métodos Teóricos:***
  - » Método Histórico-Lógico: este método sirve de apoyo para investigar si existen implementados en el país proyectos informáticos de este tipo, y en caso de existir, conocer cómo es su funcionamiento.
  - » Método de Modelación: este método se ve reflejado en la modelación de los diagramas que ayudarán a lograr un mejor entendimiento de la solución del problema.

- ***Métodos Empíricos:***
  - » Entrevistas: este método se manifiesta en la realización de las entrevistas a integrantes y a jefes de los diferentes proyectos productivos con el fin de investigar y obtener datos concretos acerca del uso de las estructuras de datos avanzadas en la universidad.
    Para consultar las entrevistas realizadas, remitirse al ANEXO 1.

Para realizar las entrevistas se seleccionó como **población** a los proyectos productivos que trabajan con Software Libre en la universidad, de los cuales se tomó una **muestra** de cuatro proyectos, arrojándose los resultados que se esperaban. Estos proyectos son:

- » Informatización de los Servicios de la Biblioteca Nacional.
- » Informatización de los Servicios de la Biblioteca UCI.

- Proyecto SCADA[2].
- Proyecto “Unicornio”.

La **técnica de muestreo** utilizada fue el Muestreo Intencional, debido a que se escogió intencionalmente la población a utilizar y de ella la muestra a la cual se le realizó la entrevista.

El presente informe está organizado por cuatro capítulos expuestos brevemente a continuación:

Capítulo 1 “Fundamentación teórica”: En este capítulo se analizan los principales conceptos asociados al dominio que respaldan el problema científico y que originan el estudio del estado del arte, además se explica de forma general el objeto de estudio donde se enmarca el problema, se presenta el dominio actual del problema y la situación problemática, también se analiza la existencia de otras posibles soluciones.

Capítulo 2 “Tendencias y tecnologías actuales a utilizar”: En este capítulo se aborda el tema relacionado con las tendencias y tecnologías actuales a utilizar para el desarrollo de la Biblioteca de Estructuras de Datos Avanzadas, se fundamenta la metodología de desarrollo de software empleada, el lenguaje de modelado, la herramienta CASE, el lenguaje de programación y otras herramientas de apoyo que de una forma u otra contribuyeron al desarrollo del trabajo.

Capítulo 3 “Presentación de la solución propuesta”: Este capítulo se centra en presentar la solución propuesta donde se exponen los elementos imprescindibles para una solución exitosa, por ejemplo: el modelo del domino, los requerimientos funcionales y no funcionales con los cuales el componente del sistema debe cumplir y una descripción general de los actores y de los casos de usos del sistema a automatizar.

[2] SCADA es el acrónimo correspondiente a *Supervisory Control And Data Acquisition*, cuya traducción literal sería: Adquisición de Datos y Control de Supervisión.

Capítulo 4 “Construcción de la solución propuesta”: En este capítulo se exponen los diagramas de clases correspondientes a la etapa del diseño organizado por paquetes, se planifica la implementación y se muestran las pruebas realizadas al sistema propuesto.

Finalmente se ofrecen otros apartados centrados en: las conclusiones obtenidas del desarrollo del trabajo, las recomendaciones, la recopilación de la bibliografía relacionada que contiene la totalidad del material utilizado para la confección del trabajo, los anexos que detallan temas que si bien no son centrales para la tesis sirven para profundizar en aspectos mencionados en esta, así como un glosario de términos y siglas.

Capítulo 1

Fundamentación Teórica.

1.1 Introducción.

En el presente capítulo se brinda una descripción de los principales conceptos asociados al dominio del problema, se realiza un estudio del estado del arte de las diferentes estructuras de datos. Se presenta el objeto de estudio donde se enmarca el problema y se ofrece una descripción general del mismo. Además se muestra el dominio actual del problema, la situación problemática y se analiza la existencia de otras posibles soluciones al problema científico planteado.

1.2 Conceptos asociados al dominio del problema.

Los sistemas o métodos de organización de los datos que permiten un almacenamiento eficiente de la información en la memoria de la computadora son conocidos como *estructuras de datos*. Estos métodos de organización constituyen las piezas básicas para la construcción de algoritmos complejos y permiten implementarlos de manera eficiente.

En este capítulo se introduce primeramente con el concepto de Tipo de Dato Abstracto (TDA) para luego abordar aquellos que representan colecciones de elementos de un mismo tipo, usando una disposición lineal para almacenarlos. Se abordarán específicamente los TDA: Lista, Pila, Cola, Tabla Hash y DCEL[3] en algunas de sus variantes de implementación.

[3] Lista de Lados Doblemente Enlazados.

Un Tipo de Dato Abstracto (TDA) es un modelo matemático compuesto por una colección de operaciones definidas sobre un conjunto de datos para el modelo. (1)

1.2.1 ¿Qué es una Lista?

Una lista se define como una n-tupla de elementos (donde l_i es el i-ésimo elemento de la lista) ordenados de forma consecutiva, o sea, el elemento l_i precede al elemento l_{i+1}: $L = (l_1, l_2, \ldots, l_n)$. Si la lista contiene cero elementos se denomina lista vacía. (1)

Operaciones básicas del TDA Lista**:**

a. **Vacía ( )**, devuelve verdadero si la longitud de la lista es cero. No se modifica la lista.
b. **Longitud ( )**, devuelve $|L|$, la longitud de la lista (cantidad de elementos). No se modifica la lista.
c. **Obtener (i)**, devuelve el i-ésimo elemento de la lista (el que se encuentra en la posición ***i***), si la posición ***i*** no existe se dispara una excepción. No se modifica la lista.
d. **Adicionar(x)**, adiciona el elemento ***x*** en la cola de la lista, haciendo que la longitud de la lista se incremente en uno. Si la operación no tiene éxito, se dispara una excepción.
e. **Insertar(x, i)**, inserta el elemento ***x*** en la posición ***i***, haciendo que los elementos $l_i, l_{i+1}, \ldots, l_n$ pasen a ser los elementos $l_{i+1}, l_{i+2}, \ldots, l_{n+1}$ y se incrementa en uno la longitud de la lista. Si la operación no tiene éxito, se dispara una excepción.
f. **Eliminar(i)**, elimina el elemento almacenado en la posición ***i*** de la lista, haciendo que los elementos $l_{i+1}, l_{i+2}, \ldots, l_n$ pasen a ser los elementos $l_i, l_{i+1}, \ldots, l_{n-1}$, esta operación disminuye en uno la longitud de la lista. Si la posición no existe se dispara una excepción.

» TDA Lista utilizando Arreglos:

Un TDA Lista puede utilizar como estructura de datos a los arreglos lineales, esta clase se deriva de la clase Lista. Los arreglos lineales son convenientes fundamentalmente por la simplicidad de su uso, además el tiempo de acceso a los elementos individuales de un arreglo es fijo para todas las operaciones, lo que resulta muy eficiente. (1)

Sin embargo, las operaciones de inserción y borrado de los elementos en arreglos son ineficientes, puesto que para insertar un elemento en la parte media del arreglo es necesario mover todos los elementos que se encuentren detrás de él para hacer espacio y al borrar un elemento es preciso mover todos los elementos para ocupar el espacio desocupado.

» ¿Qué es una Lista Enlazada?

Una lista enlazada es una secuencia de nodos enlazados donde el orden de los elementos está determinado por un campo de enlace (referencia) explícito en cada elemento. Esta definición remite al concepto de nodo.

Un nodo se compone de un campo con el tipo de dato de los elementos de la lista y por uno o más campos que son referencias a otros nodos. (2)

Figura 1.1 Representación gráfica de un Nodo.

» ¿Qué es una Lista Simplemente Enlazada?

Una lista simplemente enlazada se compone por nodos de enlace simple, es decir, por nodos que tienen solamente un campo *Dirección*. Un nodo sólo está enlazado con el nodo que le sigue. (2)

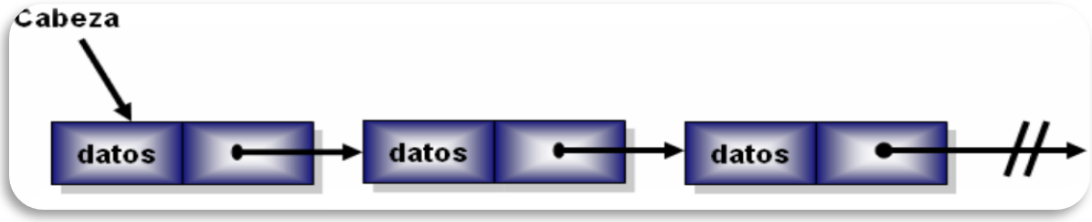


Figura 1.2 Representación gráfica de la Lista Simplemente Enlazada.

» ¿Qué es una Lista Circular Simplemente Enlazada?

Una lista circular simplemente enlazada es una lista lineal en la que el último elemento se enlaza con el primero. Cada nodo tiene un enlace similar al de las listas simplemente enlazadas, excepto que el enlace siguiente del último nodo, apunta al primero.

En las listas circulares no puede hablarse ni de "primero" ni de "último", porque cualquier nodo puede ser el nodo de entrada y de salida. Al igual que en una lista simplemente enlazada, los nuevos nodos pueden ser solo eficientemente insertados después de uno que se tenga referenciado, por esta razón, es usual quedarse con una referencia solamente al último elemento en una lista circular simplemente enlazada, esto permite rápidas inserciones al principio y accesos al primer nodo desde el puntero del último nodo. Además es posible acceder a cualquier elemento de la lista desde cualquier punto dado. La figura 1.3 muestra la representación gráfica de una Lista Circular Simplemente Enlazada.

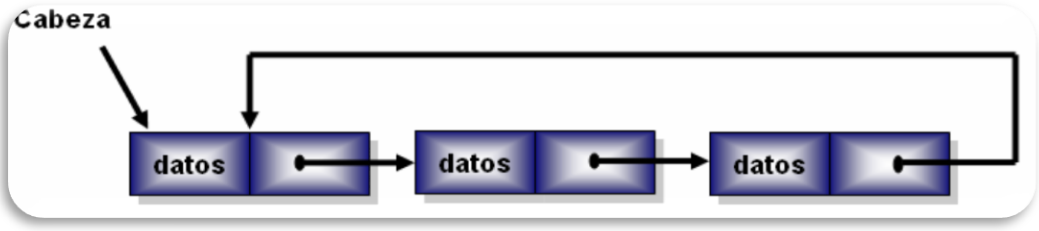


Figura 1.3 Representación gráfica de la Lista Circular Simplemente Enlazada.

» ¿Qué es una Lista Doblemente Enlazada?

Una lista doblemente enlazada es una lista lineal que se compone por nodos de enlace doble, es decir, por nodos que tienen dos campos: *Dirección1* y *Dirección2*. Un nodo está enlazado con el nodo que le sigue y con el nodo que inmediatamente le antecede. Esto permite recorrer la lista en cualquier dirección. (2)

Figura 1.4 Representación gráfica de la Lista Doblemente Enlazada.

» ¿Qué es una Lista Circular Doblemente Enlazada?

Una lista circular doblemente enlazada es una lista lineal en la que cada elemento tiene dos enlaces, similares a los de la lista doblemente enlazada, excepto que el enlace anterior del primer nodo apunta al último y el enlace siguiente del último nodo, apunta al primero.

Al igual que una lista doblemente enlazada, en la circular doble las inserciones y las eliminaciones pueden realizarse desde cualquier punto con acceso a algún nodo cercano. Aunque estructuralmente una lista circular doblemente enlazada no tiene ni principio ni final. Un puntero de acceso externo puede establecer el nodo apuntado que está en la cabeza o al nodo cola y así mantener el orden como en una lista doblemente enlazada. La figura 1.5 muestra la representación gráfica de una Lista Circular Doblemente Enlazada.

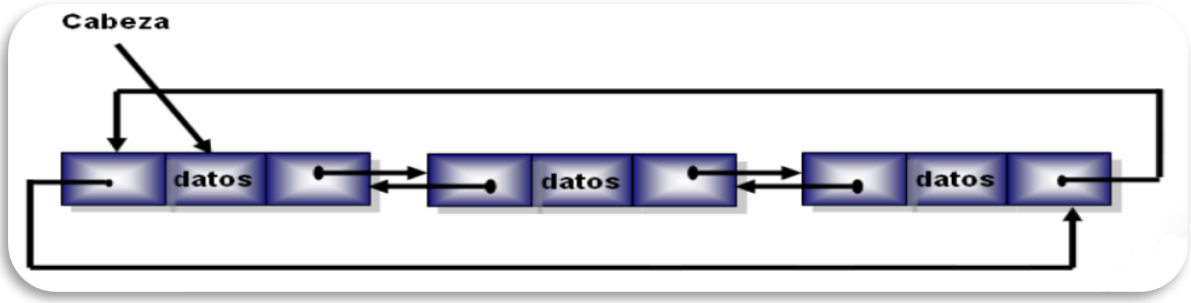


Figura 1.5 Representación gráfica de la Lista Circular Doblemente Enlazada.

1.2.2 ¿Qué es una Pila?

Una pila (*stack* en inglés) es una estructura sencilla, mucho más simple que la lista y se puede definir como una colección ordenada de elementos $S = (S_1, S_2, \ldots, S_n)$ donde se adicionan y eliminan por un mismo extremo conocido como *tope*. Se dice que S_1 es el elemento del fondo de la pila y S_n es el elemento que se encuentra en el tope. Se les conoce como estructuras LIFO[4] debido al orden en que se adicionan y extraen los elementos en la misma. (3)

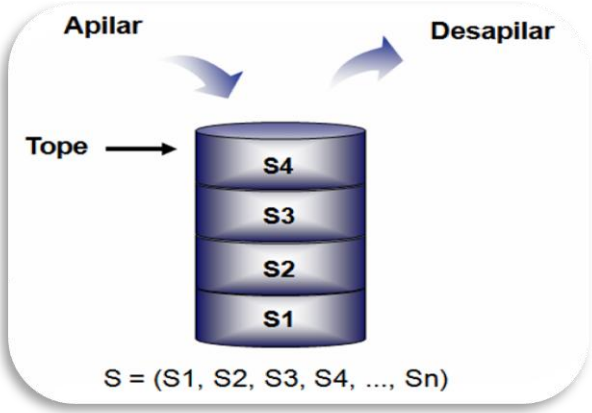


Figura 1.6 Representación gráfica de una Pila.

[4] LIFO es el acrónimo correspondiente a *Last In First Out*, cuya traducción literal sería: último en entrar, primero en salir.

La interfaz de este TDA provee las siguientes operaciones:

a. **Vacía ( )**, devuelve verdadero si la pila está vacía.
b. **Tope ( )**, devuelve el elemento que se encuentra en el tope de la pila, se conoce también con el nombre de *cima*. Si es llamado con la pila vacía se lanza una excepción.
c. **Apilar (x)**, coloca a ***x*** en el tope de la pila, se conoce también con el nombre de adicionar.
d. **Desapilar ( )**, elimina el elemento que se encuentra en el tope de la pila, se conoce también con el nombre de extraer. Si es llamado con la pila vacía se lanza una excepción.

» Pila utilizando arreglos.

Una pila puede ser implementada usando un arreglo el cual se irá llenando en la forma usual, conjuntamente con este se tiene un atributo que almacena el índice de la última posición utilizada. (3).

Si se implementa una pila utilizando un arreglo se debe especificar primero el tamaño máximo de la pila y además definir una operación que indique que la misma está llena y en caso de que se desee adicionar un elemento sería necesario crear otro arreglo mayor que el anterior, copiar todos los elementos de la pila en este arreglo y finalmente liberar la memoria ocupada por el arreglo inicial.

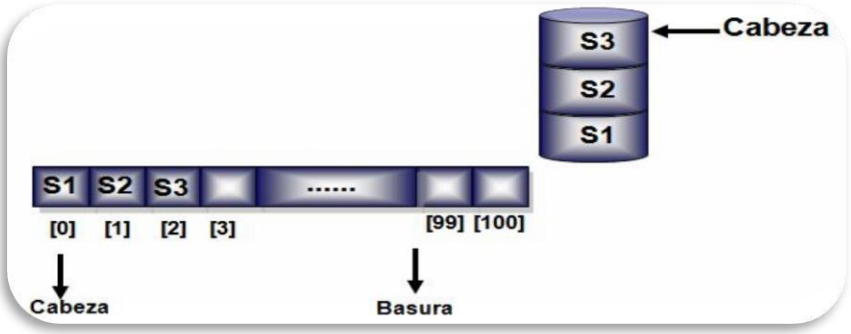


Figura 1.7 Representación gráfica de una Pila utilizando arreglos.

» Pila utilizando listas enlazadas.

Otra implementación de la pila puede ser utilizando listas enlazadas, donde los nodos de la lista simplemente enlazada se emplean para almacenar la información de la pila. En este caso no existe el problema de tener que fijar el tamaño máximo de la pila pues la capacidad de la pila está acotada por la cantidad de memoria de la computadora y la operación que indica que la misma está llena no tiene sentido. (3)

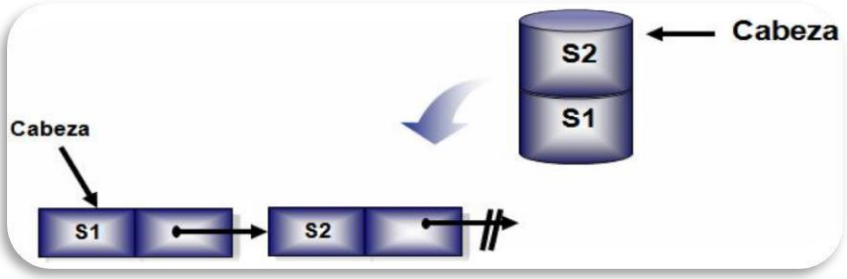


Figura 1.8 Representación gráfica de una Pila utilizando listas enlazadas.

1.2.3 ¿Qué es una Cola?

Una cola (*queue* en inglés) es una estructura de datos caracterizada por ser una lista ordenada de elementos $Q = (Q_1, Q_2, \ldots, Q_n)$, en la que la operación de adición se realiza por un extremo, llamado *cola*, y la operación de extracción por el otro, llamado *cabeza* y tiene un comportamiento de tipo FIFO[5] por el modo de acceso a sus elementos. (3)

[5] FIFO es el acrónimo correspondiente a *First In First Out,* cuya traducción literal sería: primero en entrar, primero en salir.

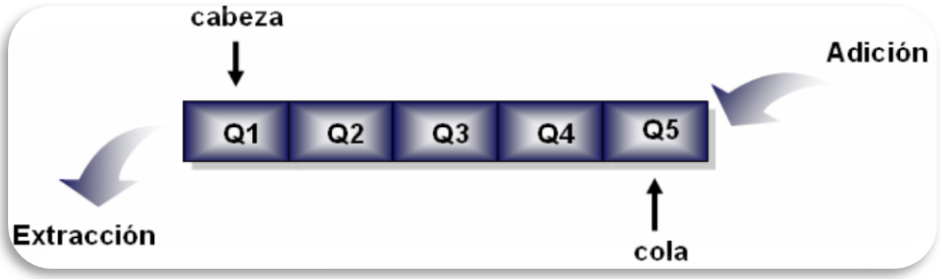


Figura 1.9 Representación gráfica de una Cola.

Operaciones básicas de la Cola:

a. **Vacía ( )**, devuelve verdadero si la cola está vacía.
b. **Frente ( )**, devuelve el elemento que se encuentra en la primera posición de la cola. Se dispara una excepción cuando la cola está vacía.
c. **Fondo ( )**, devuelve el elemento que se encuentra en la última posición de la cola. Se dispara una excepción cuando la cola está vacía.
d. **Adicionar (x)**, coloca a ***x*** en la cola después del último elemento de la misma.
e. **Extraer ( )**, elimina el elemento que se encuentra en la cabeza de la cola, si está vacía se dispara una excepción.

» ¿Qué es una Cola de Prioridad?

Una cola de prioridad es una cola cuyos elementos tienen asociado una prioridad y se atienden en el orden indicado por la misma, de forma que el orden en que los elementos son procesados sigue las siguientes reglas:

- El elemento con mayor prioridad es procesado primero.
- Si varios elementos tienen la misma prioridad, se atenderán en dependencia del orden en que fueron adicionados. Hay dos formas de implementación:

1. A cada nodo añadirle un campo con su prioridad. Resulta conveniente mantener la cola ordenada por el orden de prioridad.

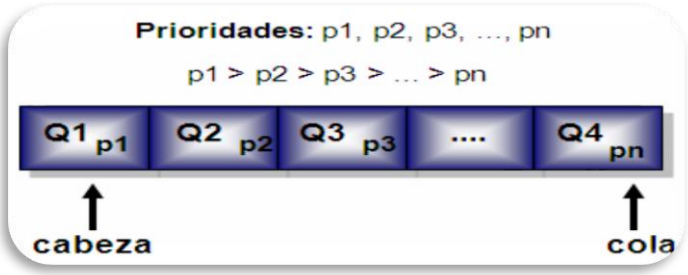


Figura 1.10 Representación gráfica de una Cola de Prioridad.

2. Crear tantas colas como prioridades exista y almacenar a cada elemento en la cola correspondiente.

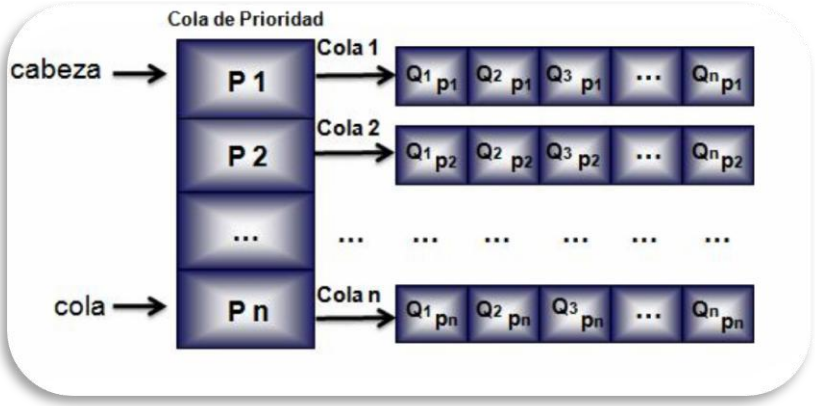


Figura 1.11 Representación gráfica de una Cola de Prioridad con varias Colas asociadas a cada prioridad.

Tipos de Colas de Prioridad:

- ✓ *Colas de prioridad con ordenamiento ascendente:* en ellas los elementos se insertan de forma arbitraria, pero a la hora de extraerlos, se extrae el elemento de menor prioridad.

- ✓ *Colas de prioridad con ordenamiento descendente:* son iguales que las colas de prioridad con ordenamiento ascendente, pero al extraer el elemento se extrae el de mayor prioridad.

Operaciones de las colas de prioridad (en cada una de estas operaciones, ***P*** se supone que es una cola de prioridad arbitraria):

a. **Insertar (P, x),** añade el elemento ***x*** a ***P***.
b. **Encontrar_Min (P),** devuelve elemento de ***P*** con la prioridad más alta (menor valor de clave). Si ***P*** está vacía esta operación produce un error.
c. **Eliminar_Min (P)**, quita y devuelve el elemento de ***P*** con la prioridad más alta (menor valor de clave). Si ***P*** está vacía esta operación produce un error.(4)

» ¿Qué es una Cola de amigos?

Una Cola de Amigos se comporta de manera similar a una Cola de Prioridad, con la particularidad de que en la misma para insertar un elemento se busca el primer elemento amigo partiendo desde el frente de la cola y se coloca delante él.

- ➢ Colas de Prioridad y de Amigos utilizando arreglos.

La implementación más sencilla que puede llevarse a cabo es utilizar un arreglo lineal. Al igual que en la lista, a estas operaciones se les pueden agregar otras. Por ejemplo si se implementa la cola utilizando arreglos se puede añadir una operación que indique cuando la misma está llena. En caso de que se quisiera adicionar nuevos elementos habría que reservar más espacio en la memoria de la computadora.(3)

- ➢ Colas de Prioridad y de Amigos utilizando listas enlazadas.

Al igual que en la pila, las implementaciones de una cola pueden ser obtenidas utilizando algunas implementaciones del TDA Lista, así como con la utilización de nodos enlazados, para esta variante la clase Cola puede utilizar un nodo *frente* y un nodo *fondo*.

1.2.4 ¿Qué es una Tabla Hash?

Una tabla hash, mapa hash o tabla de dispersión es una estructura de datos que está formada por las combinaciones de *llaves* o *claves* con *valores* organizados en "sectores de almacenamiento", para permitir realizar una búsqueda rápida. Constituye un TDA especial para la manipulación y almacenamiento de la información en la memoria secundaria de la computadora.

Una tabla hash se construye con tres elementos básicos.

- Una *estructura de acceso directo,* como un arreglo, capaz de almacenar ***N*** cantidad de elementos.
- Una *función de dispersión* que permita a partir de la clave obtener el índice donde estará almacenado el dato asociado a esa clave y cuyo dominio sea el espacio de claves y su imagen (o rango) los números naturales.
- Una *función de resolución de colisiones*. (5)

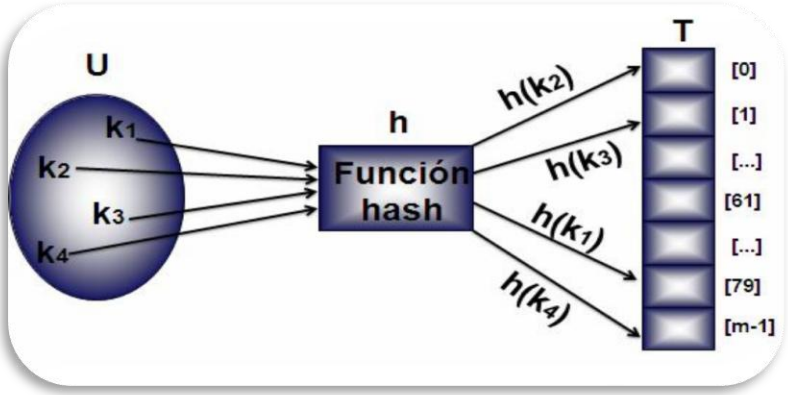


Figura 1.12 Representación gráfica de una Tabla Hash.

La operación principal que soporta de manera eficiente es la *búsqueda,* que permite el acceso a los elementos (por ejemplo: teléfono y dirección) almacenados a partir de una clave generada (por ejemplo: usando el nombre o número de cuenta). Funciona transformando a través de una función de dispersión, la clave en un índice, que no es más *que* un número que la tabla hash utiliza para localizar el valor deseado.

Las operaciones básicas implementadas en las Tablas Hash son:

- **Insertar (clave, elemento)**, inserta un elemento en un índice de la tabla hash generado por una función de dispersión dado una clave.
- **Buscar (clave)**, busca el elemento que se encuentra almacenado en el índice de la tabla hash generado por una función de dispersión dado una clave.
- **Eliminar (clave)**, elimina el elemento que se encuentra almacenado en el índice de la tabla hash generado por una función de dispersión dado una clave.

Es inevitable que claves distintas lleguen a producir el mismo resultado de la función de dispersión, dando lugar a las *colisiones* y esto constituye un problema.

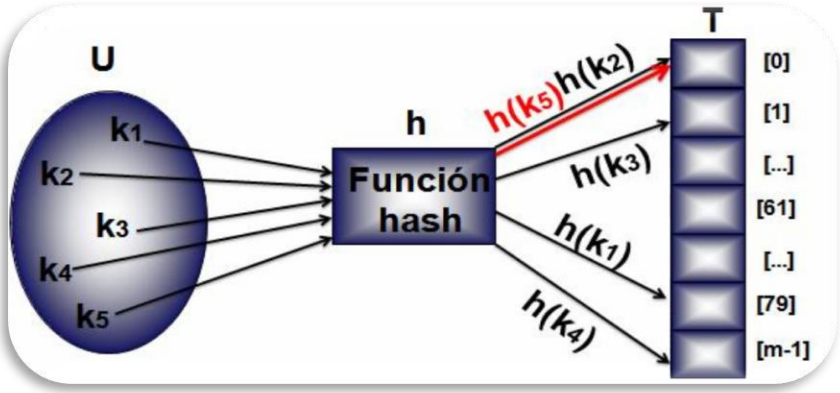


Figura 1.13 Representación gráfica de una Colisión.

Pero existen estrategias para resolverlo de mejor o peor manera:

- **Resolución de colisiones por dispersión Abierta (o externa):** este tipo de resolución de colisiones permite que se almacenen elementos en un espacio potencialmente ilimitado. Se utiliza una estructura de datos auxiliar para guardar los elementos cuya posición ya está ocupada por otros elementos, es decir, cada hueco contiene un puntero a una lista enlazada asignada dinámicamente que almacena todos los elementos que se dispersan en ese hueco.

