Universidad de las Ciencias Informáticas

Facultad 3

Desarrollo del Evaluador de Memoria para el Laboratorio Virtual de Sistemas Operativos.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE

INGENIERÍA EN CIENCIAS INFORMÁTICAS

Autor:

Yunet Estrada Grandales

Tutor:

Ing. Carlos Y. Hidalgo García

Ciudad de La Habana, Cuba

2012

Las ideas no necesitan ni de las armas, en la medida en que sean capaces de conquistar a las grandes masas.

"Fidel Castro Ruz"

DECLARACIÓN DE AUTORÍA

Declaro ser la autora de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Carlos Y. Hidalgo García **Yunet Estrada Grandales**

_____ _____

Firma del tutor Firma del autor

DEDICATORIA

Dedico este trabajo:

A mis padres por ser mis guías y mi razón de querer crecer profesionalmente, espero que se sientan orgullosos de mí.

A mi sobrino bello y a mi hermano para cuando crezcan sigan mis pasos.

A mi abuelo Maso que a pesar de no estar con nosotros lo llevo en mi corazón porque siempre luchó para que ni a mi hermana ni a mí nos faltara nada.

A mi abuelita Dora por operarse el ojo para verme cuando llegara de Venezuela, gracias por malcriarme y preocuparte siempre por mí.

AGRADECIMIENTOS

Mis agradecimientos son para toda mi familia y amigos que de una forma u otra me ayudaron, no solo aquí en la universidad sino fuera de esta:

Primero que todo agradecerle a mis Padres por apañarme en mis malcriadeces, por consentirme, siempre seré su niña, gracias por todo lo que han hecho por mí, sé que se han tenido que sacrificar bastante y espero poder compensárselos.

A mi hermana que a pesar de no estar aquí personalmente sé que en estos momentos debe estar súper nerviosa por mí, gracias por ayudarme a dar mis primeros pasos en la vida.

A mi novio Marlon, sin ti no lo hubiera logrado, gracias por acompañarme y apoyarme en estos últimos años de la carrera y espero que no sean los últimos que estés a mi lado.

A mis tíos Carmen Y Hodelín gracias porque han sido como una familia para mí en estos 5 años lejos de mi casa.

A toda mi familia (abuelos, tíos y primos) que de una forma u otra influyeron en el desarrollo de mi carrera universitaria.

A mi familia venezolana que sé que darían cualquier cosa por estar aquí en este momento.

A mis amistades de la universidad en especial a mis hermanitas (Yusmila, Dailín, Semidaris, Nana) y a los amigos de mi novio que son míos también (Evelio, Juan Manuel, Yarilaisi) por pasar juntos momentos inolvidables.

RESUMEN

El presente trabajo de diploma está enmarcado en desarrollar un Evaluador que posibilite al estudiante la evaluación y comprobación de los contenidos adquiridos en clases, específicamente en el tema de Administración de Memoria de la asignatura Sistemas Operativos, puesto que la misma es un poco compleja debido al alto nivel de abstracción que se necesita para impartir el contenido de algunos temas.

Con este fin se realizó un estudio sobre: conceptos asociados a laboratorios virtuales determinando las ventajas y desventajas que poseen, se investigó sobre los sistemas evaluadores existentes en el mundo, aspecto fundamental para el desarrollo del trabajo, metodologías de desarrollo de software, herramientas de modelado, lenguajes de programación, entre otros aspectos asociados a la Ingeniería de Software. Finalmente se aplicaron métricas y pruebas dirigidas a evaluar la calidad de los requisitos y el sistema, obteniéndose resultados satisfactorios.

PALABRAS CLAVES

“Evaluador, Administración de Memoria, Sistemas Operativos, Laboratorios Virtuales”

INDICE

INTRODUCCIÓN 11

CAPÍTULO 1: Fundamentación Teórica 15

1.1 Introducción 15

1.2 Laboratorio Virtual 15

1.2.1 Conceptos 15

1.2.2 Ventajas e Inconvenientes 16

1.2.3 Tipos de Laboratorios Virtuales 17

1.3 Definiciones 18

1.3.1 Soluciones existentes: 19

1.4 Proceso de Desarrollo del Software 20

1.4.1 Ingeniería de Requisitos 21

1.4.2 Patrones 25

1.4.3 Métrica para evaluar el Diseño de Software 31

1.4.4 Pruebas de Software 31

1.5 Metodologías de desarrollo 33

1.5.1 Proceso Unificado de Desarrollo de