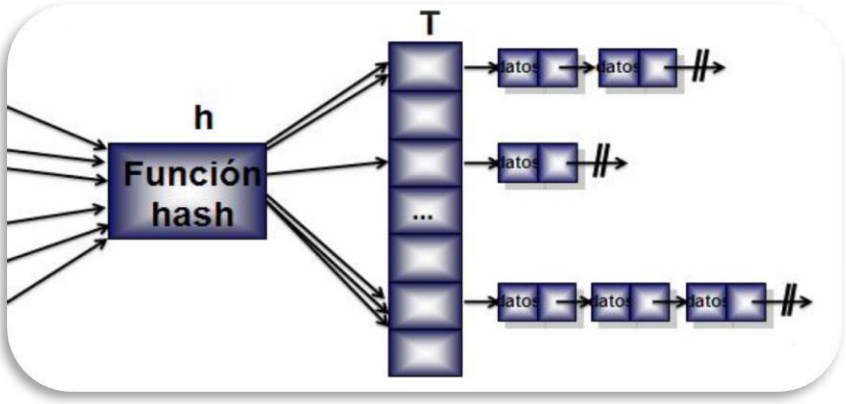


Figura 1.14 Representación gráfica de la resolución de colisiones por Dispersión Abierta en una Tabla Hash.

- **Resolución de colisiones por dispersión Cerrada (o interna):** este tipo de resolución de colisiones permite que todos los elementos se almacenen en la propia tabla. En este caso las colisiones se resuelven calculando una secuencia de huecos de dispersión. Esta secuencia es examinada o explorada sucesivamente hasta que se encuentra un hueco de la tabla dispersa vacío, en el caso de que se vaya a *Insertar* un elemento, o que se encuentre la clave deseada, en el caso de que lo se vaya a hacer es *Buscar* o *Eliminar* un elemento.(4)

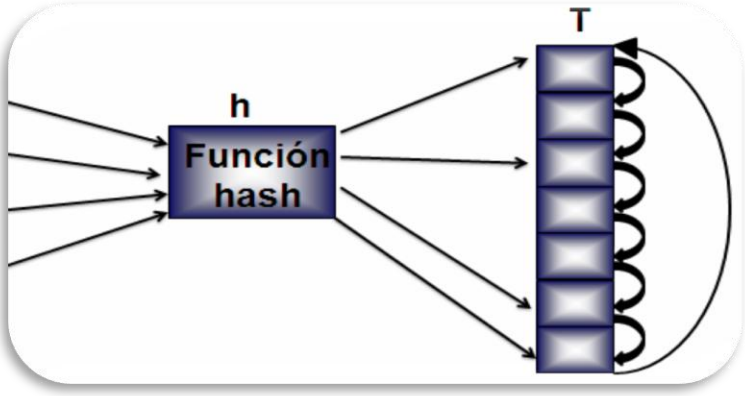


Figura 1.15 Representación gráfica de la resolución de colisiones por Dispersión Cerrada en una Tabla Hash.

1.2.5 ¿Qué es DCEL?

DCEL son las siglas inglesas de Lista de Lados Doblemente Enlazados. La estructura DCEL está formada por tres colecciones de registros:

- ✓ Lista de vértices: Un registro para un vértice ***v***, que almacena las coordenadas de ***v*** y un puntero a una arista arbitraria con origen en ***v***.

- ✓ Lista de aristas: Un registro para una arista ***a***, que almacena un puntero al vértice origen, un puntero a su arista gemela, un puntero a la cara en la que se encuentra, un puntero a la arista siguiente y otro a la anterior, todas ellas en sentido anti horario.

- ✓ Lista de caras: Un registro para una cara ***c***, almacena un puntero a una arista exterior a la cara y otro a una interior. Se puede ver que es necesario aplicar una determinada orientación a las aristas, ya que se habla de medios-lados orientados, de esta forma una arista podría considerarse interna a una cara si está orientada en sentido anti horario con respecto a la cara.(6)

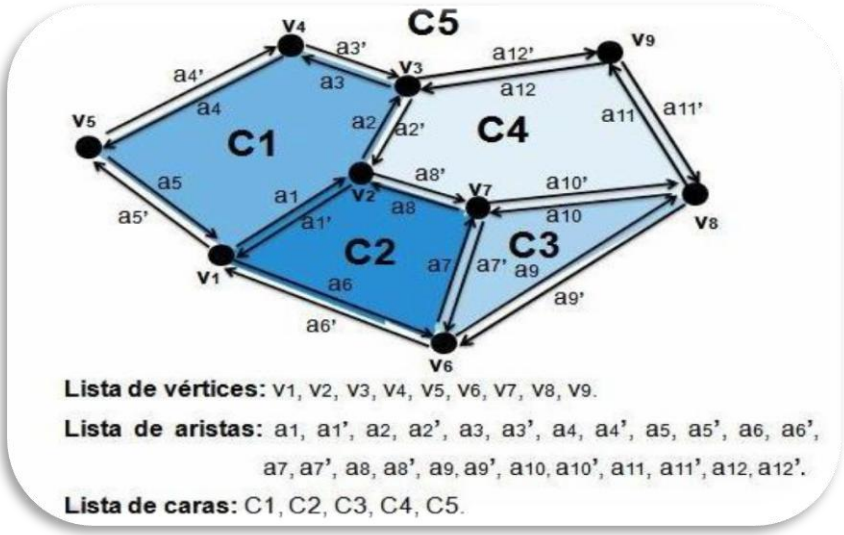


Figura 1.16 Representación gráfica de DCEL.

1.3 Objeto de Estudio.

De acuerdo a la investigación se escoge a las *Estructuras de Datos* como el objeto de estudio, debido q que son imprescindibles para la planificación del desarrollo del trabajo, además es la parte de la realidad objetiva sobre la cual se ejerce la mayor atención, tanto desde el punto de vista práctico como teórico, con vistas a la solución del problema planteado.

1.3.1 Descripción General.

Existen patrones de organización para las variables de instancia, estos patrones se denominan *estructuras de datos* y una de las aplicaciones más interesantes y potentes de la memoria dinámica y los punteros son precisamente estas.

En programación, una estructura de datos es una forma de organizar un conjunto de datos elementales [6] con el objetivo de facilitar la manipulación de estos datos como un todo o individualmente.

Los datos estándar pueden estar organizados en diferentes estructuras de datos: estáticas y dinámicas.

» *Estructuras de Datos Estáticas:*

Son aquellas en las que el espacio ocupado en la memoria de la computadora se define en tiempo de compilación y no puede ser modificado durante la ejecución del programa. Corresponden a este tipo los arreglos y los registros.

» *Estructuras de Datos Dinámicas:*

Son aquellas en las que el espacio ocupado en la memoria de la computadora puede ser modificado en tiempo de ejecución. Corresponden a este tipo las listas, las pilas, las colas, etc.

[6] Dato elemental es la mínima información que se tiene en el sistema.

La elección de la estructura de datos idónea dependerá de la naturaleza del problema a resolver y en menor medida, del lenguaje de programación. Cada estructura de datos dinámica posee un comportamiento específico y como tal una lógica diferente para operarla, tanto es así que no es lo mismo eliminar un nodo de una lista simplemente enlazada que uno en una de doble referencia, debido a que la lógica cambia aunque la operación sea la misma.

Las estructuras de datos tienen en común que un identificador y un nombre pueden representar a múltiples datos individuales.

Una estructura de datos define además la organización e interrelación de los elementos y un conjunto de operaciones que se pueden realizar sobre ellos. Las operaciones básicas son:

- ✓ **Adicionar**, adiciona un nuevo valor a la estructura.
- ✓ **Eliminar**, borra un valor de la estructura.
- ✓ **Buscar**, encuentra un determinado valor en la estructura para realizar una operación con este valor, en forma *secuencial* o *binaria* (siempre y cuando los datos estén ordenados).

Otras operaciones que se pueden realizar:

- ✓ **Ordenar**, ordena los elementos pertenecientes a la estructura.
- ✓ **Concatenar**, dadas dos estructuras origina una nueva ordenada que contenga a dichas estructuras.

Para la realización de las operaciones cada estructura ofrece ventajas y desventajas en relación a su simplicidad y eficiencia. De esta forma, la elección de la estructura de datos apropiada para cada problema depende de factores como la frecuencia y el orden en que se realiza cada operación sobre los datos, así como la naturaleza de los datos que se almacenarán.

1.3.2 Descripción actual del dominio del problema.

El entorno donde coexiste el problema se desarrolla en la Universidad de las Ciencias Informáticas, creada en el año 2002, como propuesta del Comandante en Jefe de fundar una universidad de excelencia con el objetivo de graduar Ingenieros en Ciencias Informáticas para desarrollar la producción del software cubano. La universidad está conformada por diez facultades que desarrollan cada una su perfil con sus propios proyectos productivos para los cuales, al igual que en la asignatura de programación, es necesario utilizar estructuras de datos avanzadas en la solución de determinados problemas.

Pero desafortunadamente, según investigaciones realizadas en la universidad, no se ha implementado aún una biblioteca que contenga agrupada una gran variedad de estructuras de datos y cuando hace falta utilizarlas hay que implementarlas por separado o usar las que existen aisladas.

Según lo descrito anteriormente, se ha decidido desarrollar una Biblioteca de Estructuras de Datos Avanzadas que esté disponible para ser usada en el momento que lo requiera para la elaboración de las diferentes aplicaciones informáticas.

1.3.3 Situación Problemática.

El bloqueo económico, político y social impuesto a Cuba limita, entorpece y encarece la adquisición del software legal necesario para el desarrollo de la industria cubana del software y es por esta razón que el Software Libre se abre como solución a esta problemática.

Sin duda alguna, el Software Libre es sustentable en Cuba, por este motivo su aplicación como plataforma informática de trabajo adquiere una relevante importancia que puede verse desde tres ámbitos diferentes:

- Económico: Su utilización no implica gastos adicionales por concepto de cambio de plataforma de software, por cuanto es operable en el mismo soporte de hardware con que cuenta el país.

  La adquisición de cualquiera de sus distribuciones puede hacerse de forma gratuita, descargándolas directamente de Internet o en algunos casos a muy bajos precios. Se garantiza su explotación con un mínimo de recursos, por lo cual no hay que pagar absolutamente nada para su distribución, modificación o utilización (no requiere de licencia de uso, las cuales son generalmente muy caras).

- Político: Desde un primer punto de vista representa la no utilización de productos informáticos que demanden la autorización de sus propietarios (licencias) para su explotación. En la actualidad Cuba se encuentra a merced de la empresa norteamericana Microsoft, que tiene la capacidad legal de reclamar que no siga utilizando un sistema operativo de su propiedad, basada en leyes de propiedad industrial por las cuales también Cuba se rige; esto provocaría una interrupción inmediata del programa de informatización de la sociedad que como parte de la batalla de ideas está desarrollando el país, además pudiera implementarse una campaña de desacredito a la Isla, abogando el uso de la piratería informática por parte de las instituciones estatales cubanas.

  Desde un segundo punto de vista, el Software Libre representa la alternativa para los países pobres, que una vez que comienza a circular rápidamente se encuentra disponible para todos los interesados sin costo alguno o en su defecto a muy bajo costo.

  En tercer lugar es desarrollado de forma colectiva y cooperativa, tanto en su creación como en su desarrollo, mostrando su carácter público y sus objetivos para beneficiar a toda la comunidad.

- Tecnológico: Permite su adaptación a los contextos de aplicación, al contar con su código fuente, lo cual garantiza un mayor porciento de efectividad, además de la corrección de los errores de programación y la obtención de las actualizaciones y de las nuevas versiones.

Actualmente la Universidad de las Ciencias Informáticas lleva a cabo un significativo proceso de migración hacia el Software Libre, ya que muchas de las herramientas que se utilizan son propietarias y hay que pagar grandes sumas de dinero por la adquisición de sus licencias. La universidad juega un papel importantísimo en la producción de software para las diferentes instituciones y empresas, ya sean nacionales o extranjeras, como es el caso de PDVSA; en la misma se utilizan junto con las tecnologías y herramientas, diferentes tipos de estructuras de datos para el manejo con grandes cantidades de información.

A nivel mundial existen bibliotecas conformadas por estructuras de datos pero en su forma simple, es decir, no agrupan las variantes de implementación de estas. Las estructuras están implementadas en diversos lenguajes de programación como *C++, C#* y *Java* que poseen estructuras básicas, tal es el caso de los registros, que además son estructuras de datos estáticas y tienen como marcada limitación que el espacio ocupado en la memoria de la computadora se define en tiempo de compilación y no puede ser modificado durante la ejecución del programa. Para solucionar este problema están las estructuras de datos dinámicas, como las listas, las pilas, las colas, etc., en las que el espacio ocupado en la memoria de la computadora puede ser modificado en tiempo de ejecución, por tal motivo pueden almacenar grandes cantidades de elementos de diferentes tipos. Su elección dependerá en gran medida de la naturaleza del problema a resolver y en menor medida, del lenguaje de programación empleado.

La selección adecuada de las estructuras de datos y el algoritmo empleado permite obtener un diseño eficiente, tanto en los recursos ocupados en la memoria de la computadora como en el tiempo de ejecución de los programas. Cada estructura de datos dinámica posee un comportamiento específico y como tal una lógica diferente para operarla, permitiendo así crear estructuras de datos que se adapten a las necesidades reales a las que suelen enfrentarse los programas.

En la carrera de Ingeniería Informática o similares es prácticamente obligatorio, debido a su importancia, el estudio de las estructuras de datos, pero en su versión más simple

(listas, pilas, colas, árboles y grafos sencillos), lo cual sirve para sumergirse en esta importante materia y darle solución a determinados problemas.

Para las estructuras de datos se han realizado varios trabajos en el mundo y en Cuba, principalmente en la Universidad de las Ciencias Informáticas, pero lo que se ha desarrollado con respecto a este tema en la universidad es para resolver problemas puntuales de la docencia, por ejemplo, que sirvan de apoyo en las clases y tareas finales de la asignatura de Programación o para el desarrollo de los diversos proyectos productivos.

Luego, ¿Qué sucedería si los problemas a resolver, fueran tan complejos como representar terrenos o implementar algoritmos para Gráficos por Computadora, que necesitan manejar grandes cantidades de información en la memoria de la computadora? Para estos casos es necesario utilizar estructuras de datos avanzadas que son óptimas para el trabajo con grandes volúmenes de datos, además minimizan los recursos de la computadora en cuanto a tiempo de ejecución y consumo de memoria.

Pero desafortunadamente, según investigaciones realizadas en la universidad, no se ha implementado aún una biblioteca que contenga agrupada una gran variedad de estructuras de datos y cuando hace falta utilizarlas hay que implementarlas por separado o usar las que existen aisladas. Por tal motivo el *problema científico* es precisamente: "la poca disponibilidad de bibliotecas que contengan agrupadas una gran variedad de estructuras de datos avanzadas", con el fin de ser usadas en cualquier solución que lo requiera.

1.4 Análisis de otras soluciones existentes.

A nivel mundial existen bibliotecas de estructuras de datos, pero no son tan abarcadoras ya que no tienen agrupadas una gran variedad de ellas.

» Un ejemplo es la biblioteca *Java Development Kit* (JDK), la cual incluye el paquete *Java Collections*, que está compuesta por las siguientes estructuras:

- *ArrayList:* implementada mediante un arreglo
- *HashMap*: implementada mediante una tabla hash.
- *LinkedList*: implementada mediante una lista enlazada.
- *TreeMap*: implementada mediante un árbol binario.
- *TreeSet*: implementada mediante un árbol binario ordenado.

Estas estructuras son más abarcadoras, pero no cubren más variantes de estructuras de datos y no brindan el código fuente para su posterior análisis y modificación.

» La *BCL* (Biblioteca de Clase Base) de **.**Net, es una biblioteca incluida en el ***.NET Framework*** que incluye la *System**.**Collections,* la cual permite el trabajo con estructuras de datos simples como: las listas, las pilas, las colas y los diccionarios, pero no incluyen las variaciones complejas de las mismas. (7)

» La *STL* (Biblioteca de Plantilla Estándar) es un subconjunto de la biblioteca estándar de C ++ que contiene una colección de clases y funciones que están escritas en el núcleo del lenguaje, la cual incluye solamente colecciones de estructuras como: las listas, las pilas y las colas. (8)

» Además es válido decir que *Python* soporta el trabajo con las colecciones: tuplas, listas y diccionarios.

En Cuba, específicamente en la Universidad de las Ciencias Informáticas, no existe una biblioteca que contenga agrupada una gran diversidad de estructuras de datos y en sus variantes formas de implementación, aunque se han realizado intentos de elaborar una pero no se ha llegado a publicar. Tal es el caso del proyecto "Unicornio", que elaboraron una biblioteca con listas para el trabajo con aplicaciones multiventanas.

Existen por otro lado y de forma aislada los TDA que son empleados en la docencia, específicamente en la asignatura de Programación y además se utilizan en el desarrollo de los diferentes proyectos productivos, los cuales se encargan de implementar algunas de las estructuras de datos avanzadas, pero siempre para resolver problemas puntuales en sus trabajos.

1.5 Conclusiones Parciales.

En este capítulo se efectuó una fundamentación mucho más amplia sobre temas relacionados con el objeto de estudio, estipulando las condiciones específicas que circundan al problema. Se investigó al mismo tiempo sobre las diversas soluciones que existen y que de una forma parcial sirven de alguna manera como respuesta al problema científico planteado. Además se realizó un pequeño recuento de las estructuras de datos a utilizar en esta biblioteca, lo cual permitirá realizar un trabajo más detallado y comprensible.

Capítulo 2

Tendencias y tecnologías actuales a utilizar.

2.1 Introducción.

En este capítulo se pretende hacer un recorrido a través de las diferentes tendencias y tecnologías actuales a utilizar para el desarrollo de la Biblioteca de Estructuras de Datos Avanzadas, tal es el caso del lenguaje de modelado, la metodología de desarrollo de software que lo soporta, la herramienta CASE[7], el lenguaje de programación y otras herramientas complementarias que de una forma u otra contribuyen al desarrollo de otros aspectos no menos importantes. De cada una se realiza una breve descripción con las principales características y se fundamentan las que en un final se escogen para el desarrollo de la propuesta final.

2.2 Lenguaje de modelado.

La finalidad de los diagramas es representar las diversas perspectivas de un sistema, a las cuales se les conoce como modelo. El lenguaje de modelado de objetos es un conjunto estandarizado de símbolos y la manera en que se ubican, para modelar parte del diseño de un software orientado a objetos.

[7]CASE es el acrónimo correspondiente a *Computer Aided Software Engineering*, cuya traducción literal sería: Ingeniería de Software Asistida por Computadora.

2.2.1 Lenguaje Unificado de Modelado (UML).

El Lenguaje Unificado de Modelado (UML) se compone por diversos elementos gráficos que se combinan para conformar los diagramas. Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar los artefactos de un sistema, además puede utilizarse para escribir planos de software.(9)

Es importante recalcar que UML no es una guía para realizar el análisis y el diseño orientado a objetos, es decir, no es un proceso, sino un lenguaje que permite la modelación de sistemas con tecnología orientada a objetos. Un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementarlo.

UML no es un estándar solamente de la industria del software sino de cualquier industria en general, que requiera de la elaboración de modelos como condición previa para el diseño y posteriormente para la construcción de los prototipos.(10)

El modelado de los sistemas con UML siguiendo los pasos del Proceso Unificado de Desarrollo de Software, que tiene como objetivo producir software de alta calidad, incluye actividades específicas y a su vez cada una de ellas contienen otras sub actividades que sirven como una guía de cómo deben ser las actividades desarrolladas y secuenciadas con el fin de obtener sistemas exitosos. Consecuentemente el desarrollo de los sistemas puede variar de desarrollador en proyecto o de empresa en empresa, adoptando siempre un Proceso de Desarrollo de Software.(11)

2.2.2 Fundamentación del lenguaje de modelado a utilizar.

El Lenguaje Unificado de Modelado (UML) es la notación de modelado que se utilizará como soporte para la modelación de la solución propuesta y lograr así un mayor entendimiento de la misma.

UML ayuda al usuario a entender la realidad desde el punto de vista de la tecnología y les da la posibilidad de que reflexionen antes de invertir y gastar grandes cantidades de dinero en proyectos que no estén seguros en su desarrollo y de esta forma reducir el coste y el tiempo empleado en la construcción de las piezas que conformarán el modelo.

Desde el punto de vista puramente tecnológico, UML tiene una gran variedad de propiedades que han sido las que realmente han contribuido a hacer de este, el lenguaje de modelado estándar de la industria que es en realidad.

» Propiedades de UML como lenguaje de modelado:

1. Es un lenguaje distribuido y adecuado a las necesidades actuales y futuras.
2. Es ampliamente utilizado por la industria del software desde su adopción por la OMG[8], la cual ofrece interoperabilidad multiplataforma a nivel de objetos de negocios.
3. Reemplaza a decenas de notaciones empleadas por otros lenguajes.
4. Modela estructuras complejas.
5. Las estructuras más importantes que soporta tienen su fundamento en la tecnología orientada a objeto, tales como: objetos, clases, componentes y nodos.
6. Emplea operaciones abstractas como guía para las variaciones futuras y permite añadir variables en caso de que sea necesario.
7. Muestra el comportamiento del sistema a través de: casos de uso, diagramas de secuencia y de colaboración, los cuales sirven para evaluar el estado de las máquinas.

» Beneficios de UML:

Entre más complejo es el sistema que se desea crear más beneficios presenta el uso de UML, entre ellos se encuentran:

[8] OMG es el acrónimo correspondiente a *Object Management Group.*

- La reutilización del código.
- El diseño y la documentación.
- El descubrimiento de fallas.
- El ahorro de tiempo en el desarrollo del software.
- Que las modificaciones sean mucho más fáciles.
- Que la comunicación entre los programadores sea más fluida.(12)

2.3 Metodologías de desarrollo de software.

En un proceso de desarrollo de software la metodología define "Quién" debe hacer "Qué", "Cuándo" y "Cómo" para alcanzar un objetivo y aumentar la calidad del software que se produce en todas y cada una de sus fases de desarrollo. Seleccionar la metodología más apropiada que posibilite obtener óptimos resultados en el desarrollo de un software es una de las trabas principales de hoy en día. (13)

Para dar una idea de qué metodología se puede utilizar y cual se adapta más al problema en cuestión, se analizarán seguidamente tres de las más conocidas en la actualidad para el proceso de desarrollo de un software, estas son:

- » Proceso Unificado de Desarrollo (RUP).
- » Programación Extrema (XP).
- » Desarrollo Guiado por Funcionalidad (FDD).

2.3.1 Proceso Unificado de Desarrollo (RUP).

RUP es uno de los procesos de Ingeniería de Software más generales que existe, está enfocado a cualquier tipo de proyecto y se basa en la documentación generada en cada una de sus cuatro fases en las cuales se ejecutarán varias iteraciones según el tamaño del proyecto:

1. Inicio.
2. Elaboración.
3. Construcción.
4. Transición.

Su objetivo es producir software de alta calidad, es decir, que cumpla con los requerimientos de los usuarios dentro de una planificación y presupuestos establecidos. Incluye artefactos[9] y roles[10].(14)

El ciclo de vida de RUP se caracteriza por:

1. Ser iterativo e incremental.
2. Estar centrado en la arquitectura.
3. Estar guiado por los casos de uso.

En la Figura 2.1 se representa el Proceso Unificado de Desarrollo en el que se grafican los flujos de trabajo, las fases y la dinámica expresada en iteraciones y puntos de control.

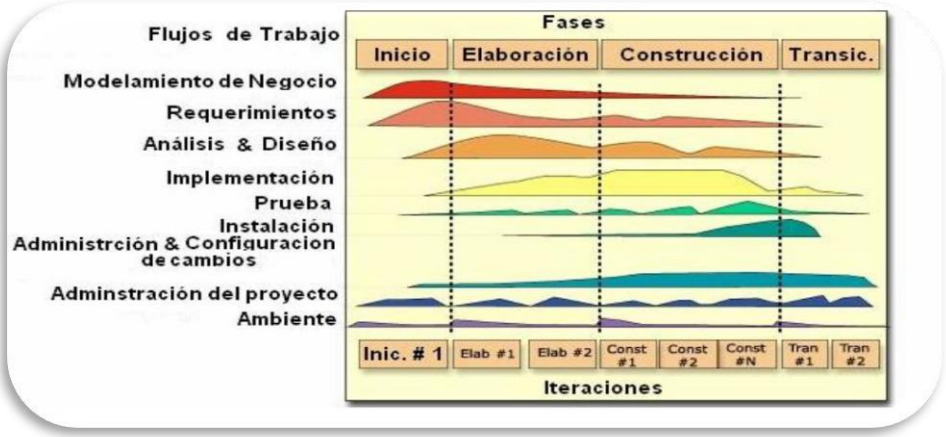


Figura 2.1 RUP en Dos Dimensiones.

[9] Artefactos son los productos tangibles del proceso, por ejemplo: el modelo de casos de uso, el código fuente, etc.

[10] Rol es el papel que desempeña una persona en un determinado momento; una persona puede desempeñar distintos roles a lo largo del proceso.

2.3.2 Programación Extrema (XP).

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo del software promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen clima de trabajo. (15)

XP se basa en:

- La realimentación continúa entre el cliente y el equipo de desarrollo.
- La comunicación fluida entre todos los participantes.
- La simplicidad en las soluciones implementadas.
- El coraje para enfrentar los cambios.

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre.

El ciclo de desarrollo consiste a grandes rasgos en los siguientes pasos:

1. El cliente define el valor del negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona que es lo que se va a construir de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso uno.

El ciclo de vida ideal de XP se define en seis fases:

1. Exploración.
2. Planificación de la Entrega (*Release*).
3. Iteraciones.
4. Producción.
5. Mantenimiento.
6. Muerte del Proyecto. (16)

Cuando en un desarrollo de software se requiere de una documentación que esté más allá del código fuente, XP no es deseado ya que no abarca toda la documentación necesaria para el cliente o la empresa.

2.3.3 Desarrollo Guiado por Funcionalidad (FDD).

FDD se considera como un punto medio entre los procesos pesados y ágiles, RUP y XP respectivamente, aunque en la práctica es más similar a este último.

Está pensado para proyectos con tiempo de desarrollo cortos que a medida que itera se genera, al igual que en XP, un software funcional que el cliente o dirección de la empresa pueden ver y monitorizar. Además se trabaja en grupo por lo que los menos expertos se benefician de la experiencia de los demás.

FDD, al ser un proceso intermedio, genera más documentación que XP, pero menos que RUP, igual determina que es lo que se debe documentar. En casos donde se requiera documentación externa más allá del código fuente, RUP y FDD son mejores alternativas. (17)

Está pensado para proyectos con tiempo de desarrollo relativamente cortos (menos de un año). Se basa en un proceso iterativo con iteraciones cortas (dos semanas) que producen un software funcional que puede ser visto, probado y monitorizado por el cliente. Estas iteraciones son decididas en base a las funcionalidades que el software debe tener, funcionalidades definidas por el cliente. Este proceso está dividido en cinco fases:

1. Desarrollo de un modelo general.
2. Construcción de una lista de funcionalidades.
3. Plan de *releases* en base a las funcionalidades a implementar.
4. Diseñar en la base las funcionalidades definidas.
5. Implementar en la base a las mismas funcionalidades.

Las primeras tres fases ocupan gran parte del tiempo en las primeras iteraciones, siendo las dos últimas las que absorben la mayor parte del tiempo según va avanzando el proyecto, limitándose las primeras a un proceso de refinamiento. Las funcionalidades a implementar en un *release*[11] se dividen entre los distintos subgrupos del equipo y se procede a implementarlas. (18)

2.3.4 Fundamentación de la metodología a utilizar.

Luego de haber hecho un estudio de estas tres metodologías, se ha llegado a la conclusión de que la metodología a utilizar como base para el desarrollo de la solución propuesta es el Proceso Unificado de Desarrollo de Software (RUP).

1. Dado que este es un proceso general que cubre todas las posibles expectativas tanto del cliente como del equipo de desarrollo.
2. Es el resultado de la evolución e integración de diferentes metodologías de desarrollo de software.
3. Permite sacar el máximo provecho de los conceptos asociados a la orientación de objetos y al modelado visual.
4. Cuenta con las mejoras prácticas en el desarrollo del software, tal es el caso de:
    - La administración de los requerimientos.
    - La utilización de una arquitectura de componentes.
    - La modelación visual.
    - La verificación de la calidad.
    - El desarrollo iterativo.
    - El control de los cambios. (14)
5. Además este proceso de desarrollo de software junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de los sistemas orientados a objetos.

[11] Término bajo el cual se reconoce una aproximación.

6. Permite la mitigación temprana de los posibles riesgos, lo cual es un proceso visible en las primeras etapas del desarrollo del software.
7. También por ser la más completa y abarcadora, ya que como señalan algunos autores las otras metodologías son casos particulares de esta. Además XP y FDD presentan algunas debilidades, lo que representa riesgos considerables a la hora de una buena obtención de los requisitos para el sistema.

2.4 Herramienta CASE de Desarrollo de Software.

En las últimas décadas se ha trabajado en el área del desarrollo de sistemas para encontrar técnicas que permitan incrementar la productividad y el control de la calidad en cualquier proceso de elaboración de software y hoy en día la tecnología CASE reemplaza al papel y al lápiz por el ordenador, para transformar la actividad de desarrollar software en un proceso automatizado. Estas herramientas se plantean como objetivo: simplificar el mantenimiento de los programas, aumentar la portabilidad de las aplicaciones, facilitar la reutilización de los componentes de software, entre otros.

2.4.1 Rational Rose Enterprise Edition.

La suite de Rational ofrece varios productos, destacándose el *Rational Rose Enterprise Edition* la cual es la herramienta CASE que comercializan los desarrolladores y que soporta de forma completa la especificación del UML.

Esta herramienta propone la utilización de modelos para realizar el diseño del sistema, lo cual representa el dominio del problema y el sistema de software, esto permite crear y refinar estas vistas:

- Vista Estática.
- Vista Dinámica de los modelos del sistema.
- Vista Lógica.
- Vista Física. (11)

A continuación se muestran los tipos de modelos:

1. Desarrollo Iterativo.
2. Generador de Código.
3. Ingeniería Inversa.
4. Trabajo en Grupo.

La principal desventaja del *Rational Rose* es que no es multiplataforma, es decir, solo defiende los intereses de Microsoft y se utiliza solamente en las plataformas que este desarrolla y no en otras como Unix y Macintosh. Otra desventaja es que es un software propietario.

2.4.2 Visual Paradigm.

Es una herramienta CASE que le facilita a los ingenieros de software diseñar, integrar y modelar visualmente los distintos diagramas que se generan a lo largo del desarrollo del software. Presenta un generador de código que proporciona la ingeniería inversa y soporta a más de 10 lenguajes, entre los que se encuentran:

- Java.
- C++.
- CORBA.
- IDL.
- PHP.
- Esquema de XML.

Esta herramienta está especializada en la ingeniería del software de las bases de datos. Tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de la base de datos a partir del esquema de clases. Incluye herramientas muy interesantes para la ingeniería inversa de bases de datos.

También es una poderosa herramienta para la generación de documentos PDF/HTML a partir de los diferentes diagramas UML.

Básicamente se trata de una herramienta para trabajar con UML, considerada como muy completa y fácil de usar, con soporte multiplataforma y que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. (19; 20)

2.4.3 Enterprise Architect.

Enterprise Architect es una herramienta CASE que cubre el desarrollo del software a través de las etapas de: Requerimientos, Análisis & Diseño y Prueba.

Es una herramienta multi-usuario diseñada para ayudar a construir un software robusto y fácil de mantener. Ofrece salida de una documentación flexible y de alta calidad. Combina el poder de la última especificación de UML 2.1 con alto rendimiento e interfaz intuitiva.

Provee una generación poderosa de documentos y herramientas de reporte con un editor de plantilla completa.

Soporta la generación de código fuente e ingeniería inversa para muchos lenguajes populares incluyendo: C++, C#, Java, Delphi, Visual Basic y PHP.

Puede desarrollar rápidamente soluciones complejas desde los simples "modelos independientes de plataforma" (MIP), que son el objetivo de los "modelos específicos de plataforma" (MEP). Un MIP se puede usar para generar y sincronizar múltiples MIP's, el cual suministra un aumento significativo de la productividad.

2.4.4 Fundamentación de la herramienta CASE a utilizar.

Después de hacer un estudio de estas herramientas CASE, se llegó a la conclusión de que la herramienta a utilizar para un buen desarrollo ingenieril de esta investigación es el *Visual Paradigm* primeramente porque es un requerimiento del proyecto y además es una herramienta de modelado multiplataforma. A continuación se muestran algunas de sus características fundamentales:

1. Unifica el formato de todos los diagramas, brinda soporte para toda la notación UML, presenta diagramas de capas sofisticados así como análisis de textos.
2. Proporciona características tales como la ingeniería inversa y la generación de código y de informes.
3. Permite invertir el código fuente de los programas, archivos ejecutables y binarios en modelos UML al instante, creando de manera simple toda la documentación.
4. Otra de las ventajas que ofrece es la navegación intuitiva entre el modelo visual y el código, además de permitir la sincronización entre el código fuente y el modelo en tiempo real o bajo demanda.
5. Permite gestionar proyectos muy complejos con gran sencillez.
6. Es un editor de figuras. Proporciona la exportación de imágenes *jpg, png* y *svg*.
7. Importa archivos desde el *Rational Rose*.

2.5 Lenguajes de programación.

En el mundo de la programación existen diversos lenguajes que se han ido creando con el paso del tiempo. Basados en los primeros lenguajes de programación han surgido muchos otros que siempre tienen la intención de tomar lo mejor, desechar lo malo y agregar alguna "novedad" respecto a los existentes.

2.5.1 C++.

La expresión "C++" significa "incremento de C" y se refiere a que *C++* es una extensión de *C*. Es un lenguaje híbrido, que se puede compilar y resulta más sencillo de aprender para los programadores que ya conocen *C*. Actualmente existe un estándar, denominado *ISO*[12] *C++,* al que se han adherido la mayoría de los fabricantes de los compiladores más modernos.

[12] Organización Internacional para la Normalización.

Las principales características son la abstracción (encapsulamiento), el soporte para la programación orientada a objetos (polimorfismo) y el soporte de plantillas o de la programación genérica (*templates*).

Es un lenguaje que abarca tres paradigmas de la programación: Programación Estructurada, Programación Genérica y Programación Orientada a Objetos.

Tiene la desventaja de que obliga a hacerlo casi todo manualmente al igual que *C*, lo que "dificulta" mucho su aprendizaje. Otra desventaja es que antes de utilizar una variable hay que declarar su tipo con antelación.(21)

2.5.2 Java.

Java es un lenguaje de programación orientado a objetos con el que se puede realizar cualquier tipo de programa, es un lenguaje muy extendido, independiente de la plataforma y es compilado en un *bytecode* que luego es interpretado. Su sintaxis está inspirada en la de C/C++.

Es un lenguaje interpretado, el intérprete a código máquina (dependiente de la plataforma) se llama *Java Virtual Machine* (JVM). El compilador produce un código intermedio independiente del sistema denominado *bytecode*.(22)

Una de las desventajas que tiene este lenguaje es que no ofrece herencia múltiple, además es un lenguaje poco apropiado para desarrollar aplicaciones base como Sistemas Operativos.

2.5.3 Python.

Python es un lenguaje de programación ágil que está en un proceso continuo de desarrollo por una gran comunidad de desarrolladores. Aproximadamente cada seis meses se hace una publicación de una nueva versión de Python y de esta manera se

enriquece manteniendo en lo posible la compatibilidad con los programas escritos para versiones anteriores.(23)

Python es un lenguaje muy expresivo y llega a ser considerado por muchos programadores como un lenguaje de programación de alto nivel. Es muy legible y ofrece un entorno interactivo que facilita la realización de pruebas, además ayuda a despejar dudas acerca de ciertas características del lenguaje.

Su sintaxis es sencilla y tiene incorporadas características muy poderosas. Permite diversos estilos de programación, incluyendo tanto técnicas convencionales como las más modernas orientadas a objetos.

Tiene varias ventajas con respecto a la versión estándar de *DOS*:

- Permite la re-paginación del cursor facilitando el retipeo de las líneas.
- Las variables en Python pueden guardar cualquier tipo de dato sin necesidad de predefinirlo, esto se denomina: variables débilmente tipadas, ellas asumirán el tipo de dato que se le asigne, teniendo el usuario la posibilidad de asignarle otro tipo de dato durante la ejecución del programa.(24)

2.5.4 Fundamentación del lenguaje de programación a utilizar.

El lenguaje de programación que se utilizará para la programación de las estructuras de datos avanzadas que conformarán a la biblioteca es Python, el cual es un lenguaje de programación potente y fácil de aprender. Para fundamentar esta decisión se mencionan a continuación algunas características y ventajas que lo hacen muy atractivo.

1. Es un lenguaje de propósito general, debido a que toda aplicación programable con *C#* o *Java* también puede ser programada en *Python*.
2. Posee múltiples compiladores en diversas plataformas como: *Unix/Linux*, *MS Windows*, *Macintosh* y otros.

3. Es un lenguaje Orientado a Objetos donde se destaca la herencia múltiple como uno de sus principales logros y ofrece una manera sencilla de crear programas con componentes reutilizables.
4. Es multi-paradigma ya que permite varios estilos de programación por ejemplo: Programación Orientada a Objetos, Programación Estructurada o Procedural y Programación Funcional.
5. Posee una sintaxis elegante y clara que se acerca al lenguaje natural, garantizando que todos los programadores tengan un mismo estilo de organizar el código y a su vez permite un fácil entendimiento a los que lo consulten. Las peculiaridades sintácticas de Python traen consigo una reducción de los caracteres utilizados y de líneas de códigos.
6. A diferencia de otros lenguajes soporta en su estructura números complejos así como números enteros de precisión ilimitada.
7. Permite dividir sus programas en módulos reutilizables desde otros programas Python. Viene con una colección bastante amplia de módulos estándares que pueden utilizarse como base de los programas.
8. Python ha sido utilizado para desarrollar grandes proyectos de software como el servidor de aplicaciones *Zope*, parte de la implementación de *Google*, *Yahoo, NASA*, *Walt Disney*, etc.
9. Es *Open Source*, razón por la cual la biblioteca de módulos de Python contiene un sin fin de módulos de utilidad y sigue en constante perfeccionamiento y crecimiento.
10. El entorno de ejecución de Python detecta mucho de los errores de programación que se escapan del control de los compiladores y proporciona información muy rica para detectarlos y corregirlos.
11. Otra ventaja fundamental de Python es la gratuidad de su intérprete y tiene versiones para prácticamente cualquier plataforma en uso: sistemas PC bajo *Linux*, sistemas PC bajo *Microsoft Windows*, sistemas *Macintosh* de *Apple*, etc.

Como nota positiva comentar que la mayoría de las palabras claves utilizadas por *Python* coinciden con las usadas en *C* o *Java*.

En general, muchos programadores e informáticos toman como desventajas ciertas características de Python y eso los hace optar por otros lenguajes:

- No es un lenguaje apropiado para la programación a bajo nivel (por ejemplo: kernels y drivers) ni tiene el control de la memoria, debido a sus características de lenguaje interpretado y scripting.
- Al ser un lenguaje interpretado, trae consigo una menor velocidad de procesamiento del código ya que para ejecutar el código tendría que chequear los errores que puedan haber surgido dentro del código, no así en los demás lenguajes como *C* o *C++* por solo citar dos, ya que estos sí que crean un ejecutable con códigos binarios gracias al compilador asociado a ellos.
- No es adecuado para aplicaciones que requieren una alta capacidad de cómputo, por ejemplo para el procesamiento de imágenes.

¿Por qué IDLE Python 2.5 y PyScripter como Entorno de Desarrollo Integrado (IDE) para la programación de la biblioteca?

Desde el surgimiento de Python han sido muchos los Entorno de Desarrollo Integrado que se han desarrollado para este lenguaje, incluso la consola de *Linux* soporta sus instrucciones y comandos. Entre los más utilizados se encuentran *IDLE Python* en sus distintas versiones, que es proporcionado por la organización *Python.org*.

Para la programación de todas las estructuras de datos avanzadas se utilizó el *IDLE Python 2.5* por ser la última versión del IDE que proporciona la organización creadora del lenguaje *Python.org*. El *PyScripter* es un IDE cómodo ya que brinda opciones de completamiento de código y traceo. Además como Python es un lenguaje interpretado, el *bytecode* que genera es entendible al mismo tiempo por todos los sistemas operativos que lo soportan.

2.6 Herramientas complementarias.

Para cumplimentar este trabajo, fueron necesarias otras herramientas que aseguraron otros aspectos no menos importantes.

- Entre las herramientas complementarias empleadas está el *Adobe Photoshop 7.0* que constituye una aplicación de edición y retoque de imágenes *bitmap, jpg, gif*, etc. Es el software estándar de edición de imágenes profesional y es el líder de la gama de productos de edición de imágenes digitales.(25)
  Esta herramienta se empleó para la representación gráfica de las estructuras de datos, así como para la confección del logo y de la portada del documento.

- El Gestor de referencia *EndNote 9.0* es una herramienta completa que integra en un solo programa: la búsqueda de bases de datos bibliográficas en Internet; permite la organización de las referencias, imágenes, PDF y otros archivos. (26)
  Esta herramienta se emplea para almacenar las referencias bibliográficas en la biblioteca personal del presente trabajo de diploma.

2.7 Conclusiones Parciales.

En este capítulo se presentó un resumen de las principales tendencias y tecnologías actuales involucradas en el desarrollo de la Biblioteca de Estructuras de Datos Avanzadas y sus principales características. Concluyéndose que en el desarrollo de la misma se hará uso de *UML* como lenguaje de modelado, *RUP* como metodología de desarrollo del software, *Visual Paradigm* como herramienta de Ingeniería del Software Asistida por Computadora, *Python* como lenguaje de programación y otras herramientas complementarias.

Capítulo 3

Presentación de la solución propuesta.

3.1 Introducción.

En el capítulo anterior se realizó un análisis de las propuestas tecnológicas para desarrollar el componente de software y a partir de este análisis y del conjunto de tecnologías involucradas se decidieron las que en un final se usarían en el desarrollo del trabajo.

A lo largo de este capítulo se exponen los resultados del Flujo de Trabajo de Requerimiento. Para facilitar un mejor entendimiento y claridad de la solución propuesta a desarrollar se definen los requerimientos funcionales y no funcionales, se justifica la definición de los actores del sistema, se presenta el diagrama de casos de uso del sistema y se brinda una descripción textual de cada uno de ellos.

3.2 Entorno donde trabajará el sistema.

3.2.1 Diagrama de clases del Modelo de Dominio.

A continuación se presenta el diagrama de clases del Modelo de Dominio General de la Biblioteca de Estructuras de Datos Avanzadas, en el cual se capturan los objetos más importantes en el contexto del sistema, estos objetos son eventos y entidades que describen la funcionalidad del sistema.

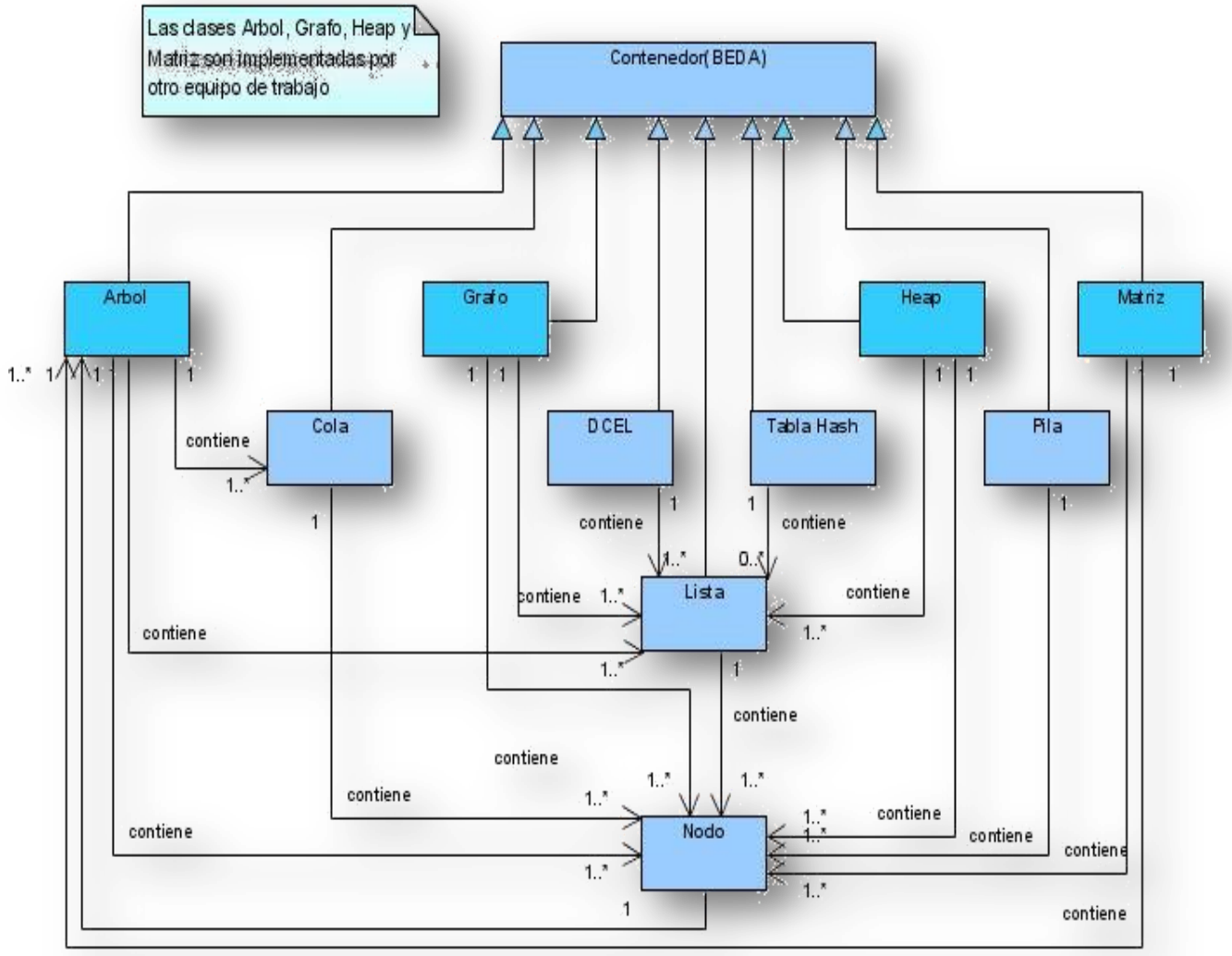


Figura 3.1 Diagrama general de clases del Modelo de Dominio.

A continuación se presenta el Modelo del Dominio de la Biblioteca con las Estructuras de Datos Avanzadas a desarrollar en el presente trabajo.

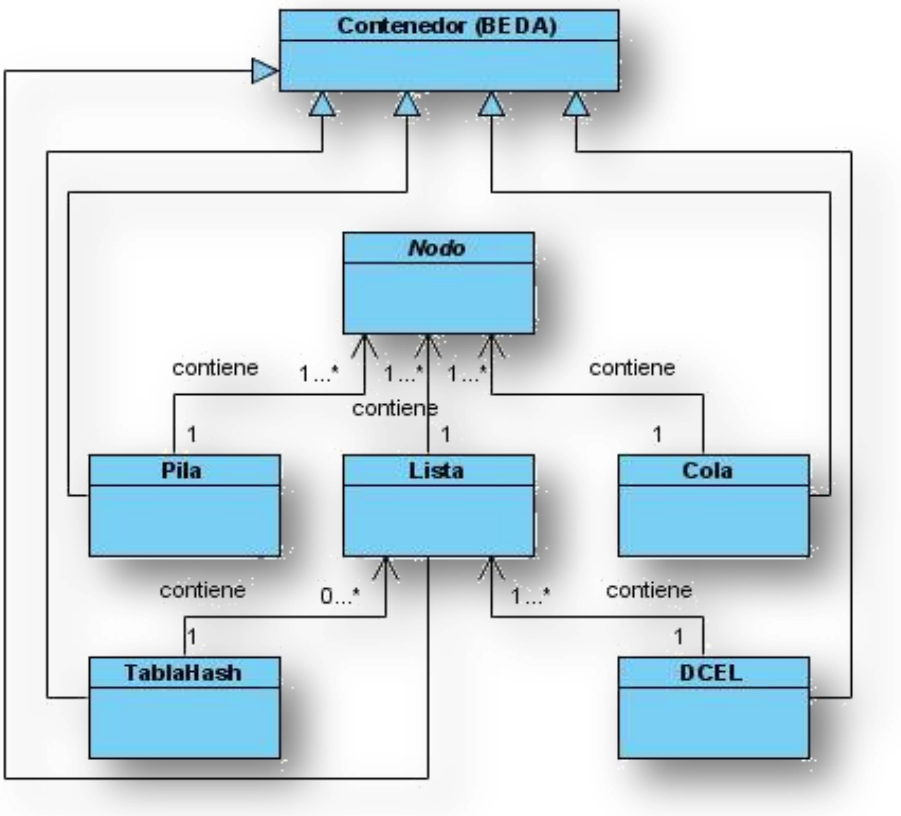


Figura 3.2 Diagrama particular de clases del Modelo de Dominio.

3.2.2 Glosario de Términos del Dominio.

Contenedor (BEDA): El contenedor BEDA[13] es una abstracción de lo que va a ser la Biblioteca de Estructuras de Datos Avanzadas, es decir, va a ser el fichero que contendrá el conjunto de estructuras de datos que se implementarán.

Nodo: Un nodo se compone por un campo que contiene el tipo de dato de los elementos y por uno o más campos que son referencias a otros nodos.

[13] BEDA es el acrónimo correspondiente a: Biblioteca de Estructuras de Datos Avanzadas.

Lista: Una lista se define como una n-tupla de elementos (donde l_i es el i-ésimo elemento de la lista) ordenados de forma consecutiva, o sea, el elemento l_i precede al elemento $l_i + 1$: $L = (l_1, l_2, \ldots, l_n)$.

Pila: Una pila es una estructura sencilla que se puede definir como una colección ordenada $S = (S_1, S_2, \ldots, S_n)$ donde los elementos se insertan y eliminan por un mismo extremo conocido como *tope*.

Cola: Una cola es una estructura de datos caracterizada por ser una lista ordenada de elementos $C = (C_1, C_2, \ldots, C_n)$, donde la operación de *Apilar* se realiza por un extremo llamado *cola* y la operación de extracción *Desapilar* por el otro llamado *cabeza*.

Tabla Hash: Una tabla hash es una estructura de datos que está formada por combinaciones de *llaves* o *claves* con *valores* organizados en sectores de almacenamiento para permitir realizar una búsqueda rápida.

DCEL: son las siglas inglesas de lista de lados doblemente enlazados. La estructura DCEL está formada por tres colecciones de registros: Lista de vértices, Lista de aristas y Lista de caras.

3.3 Funcionalidades del sistema propuesto.

Todas las ideas que los clientes, usuarios y miembros del equipo del proyecto tengan acerca de lo que debe hacer el sistema, deben ser analizadas como candidatas a requisitos.

3.3.1 Requerimientos Funcionales (RF).

Los requerimientos funcionales son las capacidades o condiciones que el sistema debe cumplir. Para determinar los requisitos funcionales del software RUP propone el Flujo de Trabajo de Requerimientos.(27). Se pueden agrupar por *entidades* o por *funcionalidades*.

A continuación se presentan los requisitos funcionales correspondientes a la Biblioteca de Estructuras de Datos Avanzadas agrupados por *funcionalidades*:

RF 1. Calcular la longitud.

El sistema debe permitir:

RF 1.1 Devolver la cantidad de elementos de la lista.

RF 1.2 Devolver la cantidad de elementos de la pila.

RF 1.3 Devolver la cantidad de elementos de la cola.

RF 2. Determinar si la estructura está vacía.

El sistema debe permitir:

RF 2.1 Determinar si la lista está vacía.

RF 2.2 Determinar si la pila está vacía.

RF 2.3 Determinar si la cola está vacía.

RF 3. Determinar si la estructura está llena.

El sistema debe permitir:

RF 3.1 Determinar si la lista con arreglo está llena.

RF 3.2 Determinar si la pila con arreglo está llena.

RF 3.3 Determinar si la cola con arreglo está llena.

RF 4. Buscar elementos.

El sistema debe permitir:

RF 4.1 Buscar elementos en la tabla hash dada una clave.

RF 5. Obtener elementos.

El sistema debe permitir:

RF 5.1 Obtener el elemento que está en una posición determinada de la lista.

RF 5.2 Obtener el elemento que se encuentra en el tope de la pila.

RF 5.3 Obtener el elemento que se encuentra en la primera posición de la cola.

RF 5.4 Obtener el elemento que se encuentra en la última posición de la cola.

RF 5.5 Obtener las aristas que pertenecen a una cara en la DCEL.

RF 5.6 Obtener las aristas salientes de un vértice en la DCEL.

RF 5.7 Obtener la cara correspondiente dado una arista en la DCEL.

RF 5.8 Obtener los vértices correspondientes a una cara en la DCEL.

RF 6. Adicionar elementos.

El sistema debe permitir:

RF 6.1 Adicionar un elemento al final de la lista.

RF 6.2 Apilar un elemento en el tope de la pila.

RF 6.3 Adicionar un elemento al final de la cola.

RF 6.4 Adicionar una arista en la DCEL.

RF 7. Insertar elementos.

El sistema debe permitir:

RF 7.1 Insertar un nuevo elemento en una posición determinada de la lista.

RF 7.2 Insertar un nuevo elemento en un índice determinado de la tabla hash.

RF 8. Eliminar elementos.

El sistema debe permitir:

RF 8.1 Eliminar el elemento que está en una posición determinada de la lista.

RF 8.2 Desapilar el elemento que se encuentra en el tope de la pila.

RF 8.3 Extraer el elemento que se encuentra en la cabeza de la cola.

RF 8.4 Eliminar el elemento que está en un índice determinado de la tabla hash.

3.3.2 Requerimientos No Funcionales (RNF).

Los requerimientos no funcionales son las propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto usable, rápido y confiable. (27).

RNF 1. Ayuda y documentación:

RNF 1.1 La Biblioteca de Estructuras de Datos Avanzadas contará con un documento de ayuda donde se esclarecerán las dudas sobre el uso de las diferentes estructuras de datos que la conforman.

RNF 2. Confiabilidad:

RNF 2.1 La herramienta de implementación a utilizar tiene soporte para la recuperación ante fallos y errores.

RNF 3. Legales:

RNF 3.1 La plataforma escogida para el desarrollo del producto está basado en la licencia *GNU/GPL*.

RNF 4. Portabilidad:

RNF 4.1 En la implementación del producto se debe prever en todo momento el uso de una herramienta capaz de funcionar en diferentes plataformas, como es el caso de *Windows*, *GNU/Linux* en sus diversas distribuciones, etc., acorde a las necesidades de los usuarios.

RNF 5. Rendimiento:

RNF 5.1 El tiempo de respuesta ante una petición debe de ser rápido.

RNF 6. Software:

RNF 6.1. El producto debe ser modelado utilizando la notación *UML*.

RNF 6.2. El producto debe ser desarrollado utilizando la metodología de desarrollo de software *RUP*.

RNF 6.3. El producto debe ser desarrollado utilizando la herramienta *CASE Visual Paradigm*.

RNF 6.4. El producto debe ser implementado utilizando el lenguaje de programación *Python*.

RNF 6.5. El producto debe permitir ser compilado sobre el Sistema Operativo *GNU/Linux* en cualquiera de sus distribuciones (*Debian*, *Ubuntu*, *Suse*).

RNF 7. Soporte:

RNF 7.1. El producto debe brindar soporte para grandes volúmenes de datos.

RNF 8. Usabilidad:

RNF 8.1. Se prevé que la usabilidad de la Biblioteca de Estructuras de Datos Avanzadas cuente con un elevado nivel de aceptación por los usuarios, debido a que constituye una forma más cómoda y rápida para la solución de problemas determinados.

3.4 Descripción del sistema propuesto.

Poniendo en práctica las habilidades y facilidades que brinda UML, en el Diagrama de Casos de Uso del Sistema, se muestra el actor que va a interactuar con el sistema y a la vez los casos de uso que van a representar a las diferentes funcionalidades.

3.4.1 Descripción de los actores del sistema a automatizar.

Basándose en que los casos de uso son una secuencia de acciones que obtienen resultados de valor para un actor y que un actor representa cualquier cosa que interactué con el sistema (puede ser un humano, un software o un hardware), se define a continuación el actor involucrado en los casos de uso del sistema.(27)

Tabla 3.1 Descripción del actor del sistema.

Actor	Descripción
Usuario	Rol que representa a un determinado sistema o programador que interactuará y se beneficiará con las funcionalidades que brinda la Biblioteca de Estructuras de Datos Avanzadas, la cual gestiona los elementos almacenados en los diferentes tipos de estructuras de datos.

3.4.2 Diagramas de casos de uso del sistema a automatizar.

Un diagrama de casos de uso del sistema representa gráficamente a los procesos y su interacción con los actores. A continuación se representa el diagrama de casos de usos del sistema de la Biblioteca de Estructuras de Datos Avanzadas.(27)

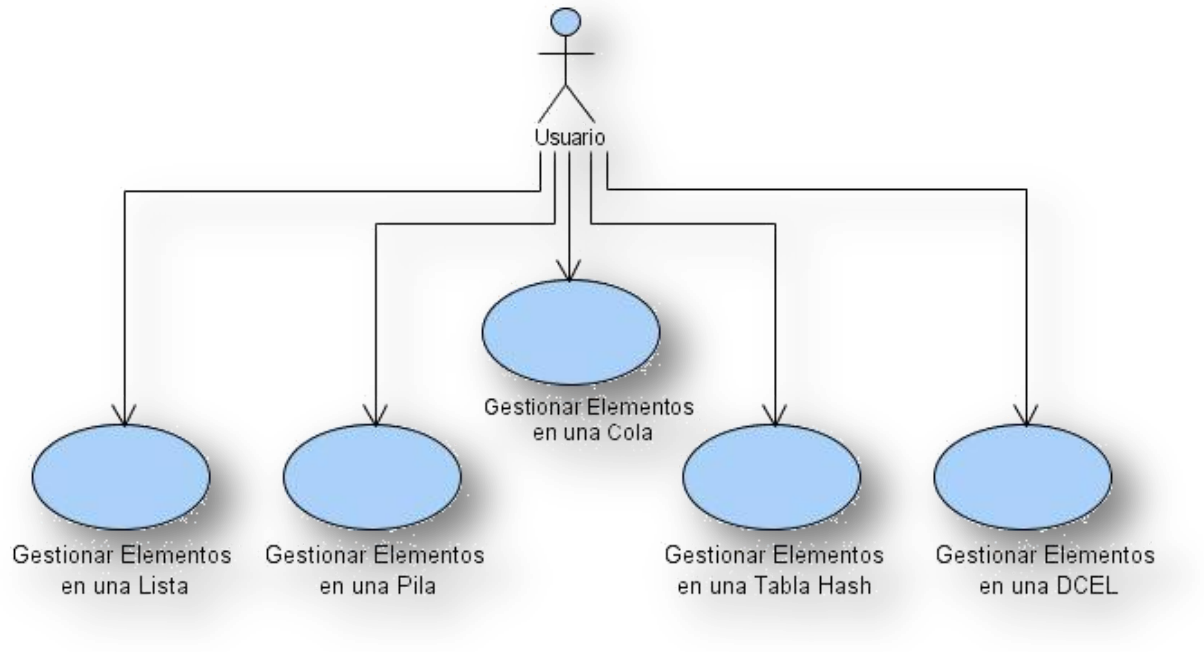


Figura 3.3 Diagrama de Casos de Uso del Sistema.

Tabla 3.2 Descripción del Caso de Uso: Gestionar elementos en una Lista.

Nombre del CUS	Gestionar elementos en una Lista.
Actor	Usuario (Inicia).
Propósito	Permite al usuario del sistema gestionar (obtener, adicionar, insertar y eliminar) los elementos que se encuentran almacenados en una lista.
Descripción	El CUS inicia cuando el usuario solicita gestionar los elementos que se encuentran almacenados en una lista, el sistema le pide

	el tipo de lista que desea emplear para la gestión de los elementos, el usuario selecciona el tipo de lista y proporciona los datos necesarios, el sistema realiza la acción seleccionada y termina el CUS.
Referencia	RF 1(RF 1.1), RF 2 (RF 2.1), RF 3 (RF 3.1), RF 5 (RF 5.1), RF 6 (RF 6.1), RF 7 (RF 7.1), RF 8 (RF 8.1).
Pos-condiciones	1- Elementos obtenidos de la lista. 2- Elementos adicionados en la lista. 3- Elementos insertados en la lista. 4- Elementos eliminados de la lista.

Flujo Normal de Eventos	
Acción del actor	**Respuesta del sistema**
1– El usuario solicita gestionar los elementos en una lista.	1.1- El sistema permite escoger el tipo de lista con la cual se van a gestionar los elementos y muestra las opciones: - Lista con Arreglo. - Lista Simplemente Enlazada. - Lista Circular Simplemente Enlazada. - Lista Doblemente Enlazada. - Lista Circular Doblemente Enlazada.
2- El usuario selecciona el tipo de lista.	2.1 – EL sistema le permite escoger el tipo de operación que desea realizar con la lista: - Obtener elementos. - Adicionar elementos. - Insertar elementos. - Eliminar elementos.

Sección 1: Obtener elementos.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de	1.1- El sistema devuelve el *i-ésimo*

Obtener elementos.	elemento de la lista (el que se encuentra en la posición ***i***) y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	1.1- Si la posición ***i*** no existe se dispara una excepción.
Sección 2: Adicionar elementos.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de *Adicionar elementos.*	1.1- El sistema adiciona un elemento ***x*** en la cola de la lista. 1.2- Se incrementa en uno la longitud de la lista y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	1.1- Si la operación no tiene éxito se lanza una excepción con un mensaje de error.
Sección 3: Insertar elementos.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de *Insertar elementos.*	1.1- El sistema inserta un nuevo elemento ***x***, en la posición ***i*** de la lista haciendo que los elementos $l_i, l_{i+1},\ldots, l_n$ pasen a ser los elementos $l_{i+1}, l_{i+2},\ldots, l_{n+1}$. 1.2- Se incrementa en uno la longitud de la lista y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	1.1- Si la operación no tiene éxito se lanza una excepción con un mensaje de error.
Sección 4: Eliminar elementos.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de	1.1- El sistema elimina el elemento que

Eliminar elementos.	está almacenado en la posición ***i*** de la lista, haciendo que los elementos l_{i+1}, $l_{i+2},\ldots,l_n$ pasen a ser los elementos l_i, $l_{i+1}, \ldots, l_{n-1}$. 1.2- Los elementos posteriores a partir de la posición ***i + 1*** pasan a tener la posición anterior inmediata es decir ***i***. 1.3- Se disminuye en uno la longitud de la lista y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	1.1- Si la posición no existe se lanza una excepción con un mensaje de error.

Tabla 3.3 Descripción del Caso de Uso: Gestionar elementos en una Pila.

Nombre del CUS	Gestionar elementos en una Pila.
Actor	Usuario (Inicia).
Propósito	Permite al usuario del sistema gestionar (obtener el elemento que se encuentra en el tope, apilar y desapilar) los elementos que se encuentran almacenados en una pila.
Descripción	El CUS inicia cuando el usuario solicita gestionar los elementos que se encuentran almacenados en una pila, el sistema le pide el tipo de pila que desea emplear para la gestión de los elementos, el usuario selecciona el tipo de pila y proporciona los datos necesarios, el sistema realiza la acción seleccionada y termina el CUS.
Referencia	RF 1 (RF 1.2), RF 2 (RF 2.2), RF 3 (RF 3.2), RF 5 (RF 5.1), RF 6 (RF 6.2), RF 8 (RF 8.2).
Pos-condiciones	1- Elemento que se encuentra en el tope de la pila obtenido.

	2- Elementos apilados en la pila. 3- Elementos desapilados de la pila.
Flujo Normal de Eventos	
Acción del actor	**Respuesta del sistema**
1– El usuario solicita gestionar los elementos en una pila. 2- El usuario selecciona el tipo de pila.	1.1- El sistema permite escoger el tipo de pila con la cual se van a gestionar los elementos y muestra las opciones: - Pila con Arreglo. - Pila con Listas Enlazables. 2.1 – EL sistema le permite escoger el tipo de operación que desea realizar con la pila. - Obtener el elemento que se encuentra en el tope. - Apilar elementos. - Desapilar elementos.
Sección 1: Obtener el elemento que se encuentra en el tope.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de *Obtener el elemento que se encuentra en el tope.*	1.1- El sistema verifica que la pila no esté vacía. 1.2- El sistema devuelve el elemento que se encuentra en el tope de la pila y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	1.1- Si en la pila no hay elemento alguno se lanza una excepción con un mensaje de error indicando que la pila está vacía.
Sección 2: Apilar elementos.	
Acción del actor	**Respuesta del sistema**

1- El usuario selecciona la opción de *Apilar elementos.*	1.1- El sistema verifica si la pila está vacía. 1.2- Si la pila está vacía el elemento sería el primero y se coloca en el tope de la pila. 1.3- Se incrementa la longitud de la pila en uno y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	1.2- Si la pila no está vacía se coloca al elemento seguidamente después del que estaba en el tope de la pila y este pasaría a ser ahora el nuevo tope.
Sección 3: Desapilar elementos.	
Acción del actor	**Respuesta del sistema**
1 - Selecciona la opción de *Desapilar elemento.*	1.1- El sistema extrae el elemento que se encuentra almacenado en el tope de la pila. 1.2- Ocurre un decremento en la longitud de la pila. 1.3- Se devuelve el elemento y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	1.1- Si en la pila no hay elemento alguno se lanza una excepción con un mensaje de error indicando que la pila está vacía.

Tabla 3.4 Descripción del Caso de Uso: Gestionar elementos en una Cola.

Nombre del CUS	Gestionar elementos en una Cola.
Actor	Usuario. (Inicia).
Propósito	Permite al usuario del sistema gestionar (obtener elemento que se encuentra en el frente, obtener elemento que se encuentra en el fondo, adicionar y extraer) los elementos que se encuentran almacenados en una cola.
Descripción	El CUS inicia cuando el usuario solicita gestionar los elementos que se encuentran almacenados en una cola, el sistema le pide el tipo de cola que desea emplear para la gestión de los elementos, el usuario selecciona el tipo de cola y proporciona los datos necesarios, el sistema realiza la acción seleccionada y termina el CUS.
Referencia	RF 1 (RF 1.3), RF 2 (RF 2.3), RF 3 (RF 3.3), RF 5 (RF 5.3), RF 6 (RF 6.3), RF 8 (RF 8.3).
Pos-condiciones	1- Obtenido el elemento que se encuentra en el frente de la cola. 2- Obtenido el elemento que se encuentra en el fondo de la cola. 3- Elementos adicionados en la cola. 4- Elementos extraídos de la cola.

Flujo Normal de Eventos	
Acción del actor	**Respuesta del sistema**
1– El usuario solicita gestionar los elementos en una cola.	1.1- El sistema permite escoger el tipo de cola con la cual se van a gestionar los elementos y muestra las opciones: - Cola de Prioridad con Arreglo. - Cola de Prioridad con listas enlazables. - Cola de Amigos con Arreglo. - Cola de Amigos con listas enlazables.

2- El usuario selecciona el tipo de cola.	2.1 – EL sistema le permite escoger el tipo de operación que desea realizar con la cola: - Obtener el primer elemento. - Obtener el último elemento. - Adicionar elementos. - Extraer elementos.
Sección 1: Obtener el primer elemento.	
Acción del actor	**Respuesta del sistema**
1 – El usuario selecciona la opción de *Obtener el primer elemento.*	1.1- El sistema verifica que la cola no esté vacía. 1.2- El sistema devuelve el elemento que se encuentra en la primera posición de la cola y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	1.1- Si en la cola no hay elemento alguno se lanza una excepción con un mensaje de error indicando que la cola está vacía.
Sección 2: Obtener el último elemento.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de *Obtener el último elemento.*	1.1- El sistema verifica que la cola no esté vacía. 1.2- El sistema devuelve el elemento que se encuentra en la última posición de la cola y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	1.1- Si en la cola no hay elemento alguno se lanza una excepción con un mensaje de error indicando que la cola está

	vacía.
Sección 3: Adicionar elementos.	
Acción del actor	**Respuesta del sistema**
1- El usuario selecciona la opción de *Adicionar elemento.*	1.1- El sistema coloca al elemento en la cola después del último. 1.2- Se incrementa en uno la longitud de la cola y termina el CUS.
Sección 4: Extraer elementos.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de *Extraer elemento.*	1.1- El sistema extrae el elemento que se encuentra en la cabeza de la cola. 1.2- Los elementos posteriores a partir de la primera posición pasan a tener la posición siguiente inmediata. 1.3- Ocurre un decremento en uno de la longitud de la cola 1.4- Se devuelve el elemento y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	1.1- Si la cola está vacía se lanza una excepción con un mensaje de error indicando que la cola está vacía.

Tabla 3.5 Descripción del Caso de Uso: Gestionar elementos en una Tabla Hash.

Nombre del CUS	Gestionar elementos en una Tabla Hash Abierta.
Actor	Usuario (Inicia).
Propósito	Permite al usuario del sistema gestionar (insertar, buscar y

	eliminar) los elementos que se encuentran almacenados en una tabla hash.
Descripción	El CUS inicia cuando el usuario solicita gestionar los elementos que se encuentran almacenados en una tabla hash, el sistema le pide el tipo de tabla hash que desea emplear para la gestión de los elementos, el usuario selecciona el tipo de tabla y proporciona los datos necesarios, el sistema realiza la acción seleccionada y termina el CUS.
Referencia	RF 4 (RF 4.1), RF 7 (RF 7.2), RF 8 (RF 8.4).
Pos-condiciones	1- Elementos insertados en la tabla hash. 2- Elementos buscados en la tabla hash. 3- Elementos eliminados de la tabla hash.

Flujo Normal de Eventos	
Acción del actor	**Respuesta del sistema**
1– El usuario solicita gestionar los elementos en una tabla hash.	1.1- El sistema permite escoger el tipo de tabla hash con la cual se van a gestionar los elementos y muestra las opciones: - Tabla Hash Abierta. - Tabla Hash Cerrada.
2- El usuario selecciona el tipo de tabla hash.	2.1 – EL sistema le permite escoger el tipo de operación que desea realizar con la tabla hash: - Insertar elementos. - Buscar elementos. - Eliminar elementos.

Sección 1: Insertar elementos.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de *Insertar elementos.*	1.1- El sistema pide introducir una clave.

2- El usuario introduce la clave.	2.1- El sistema calcula con la función de dispersión, el índice en la tabla donde se va a insertar el elemento con el valor de clave suministrado. 2.2- El sistema verifica si el índice calculado se encuentra en la tabla de dispersión. 2.3- El sistema Inserta el nuevo elemento en el índice de la tabla calculado y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	2.1- Si el índice calculado por la función de dispersión no se encuentra en la tabla se devuelve un valor falso.
Sección 2: Buscar elementos.	
Acción del actor	**Respuesta del sistema**
1- - El usuario selecciona la opción *Buscar elemento.* 2- - El usuario introduce la clave.	1.1- El sistema pide introducir una clave. 2.1 – El sistema calcula con la función de dispersión, el índice en la tabla donde se encuentra almacenado el elemento con el valor de clave suministrado. 2.2- El sistema verifica si el índice calculado se encuentra en la tabla de dispersión. 2.3- El sistema localiza el elemento que se encuentra almacenado en el índice calculado.

	2.4- Se muestra elemento buscado y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	2.2- Si el índice calculado por la función de dispersión no se encuentra en la tabla se lanza una excepción.
Sección 3: Eliminar elementos.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de *Eliminar elemento.* 2- - El usuario introduce la clave.	1.1- El sistema pide introducir una clave. 2.1 – El sistema calcula con la función de dispersión, el índice en la tabla hash donde se encuentra almacenado el elemento con el valor de clave suministrado. 2.2- El sistema verifica si el índice calculado se encuentra en la tabla de hash. 2.3- El sistema localiza el elemento que se encuentra almacenado en el índice calculado. 2.4- Se elimina el elemento y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	2.2- Si el índice calculado por la función de dispersión no se encuentra en la tabla se lanza una excepción.

Tabla 3.6 Descripción del Caso de Uso: Gestionar elementos en una DCEL.

Nombre del CUS	Gestionar elementos en una DCEL.
Actor	Usuario (Inicia).
Propósito	Permite al usuario del sistema gestionar (adicionar y obtener) los elementos en una DCEL.
Descripción	El CUS inicia cuando el usuario solicita gestionar los elementos que se encuentran almacenados en una DCEL, el usuario proporciona los datos necesarios, el sistema realiza la acción seleccionada y termina el CUS.
Referencia	RF 6 (RF 6.4), RF 5 (RF 5.5, RF 5.6, RF 5.7, RF 5.8).
Pos-condiciones	1- Adicionadas las aristas. 2- Obtenidas las aristas que pertenecen a una cara. 3- Obtenidas las aristas salientes de un vértice. 4- Obtenida la cara correspondiente a una arista. 5- Obtenidos los vértices correspondientes a una cara.
Flujo Normal de Eventos	
Acción del actor	**Respuesta del sistema**
1– El usuario solicita gestionar los elementos en una DCEL.	1.1- EL sistema le permite escoger el tipo de operación que desea realizar con la DCEL: - Adicionar aristas. - Obtener las aristas que pertenecen a una cara. - Obtener las aristas salientes de un vértice. - Obtener la cara correspondiente dada una arista. - Obtener los vértices correspondientes a una cara.
Flujos Alternos	

Sección 1: Adicionar Aristas.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de *Adicionar Aristas.*	1.1- El sistema verifica si la arista no existe. 1.2- Para cada vértice de los extremos de la arista, si existe el vértice, el sistema busca la arista que tiene como destino este vértice y que forma menor ángulo con la arista dada en sentido anti horario. 1.3- El sistema actualiza la lista de aristas doblemente enlazada, actualizando las aristas anterior y siguiente de las cuatro aristas involucradas (la arista que se va a insertar, su jimagua, la de menor ángulo y su siguiente) y las caras adyacentes de las nuevas aristas a insertar. 1.4- Si ambos vértices se encontraban en la lista de vértices, entonces el sistema asigna una de las nuevas aristas como arista exterior de la cara en la que se está insertando. 1.5- El sistema crea una nueva cara y se le asigna como arista exterior la arista jimagua. 1.6- El sistema recorre la lista de aristas a partir de la arista jimagua hasta llegar a

	ella misma, asignándole la nueva cara como cara adyacente. 1.7- El sistema inserta la nueva cara formada a la lista de caras. 1.8- El sistema inserta en la lista de arista: la arista y su arista jimagua y termina el CUS.
Flujos Alternos	
Curso Alternativo de Eventos	1.1- Si existe la arista se lanza una excepción. 1.2- Si no existe el vértice se adiciona a la lista de vértices.
Sección 2: Obtener las aristas que pertenecen a una cara.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de *Obtener las aristas que pertenecen a una cara.*	1.1- El sistema pide la arista siguiente dada una arista y así sucesivamente hasta llegar al vértice de origen de la arista dada. 1.2- El sistema devuelve una lista con las aristas que pertenecen a la cara dada y termina el CUS.
Sección 3: Obtener las aristas salientes de un vértice.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de *Obtener las aristas salientes de un vértice*	1.1- Dada una arista saliente de un vértice, el sistema le pide su arista jimagua. 1.2- El sistema le pide a esta arista jimagua su arista siguiente, que sería la otra arista saliente y así sucesivamente.

	1.3- El sistema devuelve una lista con las aristas salientes del vértice dado y termina el CUS.
Sección 4: Obtener la cara correspondiente dada una arista.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de *Obtener la cara correspondiente dada una arista*	1.1- El sistema pide la arista siguiente dada una arista y así sucesivamente hasta llegar al vértice de origen de la arista dada. 1.2- El sistema devuelve la cara a la que pertenecen las aristas y termina el CUS.
Sección 5: Obtener los vértices correspondientes a una cara.	
Acción del actor	**Respuesta del sistema**
1 - El usuario selecciona la opción de *Obtener los vértices correspondientes a una cara.*	1.1- El sistema dado una cara, a una de sus aristas se le pide su vértice de destino y así sucesivamente hasta llegar al vértice de destino de la arista dada. 1.2- El sistema devuelve una lista con los vértices correspondientes a esa cara y termina el CUS.

3.5 Conclusiones parciales.

En el estudio realizado de este capítulo se ha visto cómo en el Flujo de Trabajo de Requerimientos existen dos actividades fundamentales: la captura de los requerimientos y el modelado del sistema. Los casos de uso del sistema se forman agrupando los requerimientos funcionales. Los requerimientos deben ser verificados y validados para evitar errores que pueden resultar altamente costosos en una etapa posterior del desarrollo del sistema.

Capítulo 4

Construcción de la solución propuesta.

4.1 Introducción.

En este capítulo se construye la solución propuesta con la realización del diseño, la implementación y las pruebas del sistema en los flujos de trabajo correspondientes. Se representan los modelos y diagramas que contribuyen a la comprensión y posterior realización del producto y se proponen con un ejemplo las pruebas que se le deben hacer al componente de sistema para comprobar la calidad del mismo.

4.2 Patrones.

Los patrones no se proponen descubrir ni expresar nuevos principios de la ingeniería de software, todo lo contrario, intentan codificar el conocimiento, las expresiones y los principios ya existentes.(28)

Según la escala o nivel de abstracción los patrones se clasifican en:

- Patrones de arquitectura.
- Patrones de diseño.

4.2.1 Patrón de arquitectura.

El papel de la arquitectura es proporcionarle información del diseño a los desarrolladores para que puedan hacer cambios y correcciones al software sin romper la arquitectura. Un patrón de arquitectura es aquel que expresa un esquema organizativo estructural fundamental para sistemas de software.(28)

El patrón de arquitectura empleado fue la Arquitectura Orientada a Objetos. Toda Arquitectura Orientada a Objetos caracteriza un modelo de objeto que describe cómo se representan los objetos en el sistema. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos.

Características principales de la Arquitectura Orientada a Objetos.

- Los componentes del estilo se basan en los principios orientados a objetos, tales como: encapsulamiento, herencia y polimorfismo.
- Los objetos y sus interacciones son el centro en el diseño de la arquitectura y en la estructura de la aplicación.
- En general la distribución de los objetos es transparente y en el estado del arte de la tecnología apenas importa si los objetos son locales o remotos.

Ventajas de la Arquitectura Orientada a Objetos.

- Los componentes de software permiten a terceros, el desarrollo del software fácilmente acoplable al sistema. Así, la arquitectura orientada a objetos conjuntamente con los componentes de software constituyen un sistema versátil y robusto que puede dar soporte al desarrollo rápido de aplicaciones informáticas con tecnologías orientadas a objetos. Con este soporte se desarrollan sistemas operativos orientados a objetos, lenguajes orientados a objetos, sistemas de gestión de bases de datos orientados a objetos, etc. Además esta arquitectura constituye un buen laboratorio de pruebas de robustez de sistemas de software complejos.
- Se puede modificar la implementación de un objeto sin afectar a sus clientes. Así mismo es posible descomponer problemas de colecciones de agentes en

interacción. Además, un objeto es ante todo una entidad reutilizable en el entorno de desarrollo.

Desventaja de la Arquitectura Orientada a Objetos.

» El principal problema de este estilo se manifiesta en el hecho de que para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad.(29)

4.2.2 Patrones de diseño.

Los patrones de diseño proponen una manera de reutilizar la experiencia de los expertos, para ello se clasifican y describen formas para solucionar problemas que ocurren frecuentemente en el desarrollo de un software. (28)

¿Qué son los patrones GRASP?

Los patrones GRASP[14] describen los principios fundamentales de la asignación de responsabilidades a los objetos expresados en forma de patrones. Es importante entender y poder aplicar estos principios durante la elaboración de un diagrama de interacción, pues un diseñador de software sin mucha experiencia en la tecnología de objetos debe dominarlos cuanto antes ya que constituyen el fundamento de cómo se diseñará el sistema.(30). En el diseño del componente de software se tuvieron en cuenta principalmente los patrones que se muestran a continuación:

» **Experto:** Consiste en asignar una responsabilidad al experto en información: el módulo que cuenta con la información necesaria para cumplir la responsabilidad. Una clase, contiene toda la información necesaria para realizar la labor que tiene

[14] GRASP es el acrónimo correspondiente a *General Responsibility Assignment Software Patterns*, cuya traducción literal sería: Patrones Generales de Software para Asignar Responsabilidades.

encomendada. Hay que tener en cuenta que esto es aplicable mientras se esté considerando los mismos aspectos del sistema:

- Lógica de negocio
- Persistencia a la base de datos
- Interfaz de usuario(31)

Beneficios:

- ✓ Se conserva el encapsulamiento ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento.

- ✓ El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y más cohesivas que son más fáciles de comprender y de mantener. Así se brinda soporte a una alta cohesión.

» **Creador:** Consiste en asignar a una clase la responsabilidad de crear una instancia de otra clase. Guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Al escogerlo como creador, puede soportar un bajo acoplamiento, una mayor claridad, el encapsulamiento y la reutilización.

Beneficios:

- ✓ Brinda soporte a un bajo acoplamiento, lo cual supone menos dependencias y mejores oportunidades de reutilización. Es probable que el acoplamiento no aumente, pues la clase creada tiende a ser visible a la clase creador, debido a las asociaciones actuales que conllevaron a elegirla como el parámetro adecuado.

» **Bajo Acoplamiento:** Consiste en asignar una responsabilidad para mantener bajo el acoplamiento. Debe haber pocas dependencias entre las clases. Para determinar el nivel de acoplamiento de clases, son muy buenos los diagramas de colaboración. Uno de los principales síntomas de un mal diseño y alto acoplamiento es una herencia muy profunda. Siempre hay que considerar las ventajas de la delegación respecto de la herencia.(31)

Beneficios:

- ✓ No se afectan por cambios de otros componentes.
- ✓ Fáciles de entender por separado.
- ✓ Fáciles de reutilizar.

» **Alta Cohesión:** Consiste en asignar una responsabilidad para mantener alta la cohesión. En cuanto al diseño de objetos, la cohesión o de manera más específica la cohesión funcional, es una medida de la fuerza con la que se relacionan y del grado de focalización de las responsabilidades de un elemento. Un elemento con responsabilidad altamente relacionada, y que no hace una gran cantidad de trabajo tiene alta cohesión. A menudo, las clases con baja cohesión representan un "grano grande" de abstracción, o se les ha asignado responsabilidades que debería haberse delegado en otros objetos.

Beneficios:

- ✓ Mejoran la claridad y la facilidad con que se entiende el diseño.
- ✓ Se simplifican el mantenimiento y las mejoras en funcionalidad.
- ✓ A menudo se genera un bajo acoplamiento.
- ✓ La ventaja de una gran funcionalidad soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.(32)

» **Polimorfismo:** El polimorfismo significa "asignar el mismo nombre a servicios en varios objetos". Cuando por el tipo varían las alternativas o comportamientos afines, las responsabilidades del comportamiento se asignarán mediante operaciones polimórficas a los tipos en que el comportamiento presenta variantes. El uso del patrón Polimorfismo está acorde al espíritu del patrón *Experto*. Si podemos caracterizar a *Experto* como el patrón fundamental táctico, Polimorfismo será el más importante patrón estratégico en el diseño orientado a objetos.

Beneficios:

- ✓ Es fácil agregar las futuras extensiones que requieren las variaciones imprevistas.

4.3 Diagramas de clases del Diseño.

El Modelo de Diseño es un modelo de objetos que describe la realización física de los casos de uso, constituye la entrada principal en el Flujo de Trabajo de Implementación. Para la modelación de la solución se proponen como clases del diseño los estándares para modelar *aplicaciones Desktop.* (13)

De acuerdo a la forma en que se ha estructurado el contenido del trabajo, se muestran los modelos organizados por paquetes de manera que se pueda entender mejor la lógica del problema. En la Figura 4.1 se observa una vista general del diagrama del diseño, y más adelante se muestra una vista más detallada de cada uno de los paquetes.

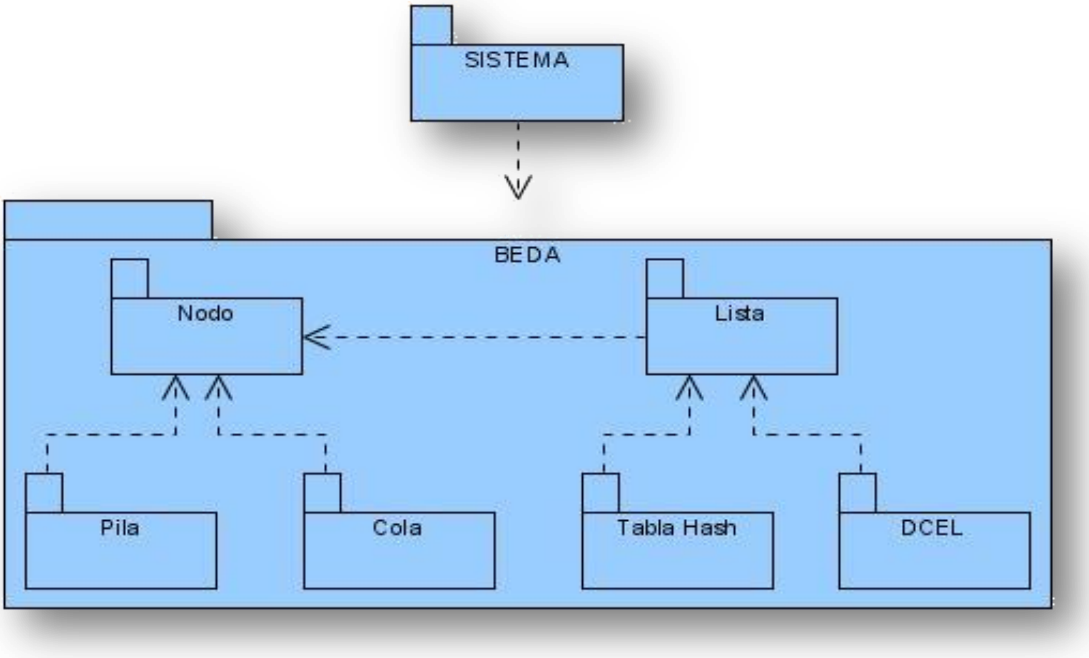


Figura 4.1 Diagrama general del diseño organizado en paquetes.

4.3.1 Paquete Nodo.

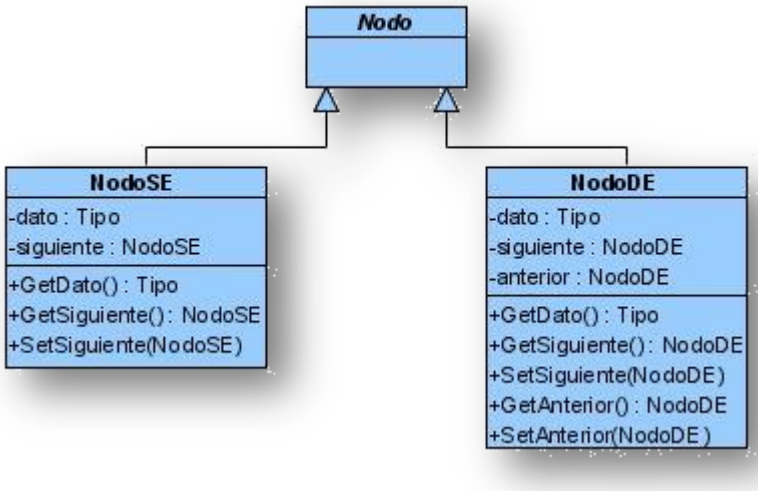


Figura 4.2 Diagrama de clases del diseño del paquete Nodo.

4.3.2 Paquete Lista.

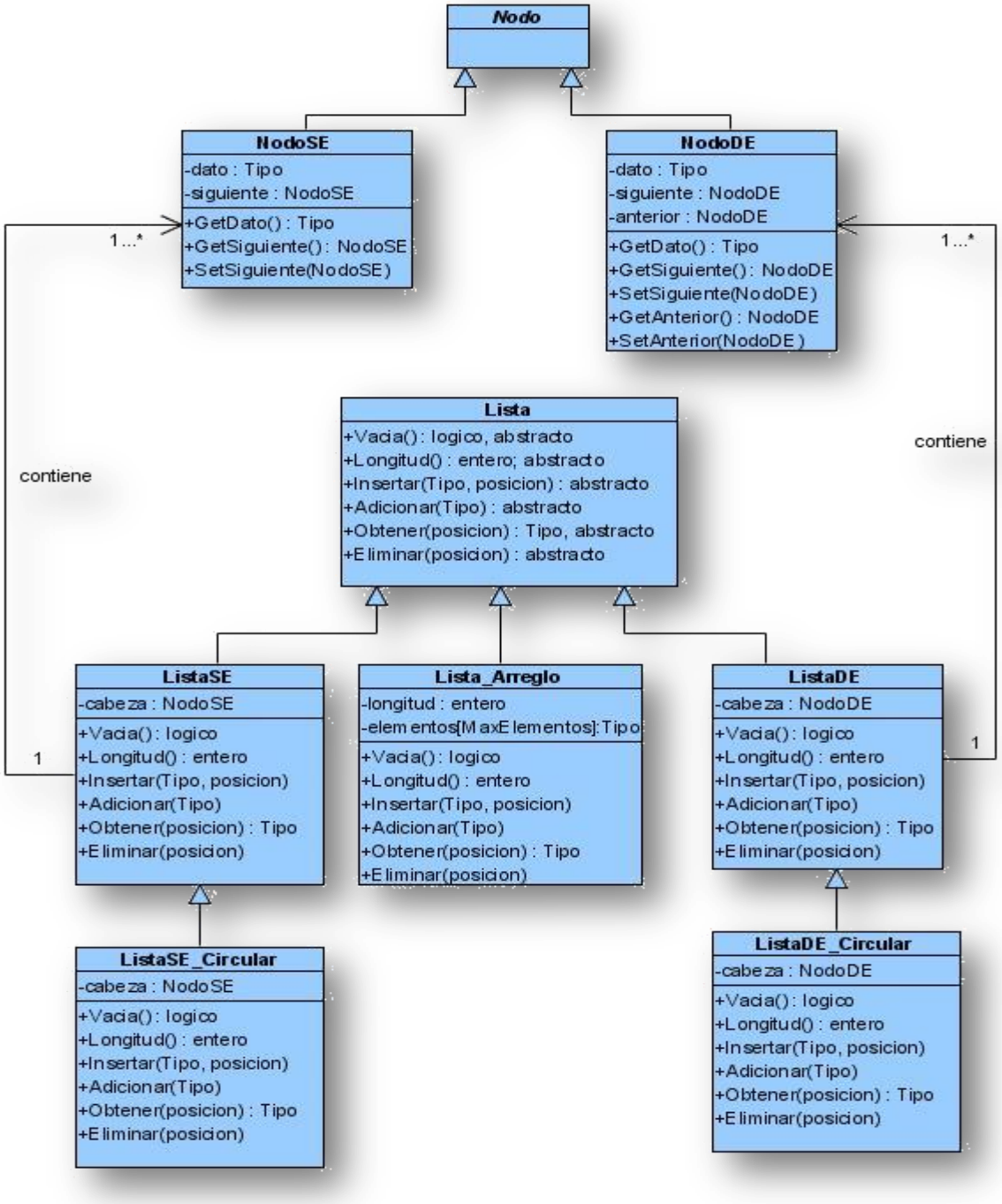


Figura 4.3 Diagrama de clases del diseño del paquete Lista.

4.3.3 Paquete Cola.

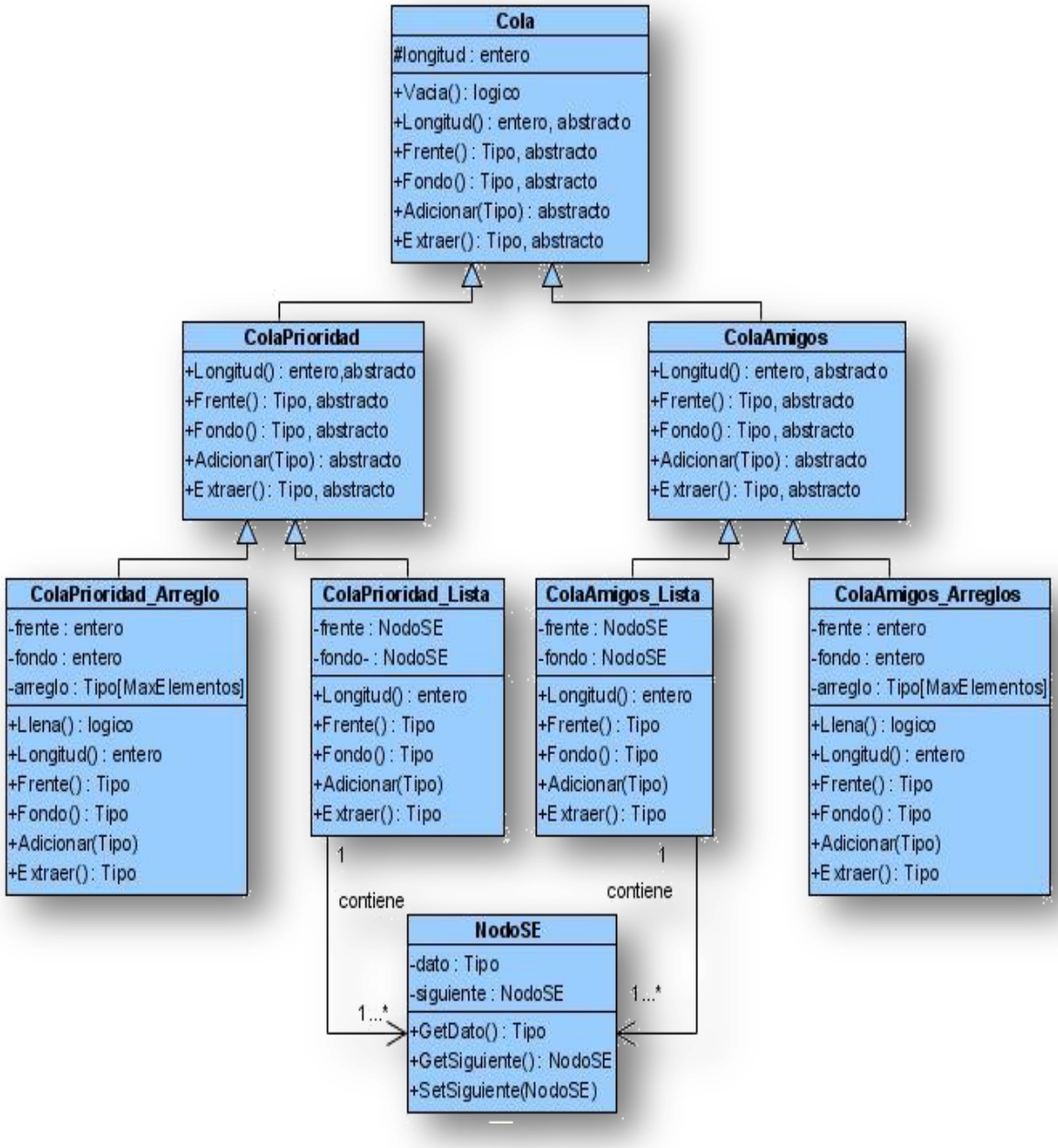


Figura 4.4 Diagrama de clases del diseño del paquete Cola.

4.3.4 Paquete Pila.

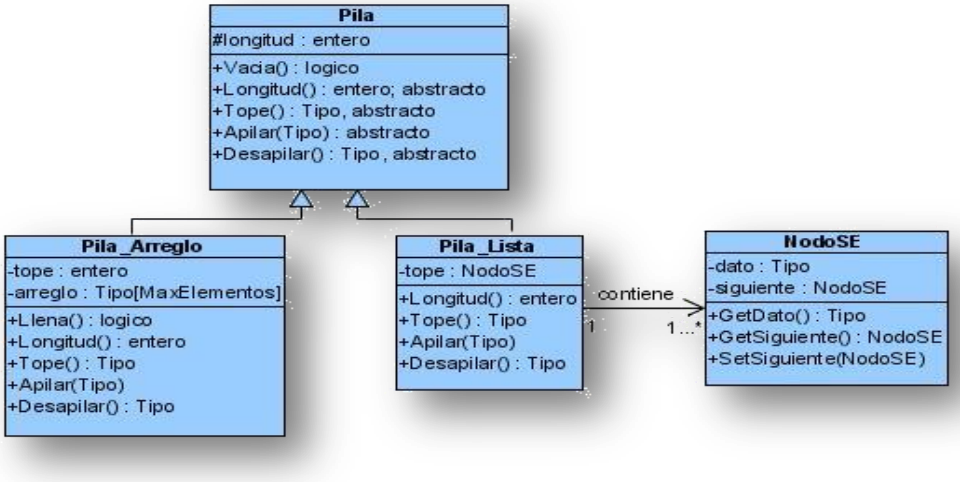


Figura 4.5 Diagrama de clases del diseño del paquete Pila.

4.3.5 Paquete Tabla Hash.

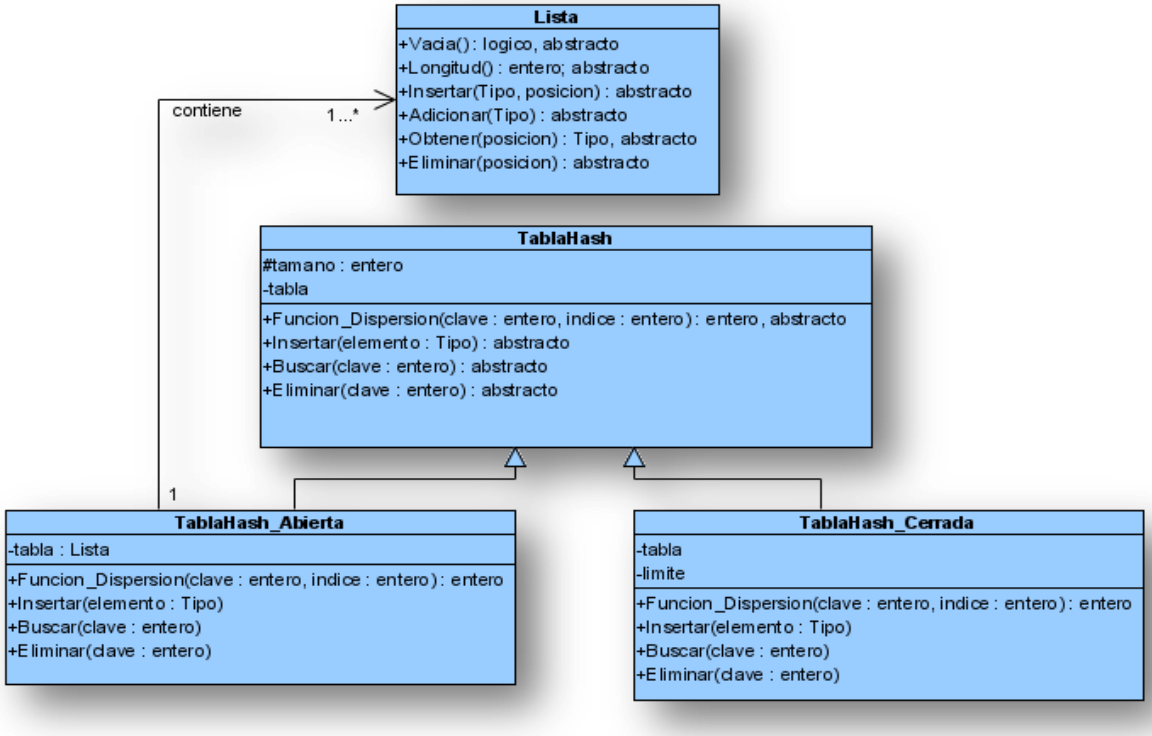


Figura 4.6 Diagrama de clases del diseño del paquete Tabla Hash.

4.3.6 Paquete DCEL.

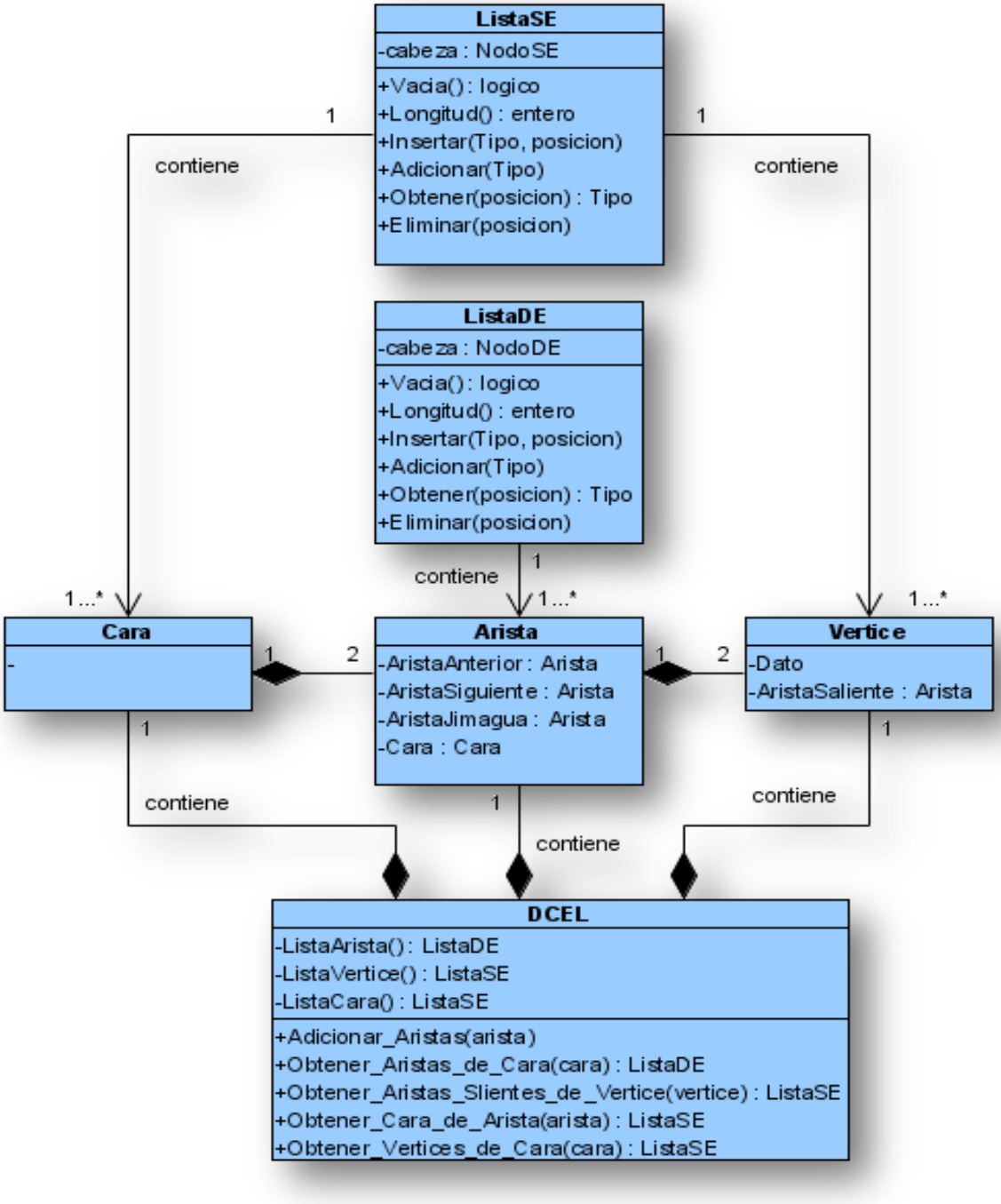


Figura 4.7 Diagrama de clases del diseño del paquete DCEL.

4.4 Diagramas de Interacción.

Un diagrama de interacción consiste en un conjunto de objetos y sus relaciones, incluyendo los mensajes que se pueden enviar entre ellos.
Los diagramas de colaboración destacan la organización estructural de los objetos que envían y reciben mensajes. Se utilizan frecuentemente en la fase de diseño, es decir, cuando se diseña la implementación. Los diagramas de secuencia destacan el orden temporal de los mensajes.
Los diagramas de colaboración y de secuencia (llamados diagramas de interacción) se utilizan para modelar los aspectos dinámicos de un sistema. Son isomorfos, es decir, se puede convertir de uno a otro sin pérdida de información.
Para consultar los Diagramas de Interacción, remitirse al ANEXO 2.

4.5 Modelo de Implementación.

En la implementación se empieza con el resultado del diseño y se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares, describe también como se organizan y se relacionan unos con otros. (33)

4.5.1 Diagrama de Componentes.

Los diagramas de componentes se utilizan para mostrar las dependencias de compilación de los ficheros de código, relaciones de derivación entre ficheros de código fuente y ficheros que son resultados de la compilación, dependencias entre los elementos de implementación y los elementos correspondientes del diseño que son implementados.(33)

Si se tienen en cuenta las dependencias asociadas al proceso de compilación, un componente podría ser un código ejecutable que puede depender de otros programas ejecutables con los que interactúa en tiempo de ejecución.

Una biblioteca es una combinación de estos ficheros, y al mostrar las dependencias entre ellos, se obtiene una visión de las partes necesarias para la creación de la biblioteca.

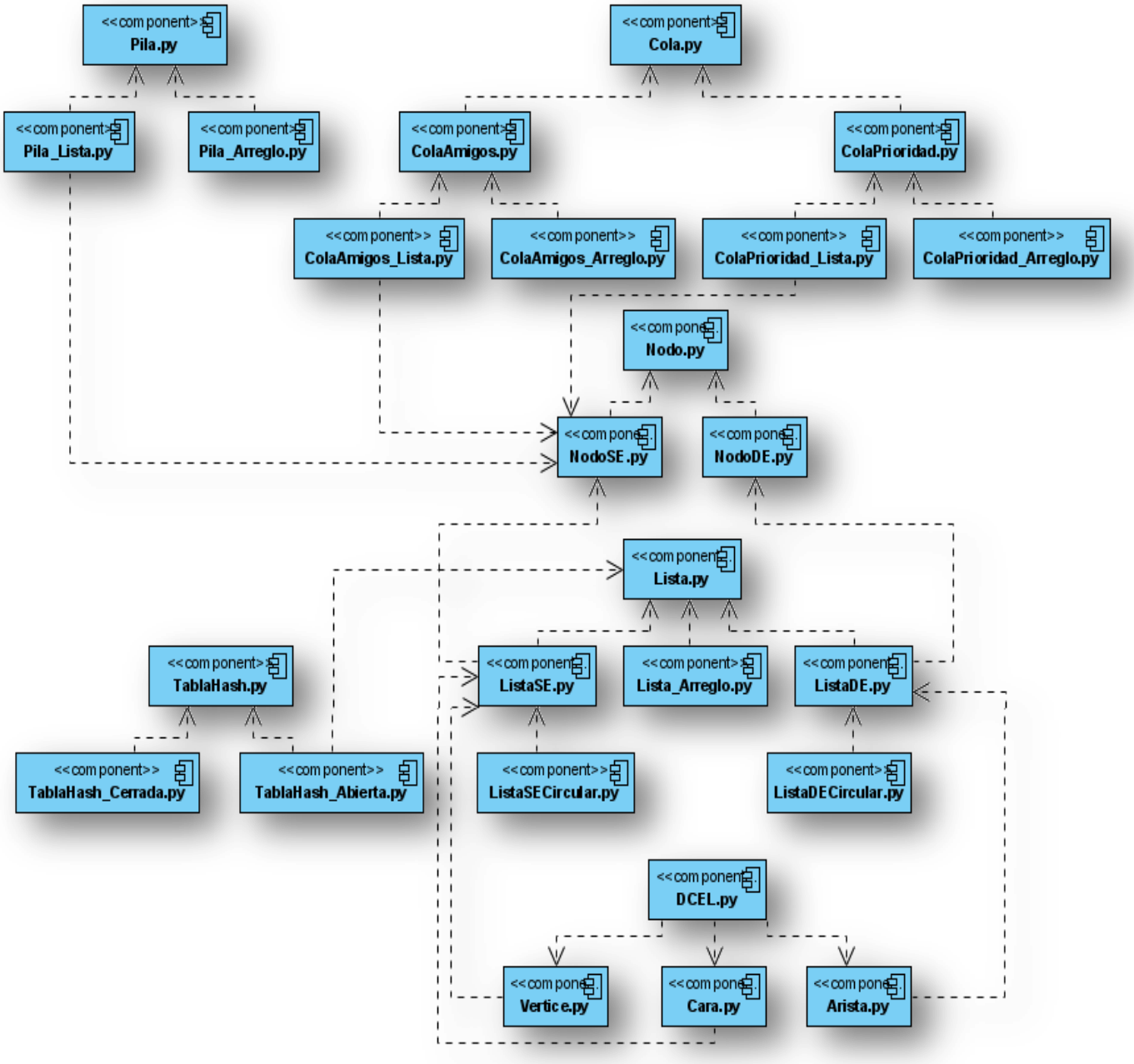


Figura 4.8 Diagrama de Componentes.

4.6 Técnicas de Pruebas.

Las pruebas del software son un elemento crítico para garantizar la calidad del mismo y representa una revisión final de las especificaciones del diseño y de la codificación.

4.6.1 Pruebas de Caja Blanca.

Las pruebas de Caja Blanca se basan en un minucioso examen de los detalles procedimentales del código a evaluar, por lo que es necesario conocer la lógica del programa. La Complejidad Ciclomática es una métrica del software que proporciona una medida cuantitativa de la complejidad lógica de un programa. En el contexto del método de prueba del camino básico, el valor de la complejidad ciclomática define el número de caminos independientes de dicho programa, y por lo tanto, el número de casos de prueba a realizar para garantizar que todas las sentencias de un programa se han ejecutado al menos una vez, y que cada condición se habrá ejecutado en sus vertientes verdaderas y falsas.

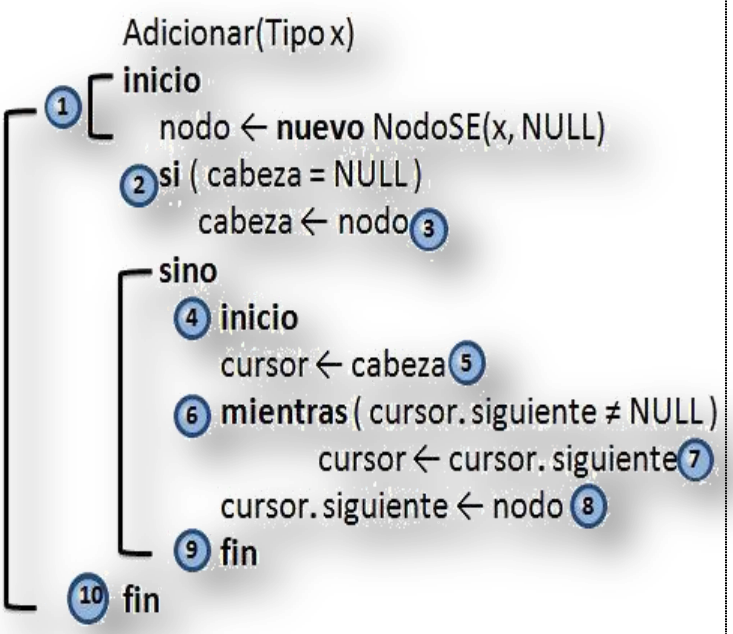
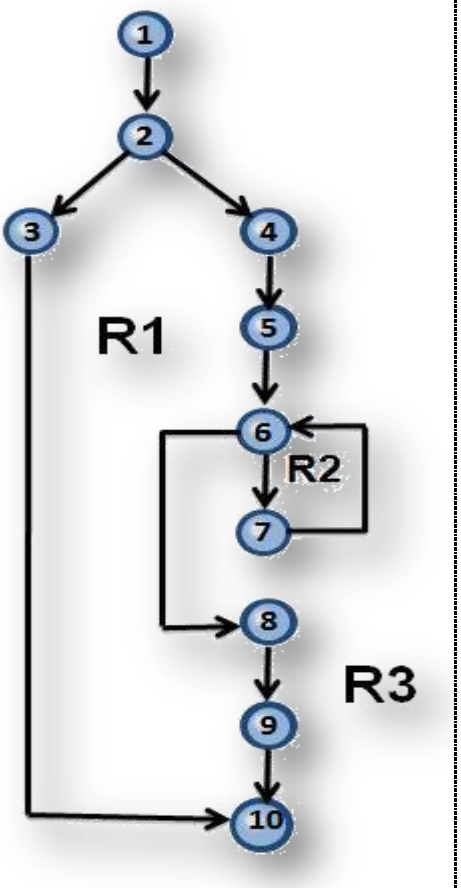
Notación del Grafo de Flujo:

- Cada círculo denominado nodo del grafo de flujo, representa una o más sentencias procedimentales.
- Las flechas representan el flujo de control.
- Las regiones son aéreas delimitadas por aristas y nodos.

La complejidad ciclomática, **V (G)**, de un grafo de flujo **G** se define como:

V (G) = número de regiones O **V (G) = Aristas – Nodos + 2**

Figura 4.9 Prueba del camino básico para el CU: Gestionar elementos en una Lista, escenario: Adicionar elemento.

Seudocódigo	Grafo de Flujo
	

El grafo tiene tres regiones por lo que el camino básico estará formado por tres caminos en el programa. Estos caminos son:

Camino 1: 1, 2, 3, 10

Camino 2: 1, 2, 4, 5, 6, 8, 9, 10.

Camino 3: 1, 2, 4, 5, 6, 7, 6, 8, 9,10.

Posibles casos de prueba que se pueden generar para probar estos caminos.

Número del Camino	Caso de Prueba	Objetivo	Resultado Esperado
1	elemento = 5	Probar adicionar un elemento cuando la lista está vacía.	Elemento adicionado correctamente
2	elemento = 6	Probar adicionar un elemento seguidamente del primero.	Elemento adicionado correctamente
3	elemento = 7	Probar adicionar un elemento independientemente de los elementos que se encuentren en la lista.	Elemento adicionado correctamente

Para consultar algunos de los casos de prueba, remitirse al ANEXO 3.

4.7 Conclusiones parciales.

En el flujo de trabajo de Diseño, el sistema se modela de modo que soporte todos los requisitos, tanto funcionales como no funcionales, creándose así una entrada apropiada para el flujo de trabajo de Implementación. Como resultado del estudio realizado en este capítulo correspondiente a las etapas de diseño, implementación y prueba del sistema, se modelaron los diagramas de clases agrupados por paquetes y el diagrama de componentes correspondiente al flujo de trabajo de Implementación. Se utilizó como técnica de prueba a las pruebas de Caja Blanca que se basan en un minucioso examen de los detalles procedimentales del código a evaluar.

Conclusiones.

La investigación y el estudio siempre han sido los protagonistas de los buenos resultados de un proyecto. Durante el desarrollo de este trabajo se realizó una investigación exhaustiva del estudio del estado del arte de las diferentes estructuras de datos que conforman a la biblioteca y se analizó la existencia de otras posibles soluciones al problema científico planteado.

Se demostró la eficiencia de las tendencias y las tecnologías actuales utilizadas para el desarrollo de la Biblioteca de Estructuras de Datos Avanzadas cubriéndose los pasos propuestos por *RUP* como metodología de desarrollo de software la cual le dio soporte al lenguaje de modelado *UML* que se empleó para la modelación de los diferentes diagramas, se utilizó además al *Visual Paradigm* como herramienta *CASE* y como lenguaje de programación a *Python* para la implementación de las diferentes estructuras de datos, además de otras herramientas de apoyo que aseguraron otros aspectos no menos importantes.

Se realizó el diseño, la implementación y las pruebas del sistema cumpliéndose con el objetivo general propuesto, reafirmando así la utilidad y validez de emplear las tecnologías informáticas para apoyar las labores que se desarrollan en cualquier esfera del desarrollo social.

Finalmente se concluye que durante la realización del presente trabajo se cumplió con el objetivo general propuesto: “desarrollar en una plataforma de Software Libre una Biblioteca de Estructuras de Datos Avanzadas”, específicamente con las estructuras de datos: Listas, Pilas, Colas, Tablas Hash y DCEL, en algunas de sus variantes de implementación; conjuntamente se elaboró un documento de apoyo que servirá de ayuda a los usuarios de la biblioteca.

Recomendaciones.

A partir de los resultados y beneficios que proporciona el presente trabajo de diploma, se recomienda:

- Continuar con el estudio y desarrollo de estas y otras estructuras de datos con el objetivo de enriquecer aún más la Biblioteca de Estructuras de Datos Avanzadas.

- Aumentar las funcionalidades de cada una de las estructuras de datos que conforman la Biblioteca de Estructuras de Datos Avanzadas.

Referencias Bibliográficas.

1. PROGRAMACIÓN, D. D. T. D. *P2. Conferencia #1. Estructuras de Datos Lineales. Los Tipos de Datos Abstractos. El TDA Lista.* 2006-2007.

2. ---. *P2. Conferencia #2. Estructuras de Datos Lineales. Implementación del TDA Lista utilizando listas enlazadas.* 2006-2007.

3. ---. *P2. Conferencia #3. Estructuras de Datos Lineales. Los TDA Pila y Cola* 2006-2007.

4. GREGORY, H. *Estructuras de datos, algoritmos y programación orientada a objetos.* La Habana: Félix Varela, 2003.

5. MICTLAN. *Página de entrenamiento para el ACM ICPC de la Universidad Tecnológica de la Mixteca. Diccionarios,* de 2008]. Disponible en: http://mictlan.utm.mx/html/jaws/html/index.php?page/diccionarios.

6. FERRERA, J. C. *Diagramas de Voronoi* Facultad de Informática. Universidad Politécnica de Madrid: Disponible en: http://www.dma.fi.upm.es/mabellanas/voronoi/voronoi/voronoi.html.

7. SECO, J. A. G. *El lenguaje de programación C#. Introducción a Microsoft.NET. Librería de clase base (BCL)* Disponible en: http://programmatium.blogspot.com/2008/01/el-lenguaje-de-programacin-c_4609.html#apartado6.

8. ---. Clikear.com - Portal para desarrolladores de la plataforma Microsoft.NET. Framework 2001, nº

9. BOOCH, G., RUMBAUGH, J. Y JACOBSON I. *"El Lenguaje Unificado de Modelado".* 2000.

10. ORALLO, H. *El Lenguaje Unificado de Modelado (UML).* Universidad Politécnica de Valencia, de 2008]. Disponible en: http://www.disca.upv.es/enheror/pdf/ActaUML.PDF.

11. LARMAN, C. *"UML y Patrones. Introducción al análisis y el diseño orientado a objetos"* Disponible en: http://bibliodoc.uci.cu/pdf/reg00061.pdf.

12. TECNOLOGIAS, F. E. D. C. Y. *Uso de UML en el modelado de datos.* Universidad de Carabobo: Disponible en: http://alfa.facyt.uc.edu.ve/computacion/pensum/cs0347/download/exposiciones2005-2006/uml.pdf.

13. JACOBSON, I., BOOCH G. Y RUMBAUGH J. "El Proceso Unificado de Desarrollo de Software". 2000, nº [Consultado el: 2008]. Disponible en: http://bibliodoc.uci.cu/pdf/reg00060.pdf.

14. SOFTWARE, D. D. I. Y. G. D. ISW1. Conferencia #1. Introducción a la Ingeniería de Software. De *2007-2008*.

15. WELLS, D. *Extreme Programming: A gentle introduction* de 2008]. Disponible en: http://www.extremeprogramming.org/.

16. BECK, K. *Extreme Programming Explined. Embrace change. s.l.* Pearson Education, 1999.

17. COAD, P., LEFEBRE E., DE LUCA J. *"Java Modeling in color with UML: Enterprise Components and Process".* Prentice Hall, 1999.

18. MOLPECERES, A. *"Procesos de desarrollo: RUP, XP y FDD"* de 2008]. Disponible en: http://www.javahispano.org/articles.article.action?id=76.

19. PARADIGM, V. *Build Quality Applications Faster, Better and Cheaper* [Consultado el: 2008 Disponible en: http://www.visual-paradigm.com/product/vpuml/index.jsp.

20. ---. *UML Modeling Tool: Visual Paradigm for UML 4.1* [Consultado el: 2008 Disponible en: http://c2.com/cgi/wiki?VisualParadigm.

21. GONZÁLEZ, C. I. *"El lenguaje C++".*

22. JAVA, M. D. *Características de Java* Disponible en: http://www.manual-java.com/manualjava/caracteristicas-java.html.

23. KUHLMAN, D. *Python 101 - Introduction to Python* de 2008]. Disponible en: http://www.rexx.com/~dkuhlman/python_101/python_101.html.

24. IPIÑA, G. D. A., DIEGO LZ., PROFESOR DEL DEPARTAMENTO DE INGENIERÍA DE SOFTWARE. *"Pensando en Python (I):3 en raya en modo texto".* Facultad de Ingeniería (ESIDE) de la Universidad de Deusto.

25. GMBH, C. *Opiniones de Adobe Photoshop CS* Disponible en: http://www.ciao.es/Adobe_Photoshop_CS__376928.

26. SHOP, S. *Información acerca de EndNote* Disponible en: http://www.software-shop.com/in.php?mod=ver_producto&prdID=102

27. SOFTWARE, D. D. I. Y. G. D. *ISW1. Conferencia #3. Flujo de trabajo de requerimientos.* 2007-2008.

28. ---. ISW2. Conferencia #8. Arquitectura y Patrones de diseño. En *2007-2008*.

29. PROGRAMACIÓN, D. D. S. I. Y. Curso de doctorado. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. Patrones de diseño orientado a objetos. Diseños de Arquitecturas de Software Orientadas a Objetos. 1999-2000, nº

30. REYES, R. A. M. *Los Patrones como un Medio del Diseño Orientado a Objetos* Disponible en: http://www.revistaupiicsa.20m.com/Emilia/RevMayAgo04/Machorro1.pdf.

31. GUTIERREZ, J. A. S. *Ingeniería de Sistemas. Patrones Grasp (Craig Larman) Parte I* Disponible en: http://jorgesaavedra.wordpress.com/2006/08/17/patrones-grasp-craig-larman/.

32. RIOS, S. S. *Metodologías de Análisis y Diseño. Unidad VII Diseño O.O – Diagramas de Interacción "Patrones de Diseño"* Universidad Viña del Mar: Disponible en: http://www.uvmsf.cl/~ssanchez/images/Metodologias/Unidad8_MAD.pdf.

33. SOFTWARE, D. D. I. Y. G. D. *ISW2. Conferencia #4. Flujo de Trabajo de Implementación.* 2007-2008.

Glosario de términos y siglas.

A

Acoplamiento: El acoplamiento mide qué tan fuerte está una clase conectada con otras (es decir, cuántas clases conoce y necesita). Una clase con bajo o débil acoplamiento no depende de muchas otras clases. Una clase con alto (o fuerte) acoplamiento recurre a muchas otras clases Este tipo de clase no es conveniente, pues: cambios en las clases relacionadas ocasionan cambios en la clase local; son más difíciles de entender y de reutilizar.

Apilar: Operación en el que un elemento de datos se coloca en el lugar apuntado por el puntero de la pila, y la dirección en el puntero de la pila se ajusta por el tamaño de los datos de partida.

Arreglo: O como también se le conoce: vector, *array*, o alineación, en programación no es más que un conjunto o agrupación de variables del mismo tipo cuyo acceso se realiza por índices. Se almacenan en forma contigua en la memoria RAM y son referenciados con un nombre común y una posición relativa.

B

Bits: Es el acrónimo de **B**inary dig**it**. (Dígito binario). Un bit es un dígito del sistema de numeración binario.

Bytes: Secuencia de bits contiguos, cuyo tamaño depende del código de información o código de caracteres en que sea definido. La unidad byte se representa con el símbolo “B”.

Bytecode: Es un código intermedio más abstracto que el código máquina. Habitualmente es tratado como un fichero binario que contiene un programa ejecutable similar a un módulo objeto, que es un fichero binario producido por el compilador cuyo contenido es el código objeto o código máquina. El *bytecode* recibe su nombre porque generalmente cada código de operación tiene una longitud de un byte, si bien la longitud del código de las instrucciones varía.

C

Caso de Uso: Operación o tarea específica que se realiza tras una orden de algún agente externo, sea desde una petición de un actor o bien desde la invocación desde otro caso de uso. Por otra parte un Actor no es como tal, parte del sistema, sino que es un Rol de un usuario donde este puede intercambiar información y representa a un ser humano, software o a una máquina que interactúa con el sistema.

D

DEBIAN: Asociación o comunidad conformada por desarrolladores y usuarios que pretenden crear y mantener un Sistema Operativo *GNU* basado en Software Libre pre compilado y empaquetado en un formato sencillo en múltiples arquitecturas y en varios núcleos.

Desapilar: Operación en el que un elemento de datos en la ubicación actual apuntada por el puntero de la pila es eliminado, y el puntero de la pila se ajusta por el tamaño de los datos de partida.

Datos: es una representación simbólica (numérica, alfabética, etc.), atributo o característica de una entidad. El dato no tiene valor semántico (sentido) en sí mismo.

E

Encapsulamiento: Esta abstracción consiste en ocultar las características de un objeto por separado y obviarlas, de tal forma que solo se utiliza el nombre de dicho objeto en el programa.

Estructura de datos: son colecciones de variables, no necesariamente del mismo tipo, relacionadas entre sí de alguna forma, sobre la que se han implementado las operaciones definidas para el TDA y que permite finalmente la implementación de un tipo de dato (a través de una clase).

F

Función de hash: es una función para resumir o identificar probabilísticamente un gran conjunto de información, dando como resultado un conjunto imagen finito generalmente menor (por ejemplo: un subconjunto de los números naturales). Una buena función hash es una que experimenta pocas colisiones en el conjunto esperado de entrada; es decir que se podrán identificar unívocamente las entradas.

G

GNU: Conjunto de programas desarrollados por miembros de la Fundación por el Software Libre, son de uso gratuito (FSF- *Free Software Foundation*).

GNU/LINUX: Es un sistema operativo. Es una implementación de libre distribución *UNIX* para computadoras personales (PC), servidores, y estaciones de trabajo. Es multitarea, multiusuario, multiplataforma y multiprocesador.

GPL: Es una licencia creada por la Free Software Foundation y orientada principalmente a los términos de distribución, modificación y uso del software. Su propósito es declarar que el software cubierto por esta licencia es Software Libre (*General Public License*).

H

Herramientas: Software que se utiliza para automatizar las actividades definidas en el proceso.

L

Linux: Linux es el núcleo del sistema operativo libre más popular y constituye la base del sistema operativo Linux (también llamado GNU/Linux), que ha comenzado a competir con sistemas operativos no libres como Unix y Windows.

N

Números primos: Es un subconjunto de los números naturales que engloba a todos los elementos de este conjunto mayores que uno que son divisibles únicamente por sí mismos y por la unidad.

P

Prueba: Proceso de ejecución de un programa con la intención de descubrir errores.

Programación Orientada a Objetos: Los conceptos fundamentales de este paradigma son: Objetos, Clases, Polimorfismo, Encapsulamiento, Envío de mensajes, Asociaciones, Herencia, Agregación & Composición.

R

RUP: *Rational Unified Process* (Proceso Unificado de Desarrollo) es un proceso de desarrollo de software y junto con el UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

T

Tipo de dato abstracto (TDA) o Tipo abstracto de datos (TAD): Es un modelo matemático compuesto por una colección de operaciones definidas sobre un conjunto de datos para el modelo.

U

UML: *Unified Modeling Language* (Lenguaje Unificado de Modelado) es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad.

UNIX: Sistema Operativo portable, multitarea y multiusuario.

Anexos.

ANEXO 1 Entrevistas realizadas a estudiantes y a jefes de los diversos proyectos productivos.

Resultado del Método Empírico: *Entrevistas*, las cuales se les realizaron a los integrantes y a los jefes de diferentes proyectos productivos con el fin de investigar y obtener datos concretos acerca del uso de las estructuras de datos avanzadas en la universidad. En las entrevistas realizadas se seleccionó como **población** a los proyectos productivos que trabajan con Software Libre en la universidad, de los cuales se tomó una **muestra** de cuatro proyectos, arrojándose los resultados que se esperaban. Estos proyectos son:

- Informatización de los Servicios de la Biblioteca Nacional.
- Informatización de los Servicios de la Biblioteca UCI.
- Proyecto “Unicornio”.
- Proyecto SCADA[15].

Entrevistas realizadas a estudiantes y jefes de proyectos productivos de la facultad 10.

Fecha: 27/11/07

Nombre del entrevistado:	Alain Guerrero Enamorado.
Cargo que ocupa:	Vicedecano de producción.
Preguntas:	**Respuestas:**
1. ¿Cuáles son las estructuras de	**R: /** Entre las estructuras de datos que más

[15] SCADA es el acrónimo correspondiente a *Supervisory Control And Data Acquisition*, cuya traducción literal seria: Adquisición de Datos y Control de Supervisión.

datos más conocidas por los estudiantes y profesores, principalmente de la asignatura de Programación?	conocen nuestros estudiantes y profesores debido al uso que le dan en la solución de diferentes problemas, están: las listas, las pilas, las colas, los grafos, los árboles y en menor uso, las tablas hash para conexiones a bases de datos.
2. ¿En qué se emplean las diferentes estructuras de datos en la facultad?	**R: /** Estas estructuras son usadas en la solución de diferentes problemas, no solo en la facultad sino en toda la universidad, por ejemplo, las mismas pruebas de nivel de programación y en la elaboración de diferentes aplicaciones ya sean en ambiente Web o de Escritorio.
3. ¿Conoce usted alguna biblioteca que contenga estructuras de datos avanzadas o si se ha realizado algún trabajo o investigación sobre el tema?	**R: /** Hasta donde tengo conocimientos, no existe una biblioteca que contenga dichas estructuras. Recomiendo visitar los laboratorios de producción, en ellos podrás entrevistar a los jefes o lideres para obtener más datos sobre el tema, ellos podrán explicar si hay algo hecho o alguna investigación realizada, pues son los que trabajan directamente con las estructuras de datos**.**

Nombre del entrevistado:	Kenia Reyes Guerrero.
Cargo que ocupa:	Colíder del proyecto “Automatización de los servicios para la Biblioteca Nacional y UCI”
Preguntas:	**Respuestas:**
1. ¿En qué se emplean las diferentes estructuras de datos en la facultad?	**R: /** De las estructuras de datos hay mucho de que hablar, pero su utilización esta dada por lo que puedan realizar nuestros propios estudiantes

	y profesores, se que algunos lenguajes de programación como C# traen algunas incluidas.
2. ¿Conoce usted alguna biblioteca que contenga estructuras de datos avanzadas o si se ha hecho algún trabajo o investigación sobre el tema?	**R: /** Como te decía, algunos lenguajes traen algunas estructuras de datos incluidas para facilitar el trabajo con el manejo de los tipos de datos, pero no existe una biblioteca que recoja la mayor variedad de estas para usarlas. Pienso que se debería trabajar en base a esto.

Nombre del entrevistado:	Edisniel Carrazana
Cargo que ocupa:	Jefe general del proyecto de "Automatización de los servicios para la biblioteca Nacional y UCI"
Preguntas:	**Respuestas:**
1. ¿Cómo utilizan en el proyecto las estructuras de datos?	**R: /** Nosotros nos tomamos el trabajo de implementar las estructuras que necesitamos para trabajar. Está buena la idea de ustedes, de realizar una biblioteca que contenga estas estructuras, pero que no se quede solo en su facultad, debería de existir un sitio donde se publicaran esos trabajos, para el conocimiento de todos ya que muchas veces tenemos que gastar más tiempo del requerido en la elaboración de determinada aplicación para buscar o implementar las estructuras de datos que necesitamos.

Nombre del entrevistado:	Abel Meneses
Cargo que ocupa:	Jefe del proyecto "Servicio y soporte para la migración a

	Software Libre. Unicornio”

Preguntas:	Respuestas:
1. ¿Qué bibliotecas o estructuras de datos utilizan para el desarrollo del proyecto? ¿Conocen otras?	**R: /** Básicamente utilizamos las bibliotecas GTK, aunque realizamos una pequeña librería para el trabajo con multiventanas pero solo contenía algunas variantes de listas.
2. ¿Conocen de otros proyectos o personas ustedes en la universidad que hayan hecho algo parecido a lo que hicieron?	**R: /** Realmente no ha habido en la UCI un trabajo con esa complejidad, no tenemos conocimiento de la existencia de alguna otra biblioteca parecida.

Nombre del entrevistado:	Eiger Mora.
Cargo que ocupa:	Analista principal de trabajo con Multiventanas (estudiante).

Preguntas:	Respuestas:
1. ¿Qué opina de la investigación que queremos desarrollar?	**R: /** Creo que es una excelente idea, que tendrá una buena repercusión en la universidad si se llega a consumar, además valdría la pena que se revisaran las estructuras de datos que ya existen pues algunas tienen errores. Tienen un gran trabajo que realizar y ojala tengan éxito.

ANEXO 2. Diagramas de Interacción (Colaboración y Secuencia) del Diseño.

CU Gestionar elementos en una Lista – Escenario: Obtener Elemento.

Diagrama de Colaboración

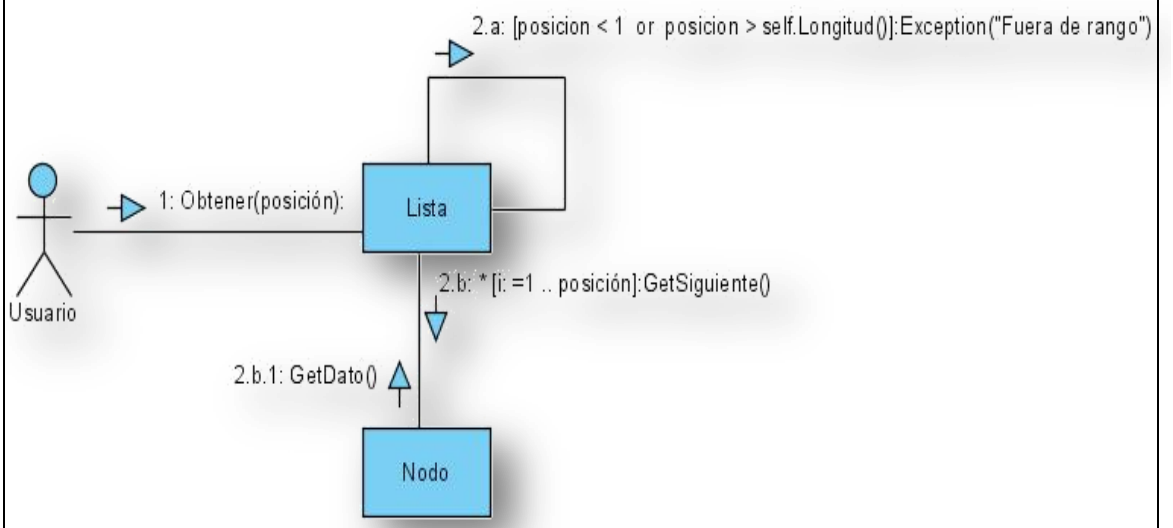
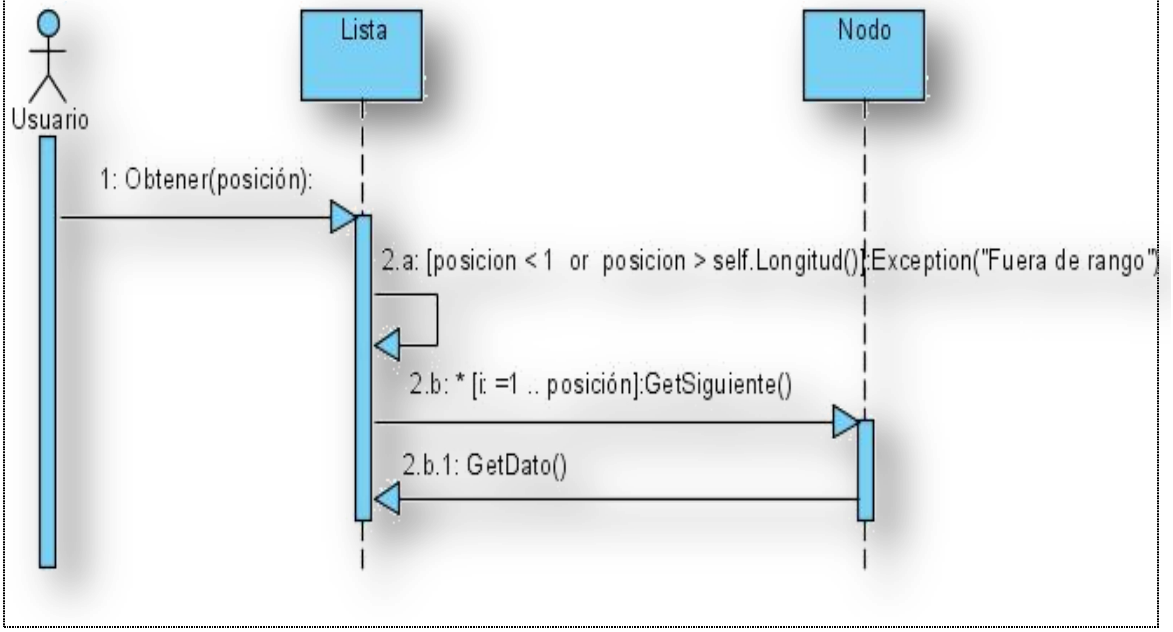


Diagrama de Secuencia



CU Gestionar elementos en una Lista – Escenario: Insertar Elemento.

Diagrama de Colaboración

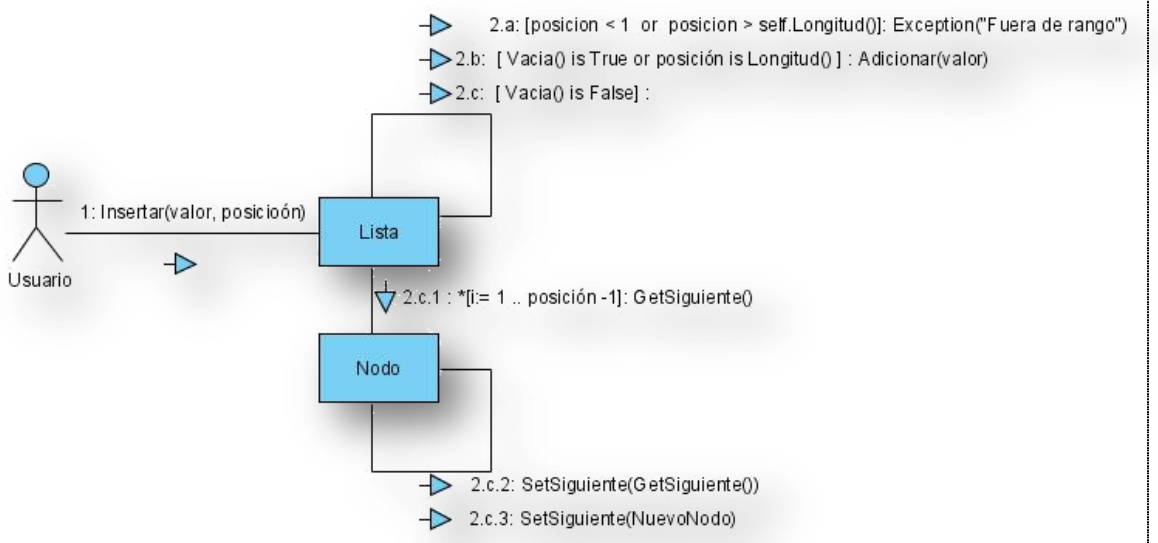
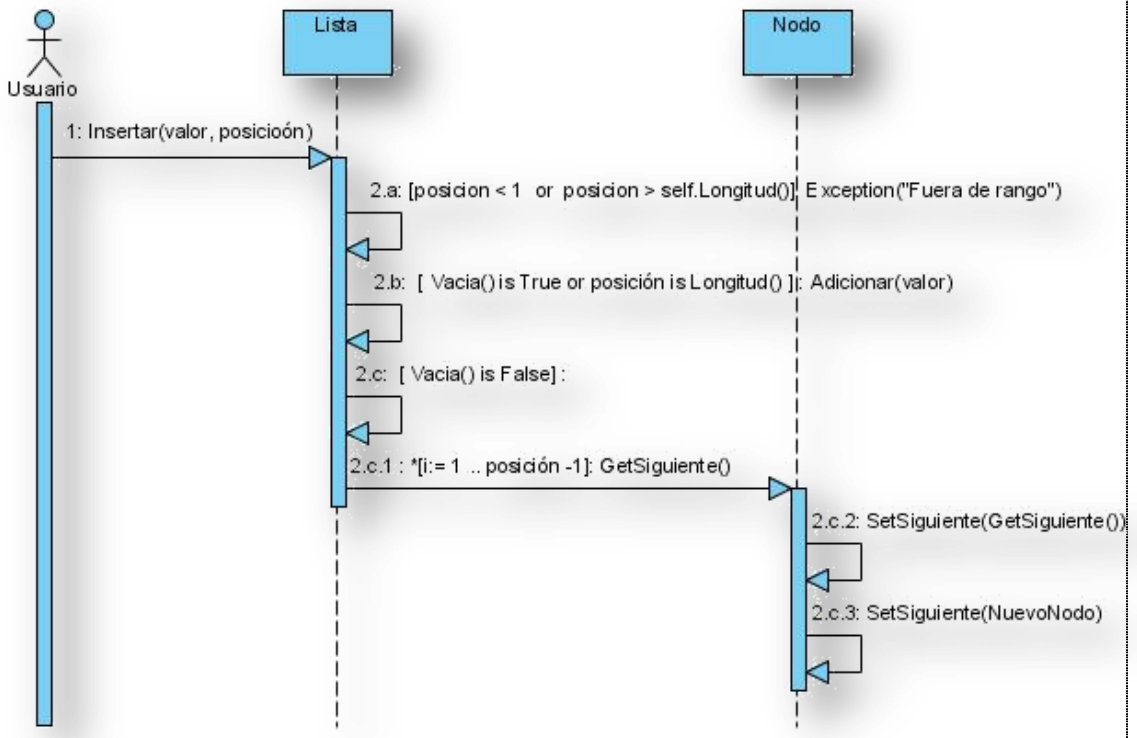


Diagrama de Secuencia



CU Gestionar elementos en una Lista – Escenario: Eliminar Elemento.

Diagrama de Colaboración

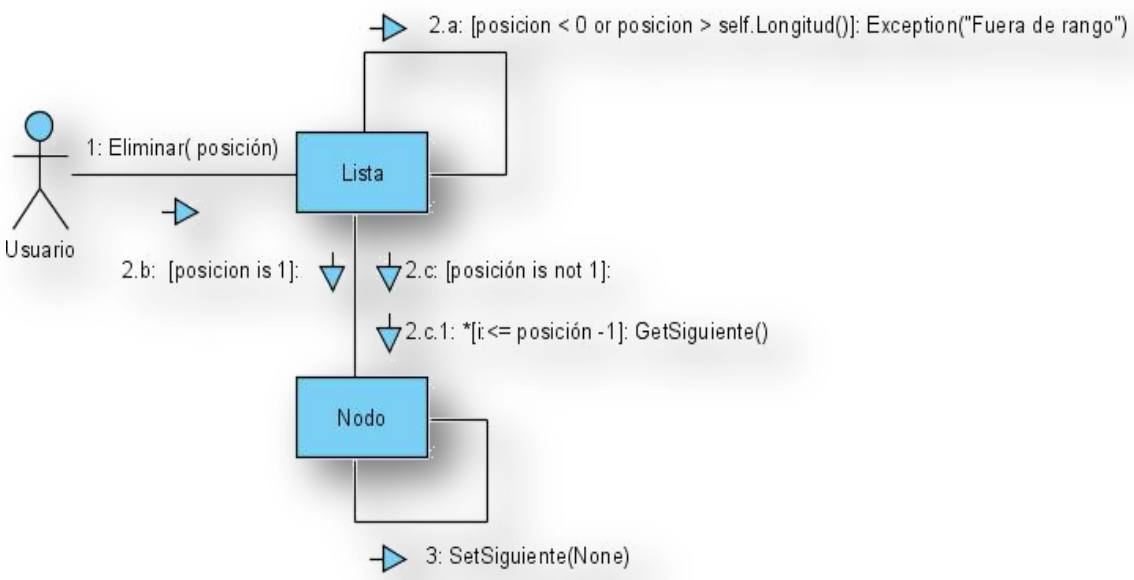
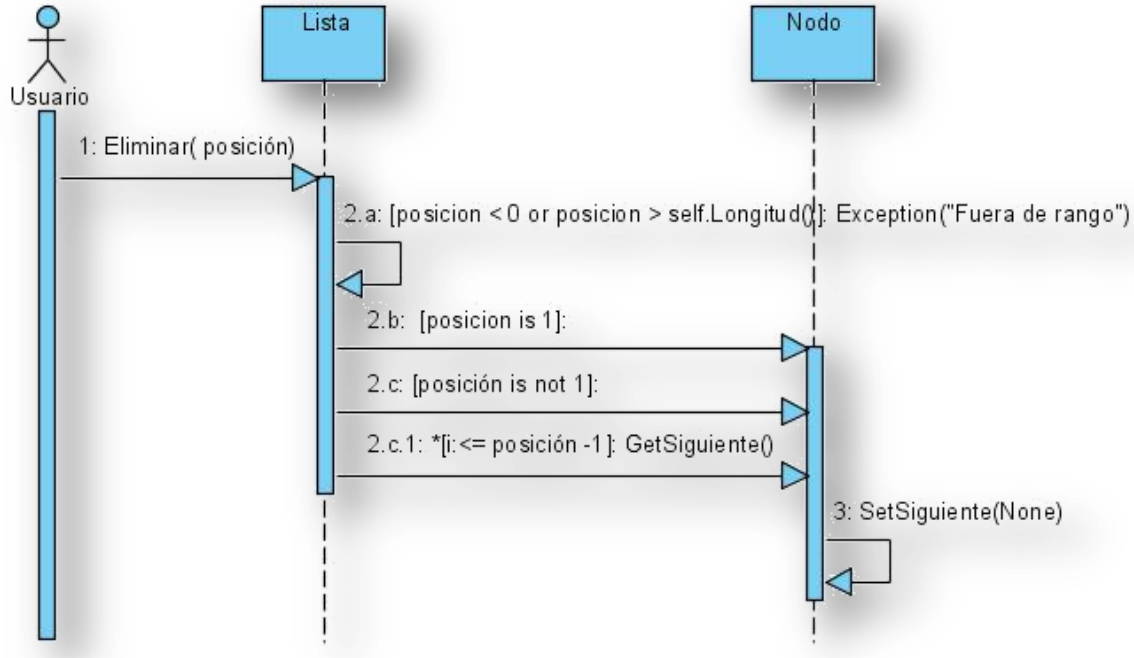


Diagrama de Secuencia



CU Gestionar elementos en una Pila – Escenario: Tope.

Diagrama de Colaboración

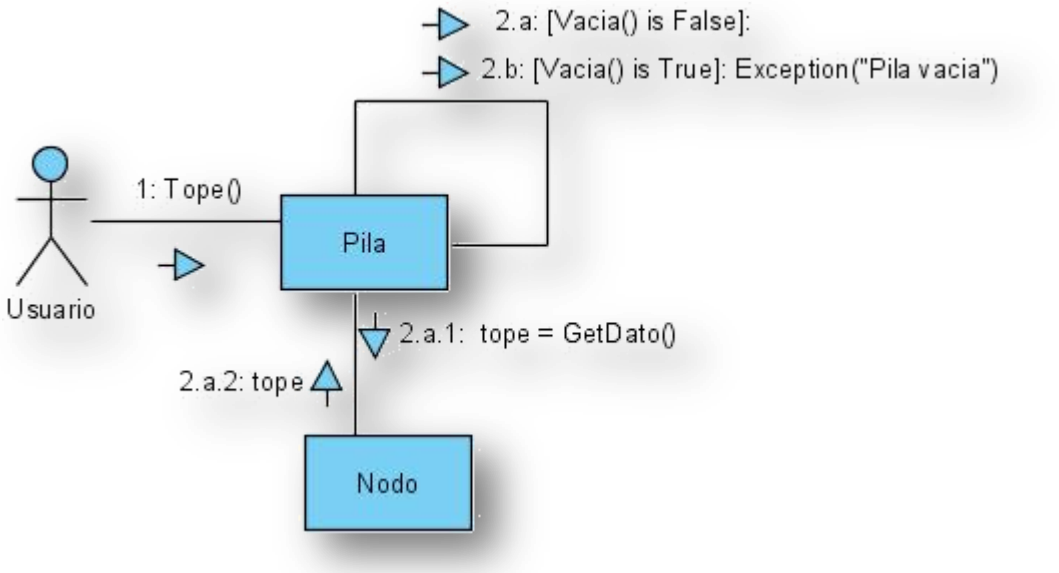
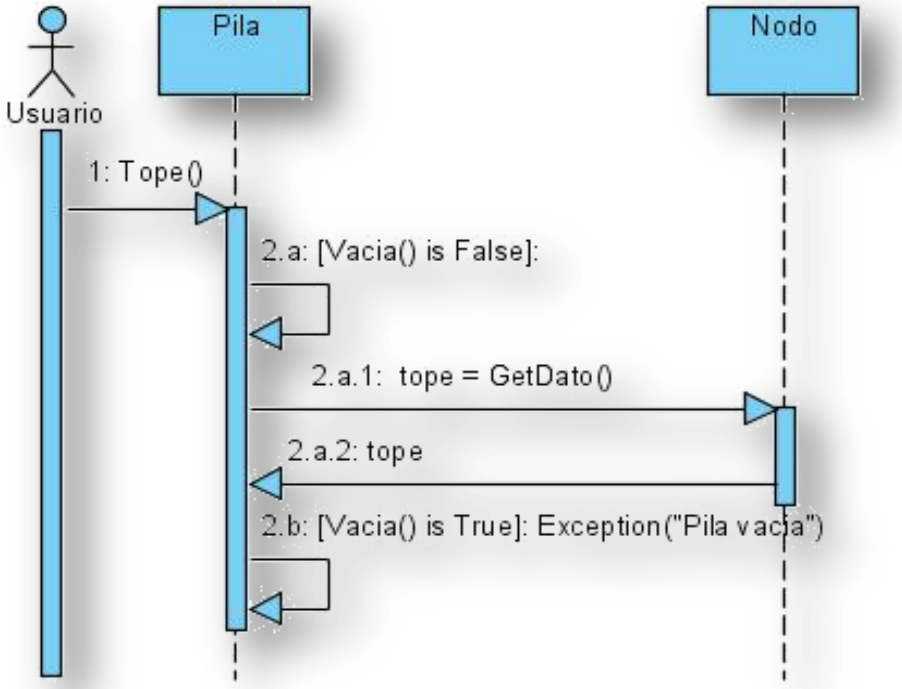


Diagrama de Secuencia



CU Gestionar elementos en una Pila – Escenario: Apilar elemento.

Diagrama de Colaboración

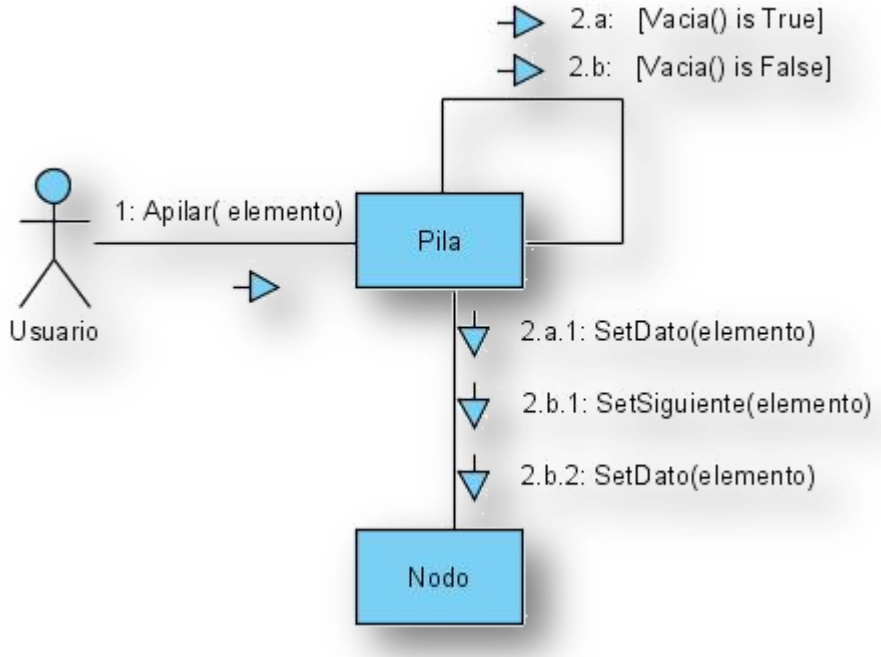
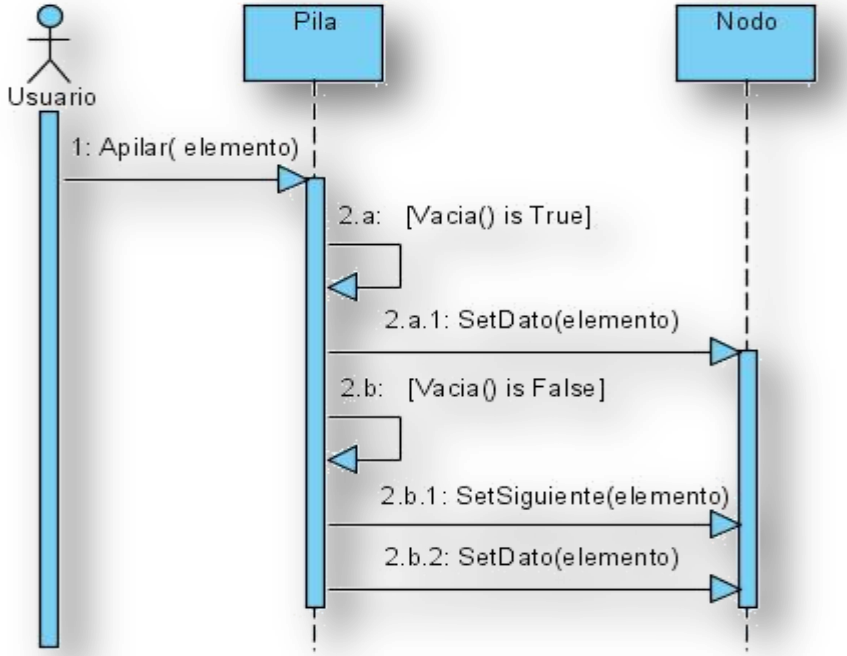


Diagrama de Secuencia



CU Gestionar elementos en una Pila – Escenario: Desapilar elemento.

Diagrama de Colaboración

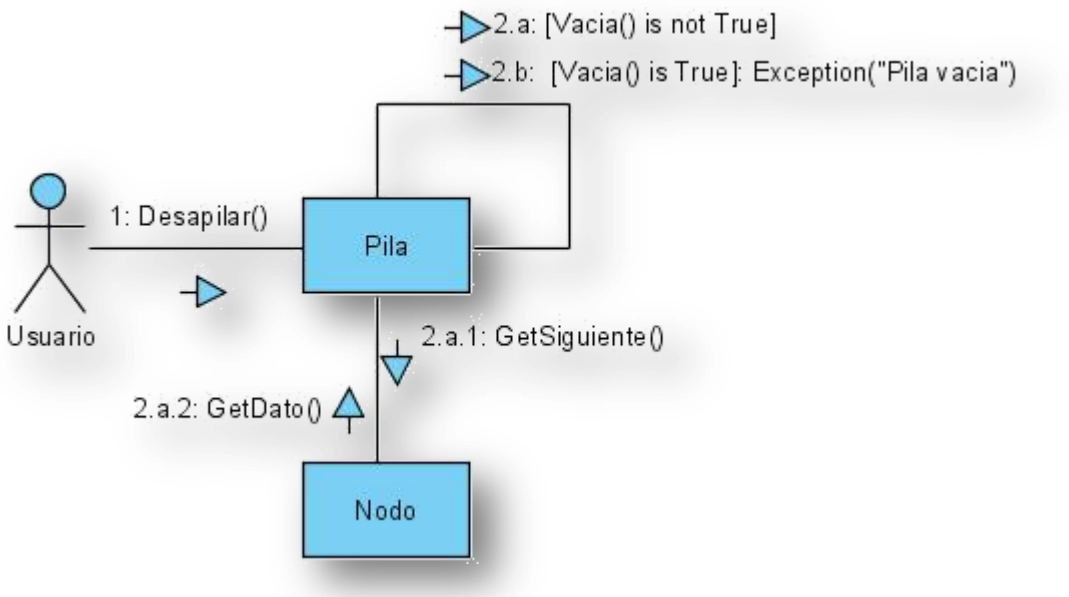
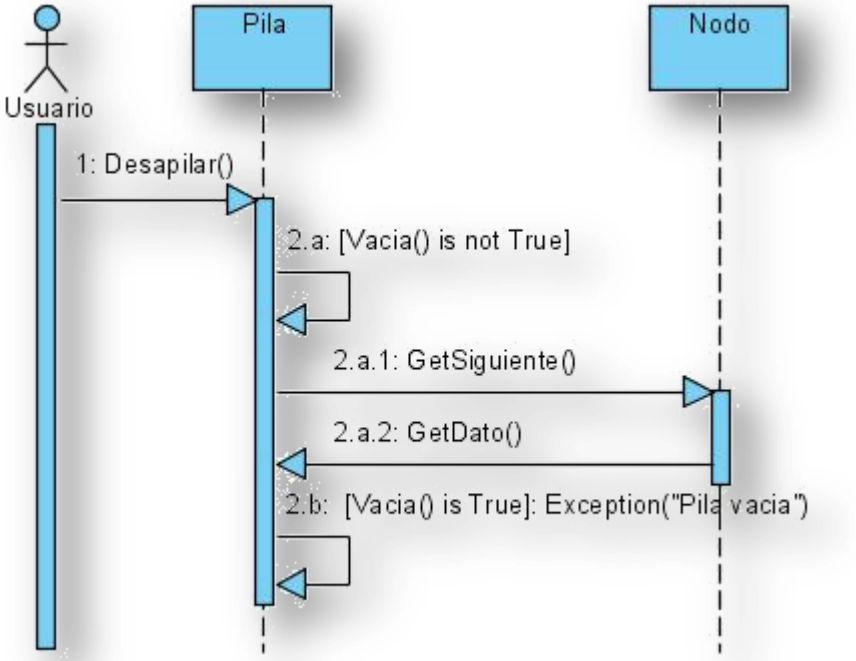


Diagrama de Secuencia



CU Gestionar elementos en una Cola – Escenario: Frente.

Diagrama de Colaboración

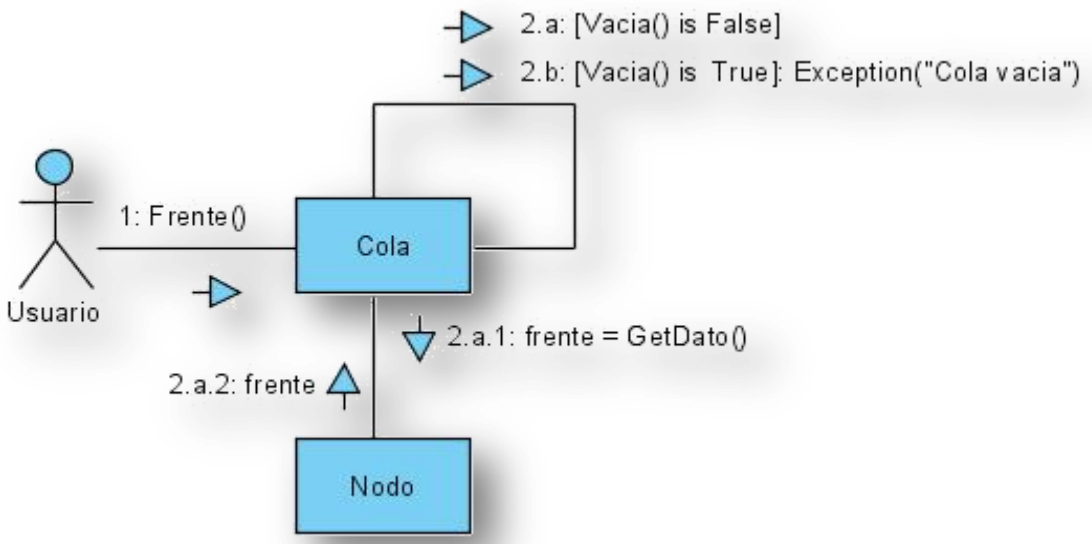
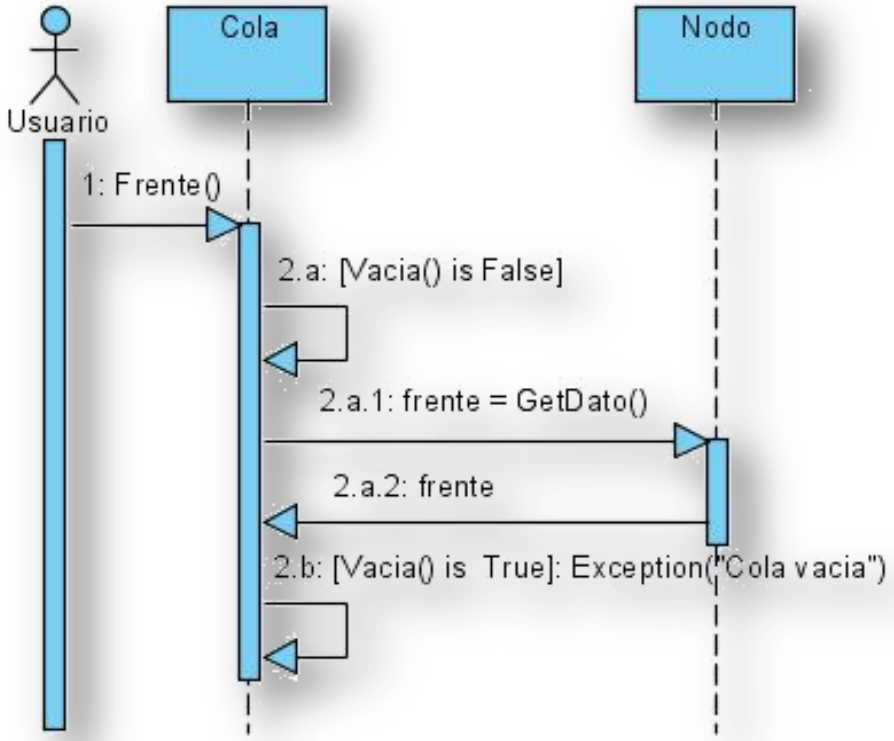


Diagrama de Secuencia



CU Gestionar elementos en una Cola – Escenario: Fondo.

Diagrama de Colaboración

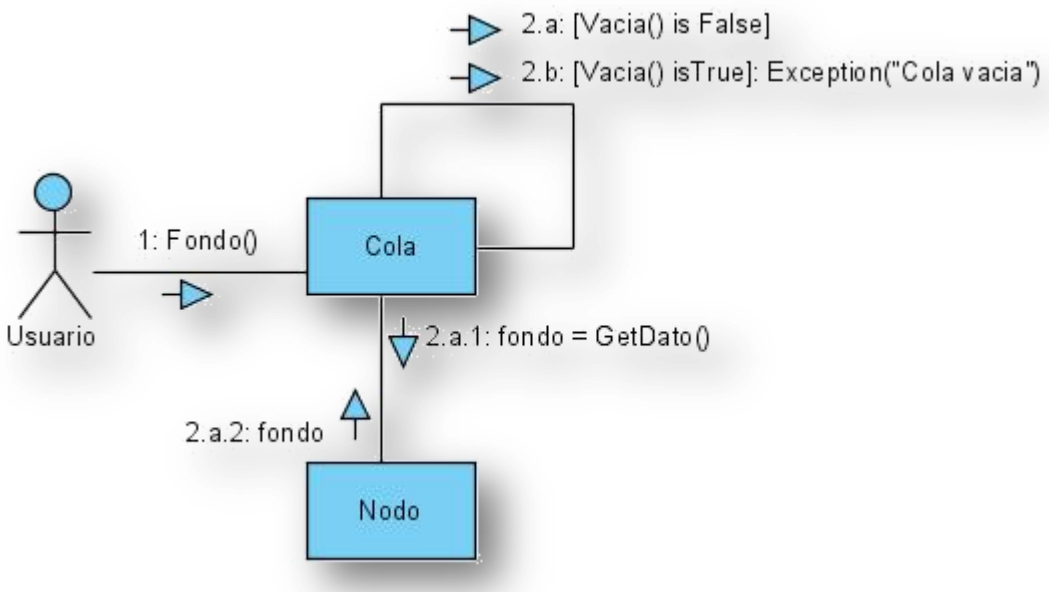
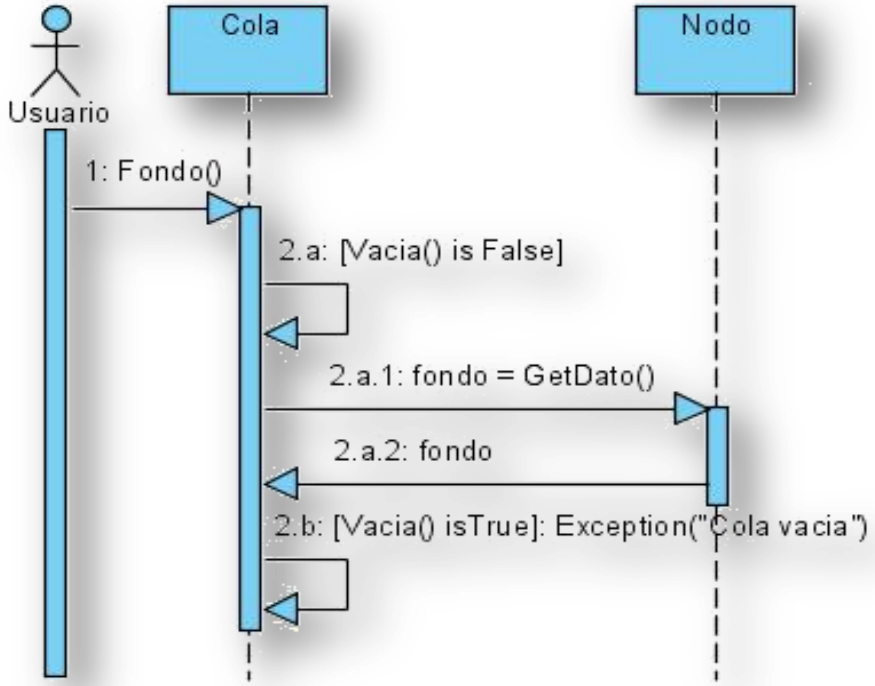


Diagrama de Secuencia



CU Gestionar elementos en una Cola – Escenario: Adicionar elemento.

Diagrama de Colaboración

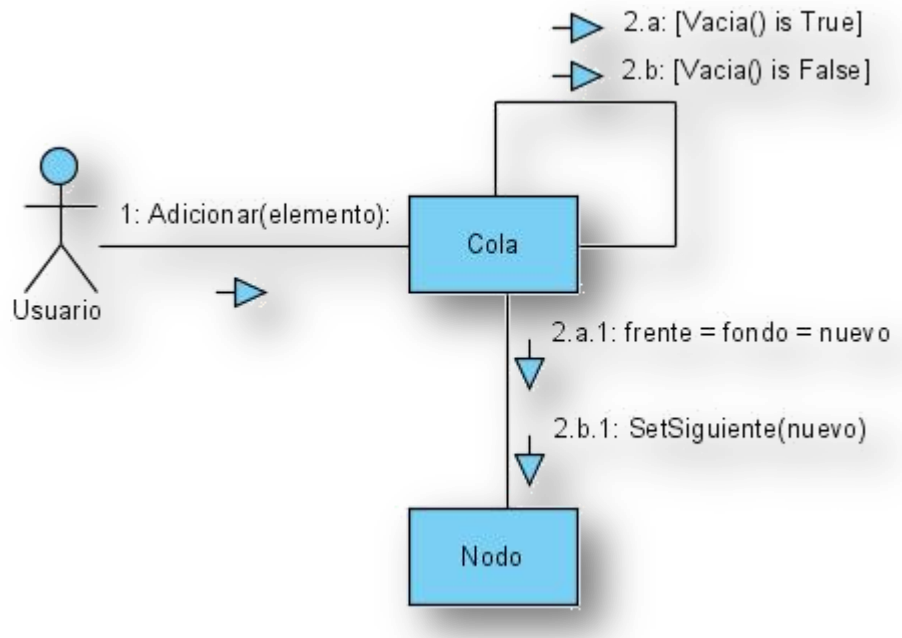
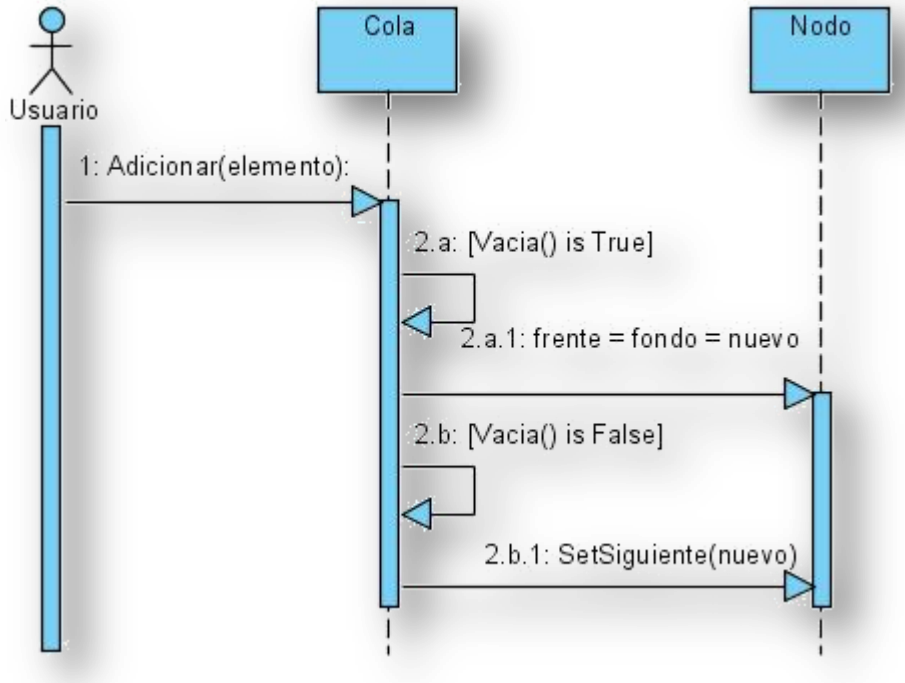


Diagrama de Secuencia



CU Gestionar elementos en una Cola – Escenario: Extraer elemento.

Diagrama de Colaboración

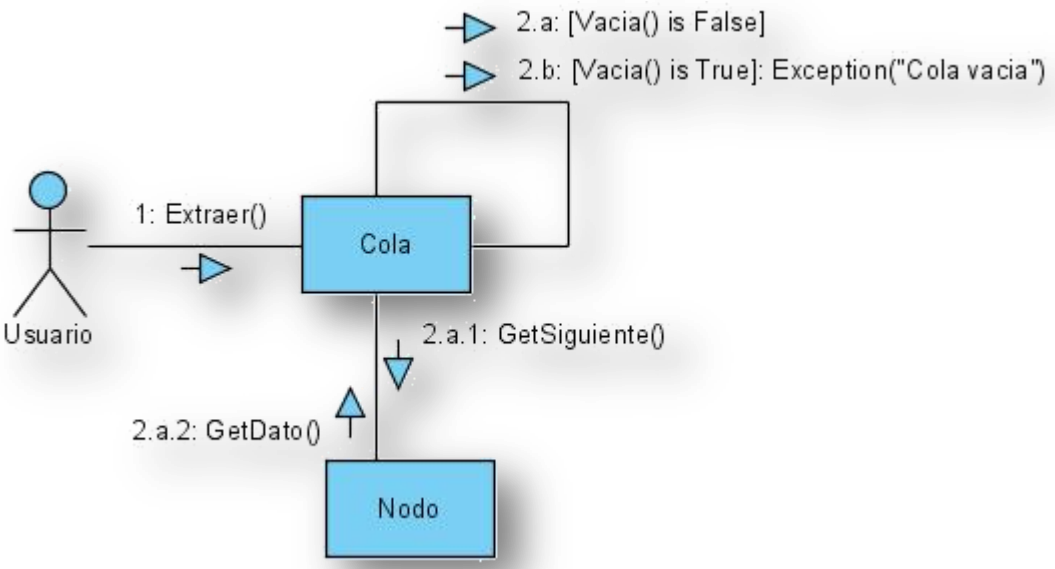
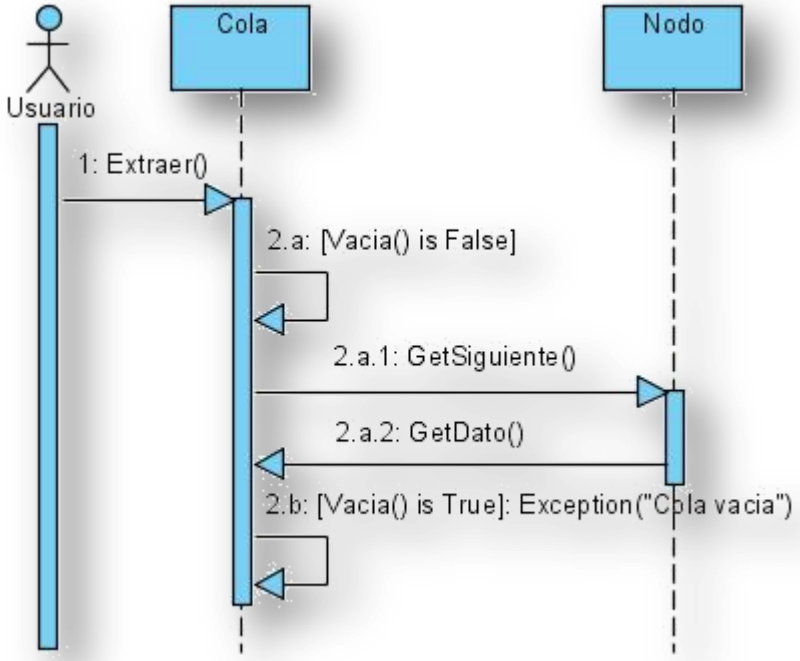


Diagrama de Secuencia



CU Gestionar elementos en una Tabla Hash – Escenario: Insertar elemento.

Diagrama de Colaboración

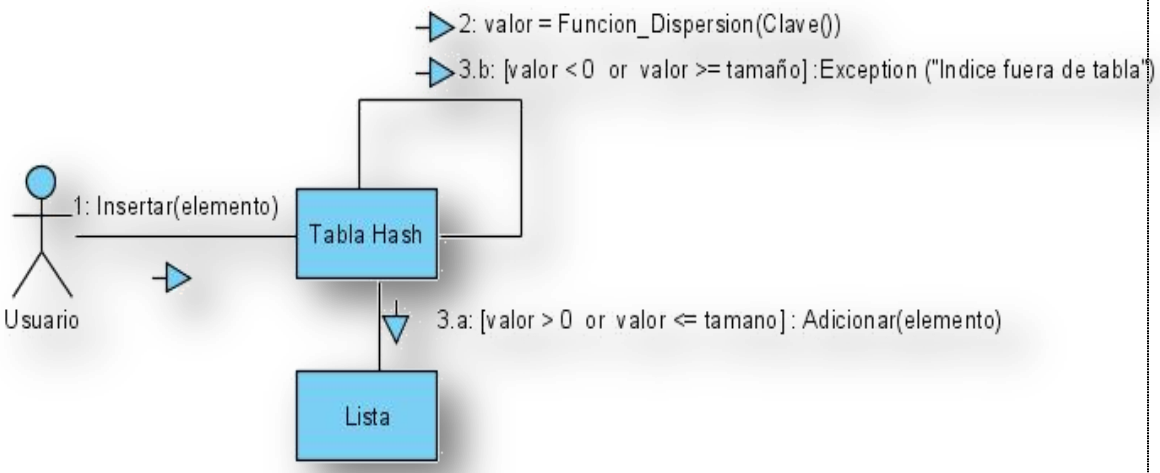
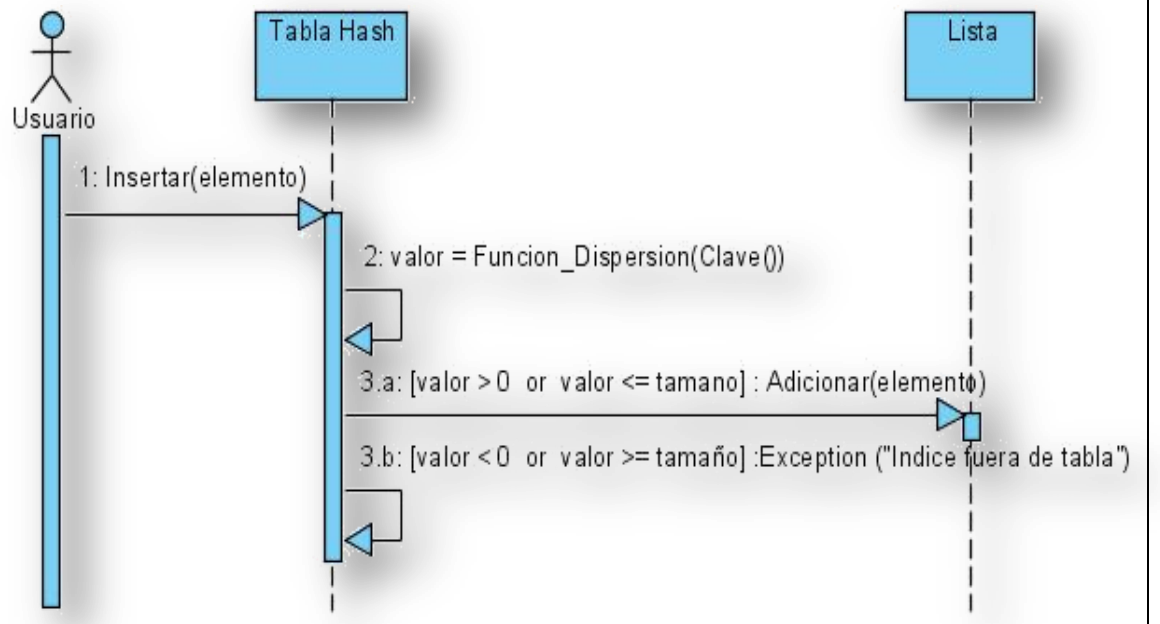


Diagrama de Secuencia



CU Gestionar elementos en una Tabla Hash – Escenario: Buscar elemento.

Diagrama de Colaboración

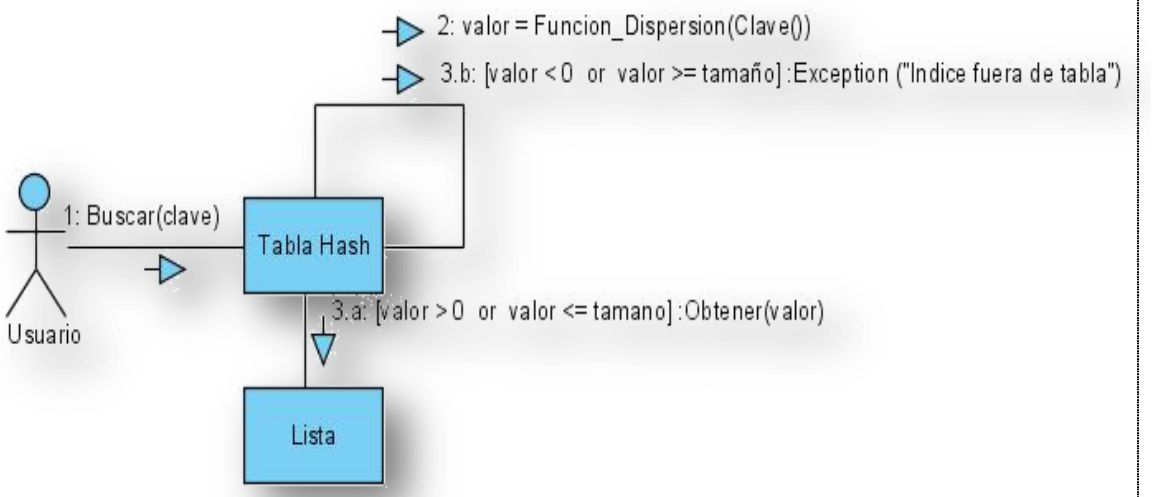
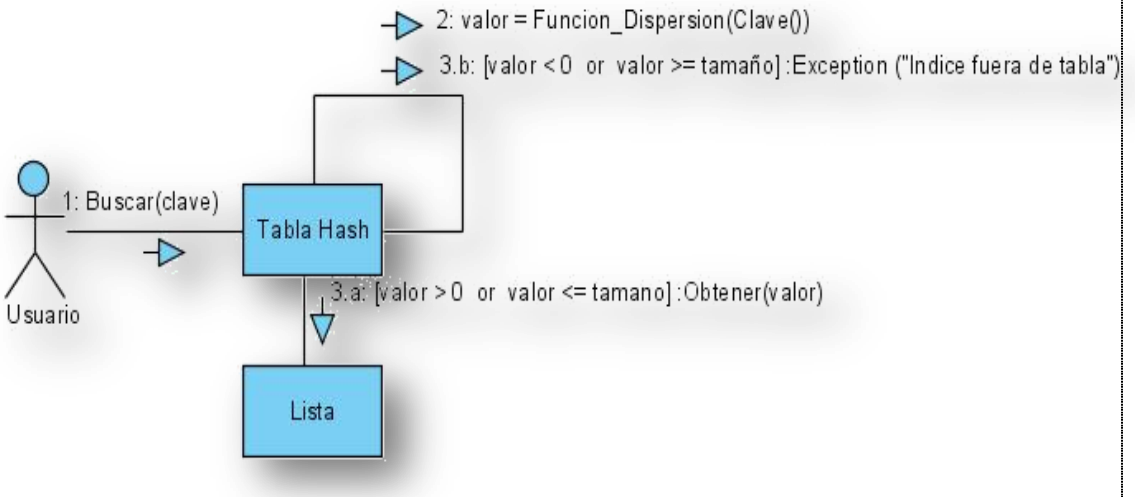


Diagrama de Secuencia



CU Gestionar elementos en una Tabla Hash – Escenario: Eliminar elemento.

Diagrama de Colaboración

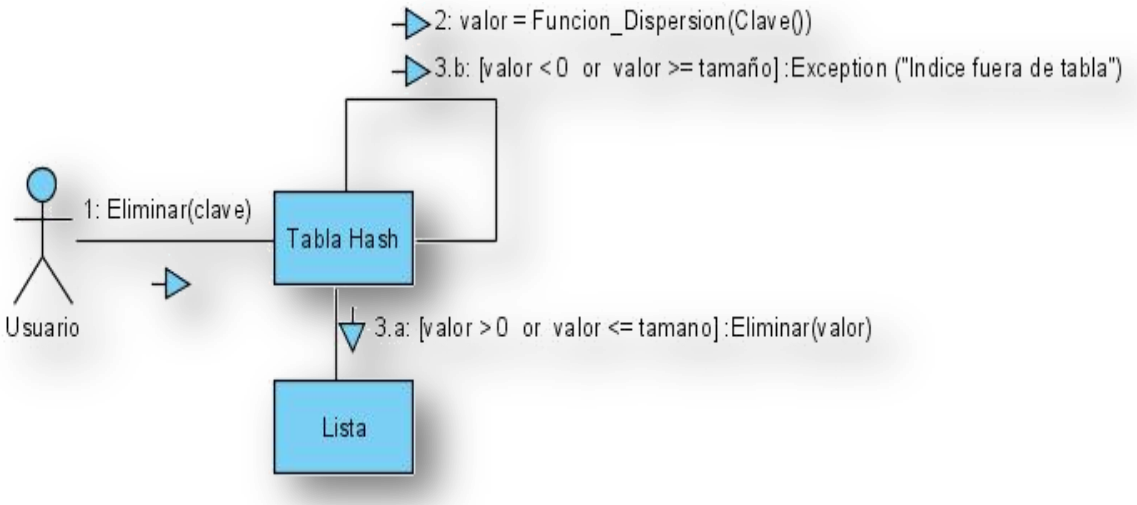
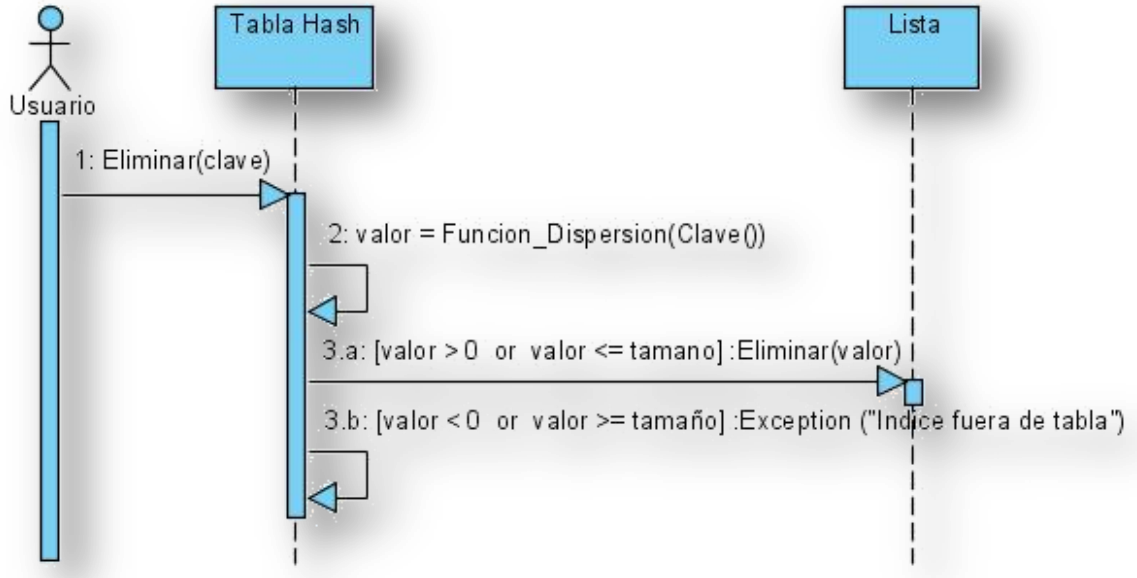
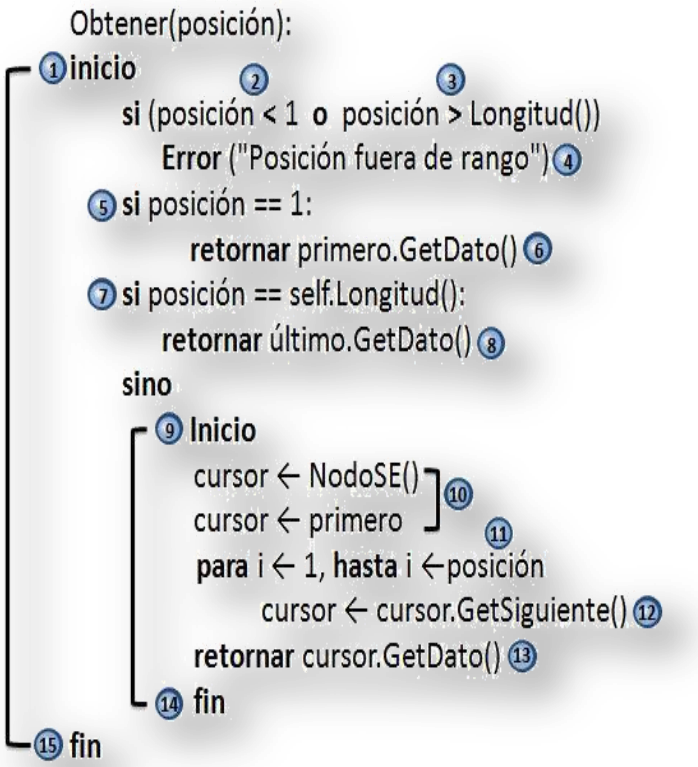
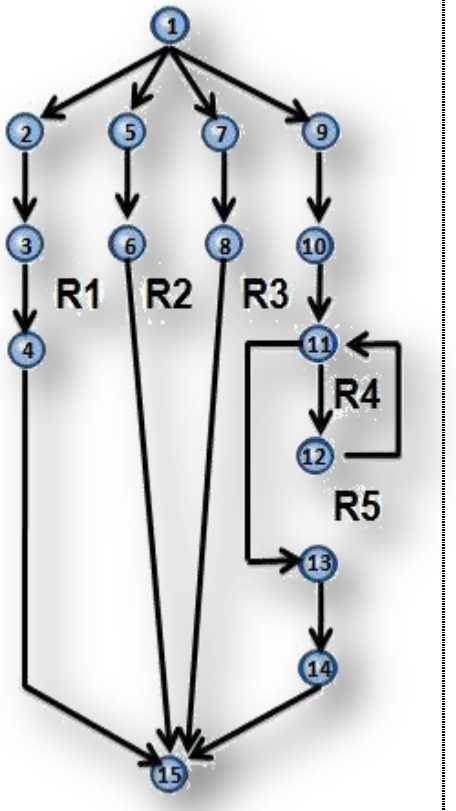


Diagrama de Secuencia



ANEXO 3 Casos de Prueba.

Figura 4.10 Prueba del camino básico para el CU: Gestionar los elementos en una Lista, escenario: Obtener elemento.

Seudocódigo	Grafo de Flujo
	

El grafo tiene cuatro regiones por lo que el camino básico estará formado por cinco caminos en el programa. Estos caminos pueden ser:

Camino 1: 1, 2, 3, 4, 15.

Camino 2: 1, 5, 6, 15.

Camino 3: 1, 7, 8, 15.

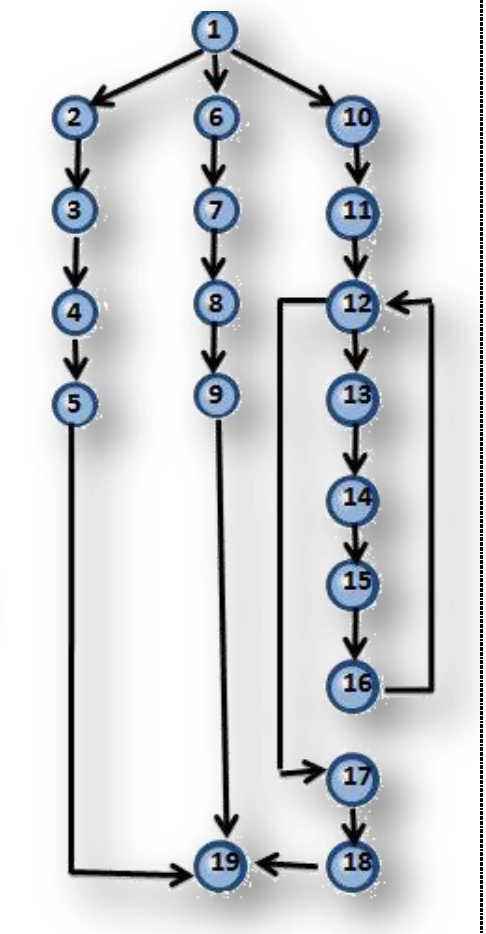
Camino 4: 1, 9, 10, 11, 12, 11, 13, 14, 15.

Camino 5: 1, 9, 10, 11, 13, 14, 15.(se realiza lo mismo que el camino 1)

Posibles casos de prueba que se pueden generar para probar estos caminos.

Número del Camino	Caso de Prueba	Objetivo	Resultado Esperado
1	Posición < 1 Posición > Longitud	Probar obtener un elemento en una posición no válida.	Excepción (“Posición fuera de rango.”)
2	Posición = 1	Probar obtener el elemento que se encuentra en la primera posición.	Elemento obtenido correctamente.
3	Posición = Longitud	Probar obtener el elemento que se encuentra en la última posición.	Elemento obtenido correctamente.
4	Posición = 10	Probar obtener un elemento en cualquier posición de la lista.	Elemento obtenido correctamente.

Figura 4.11 Prueba del camino básico para el CU: Gestionar los elementos en una Lista, escenario: Insertar elemento.

Seudocódigo	Grafo de Flujo
Insertar (Tipo x, **entero** pos) 1 **inicio** 2 3 **si** ((pos < 0) **o** (pos ≥ longitud)) **entonces** Error("Posición No Válida") 4 nodo ← **nuevo** NodoSE(x, NULL) 5 6 **si** (pos = 0) **entonces** 7 **inicio** nodo. siguiente ← cabeza cabeza ← nodo 8 9 **fin** **sino entonces** 10 **inicio** 11 cursor ← cabeza pos_cursor ← 0 12 13 **mientras** (cursor ≠ NULL **y** pos_cursor+1 ≠ pos) 14 **inicio** pos_cursor ← pos_cursor + 1 cursor ← cursor. siguiente 15 16 **fin** nodo. siguiente ← cursor. siguiente cursor. siguiente ← nodo 17 18 **fin** 19 **fin**	

El grafo tiene cuatro regiones por lo que el camino básico estará formado por cuatro caminos en el programa. Estos caminos pueden ser:

Camino 1: 1, 2, 3, 4, 5, 19.

Camino 2: 1, 6, 7, 8, 9, 19.

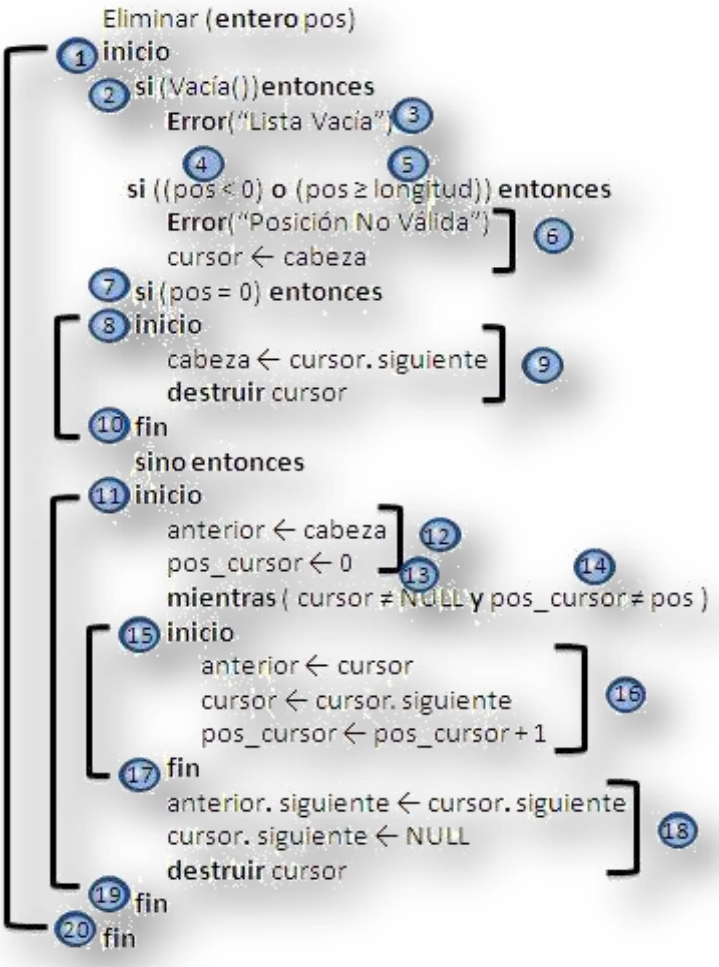
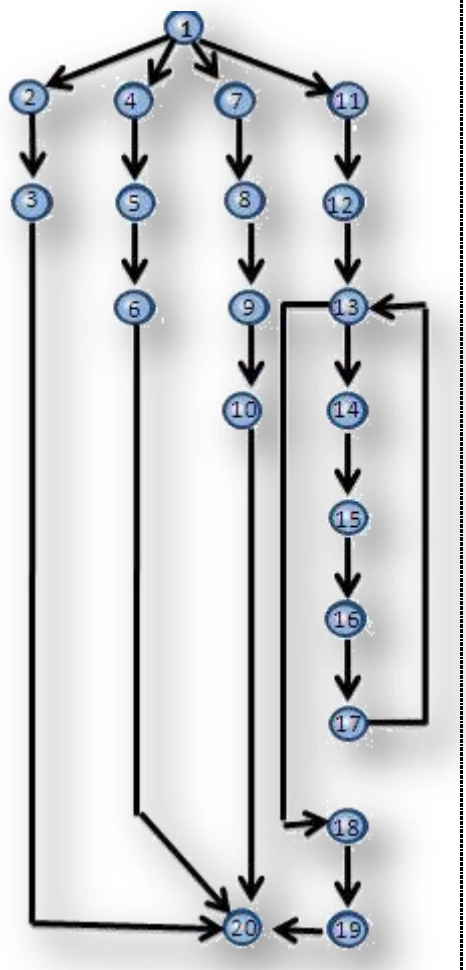
Camino 3: 1, 10, 11, 12, 17, 18, 19.

Camino 4: 1, 10, 11, 12, 13, 14, 15, 16, 12, 17, 18, 19.

Posibles casos de prueba que se pueden generar para probar estos caminos.

Número del Camino	Caso de Prueba	Objetivo	Resultado Esperado
1	Posición < 0 Posición >= Longitud	Probar insertar un elemento en una posición no válida.	Excepción (“Posición no válida”)
2	Posición = 0 elemento = 6	Probar insertar un elemento en la posición cero.	Elemento insertado correctamente.
3	Cursor = NULL Posición_cursor + 1 = posición	Probar insertar un elemento al final de la lista.	Elemento insertado correctamente.
4	Posición = 5 elemento = 20	Probar insertar un elemento en cualquier posición de la lista.	Elemento insertado correctamente.

Figura 4.12 Prueba del camino básico para el CU: Gestionar los elementos en una Lista, escenario: Eliminar elemento.

Seudocódigo	Grafo de Flujo
	

El grafo tiene cinco regiones por lo que el camino básico estará formado por cinco caminos en el programa. Estos caminos pueden ser:

Camino 1: 1, 2, 3, 20.

Camino 2: 1, 4, 5, 6, 20.

Camino 3: 1, 7, 8, 9, 10, 20.

Camino 4: 1, 11, 12, 13, 18, 19, 20.

Camino 5: 1, 11, 12, 13, 14, 15, 16, 17, 13, 18, 19, 20.

Posibles casos de prueba que se pueden generar para probar estos caminos.

Número del Camino	Caso de Prueba	Objetivo	Resultado Esperado
1	Lista vacía	Probar eliminar un elemento con la lista vacía.	Excepción (“Lista vacía”)
2	Posición < 0 Posición >= Longitud	Probar eliminar un elemento en una posición no válida.	Excepción (“Posición no válida”)
3	Posición = 0	Probar eliminar un elemento en la primera posición	Elemento eliminado correctamente.
4	Posición = 5 elemento = 20	Probar eliminar un elemento en cualquier posición de la lista.	Elemento eliminado correctamente.
5		Probar eliminar el último elemento que queda en la lista.	Elemento eliminado correctamente.

Universidad de las Ciencias Informáticas.

Facultad 9.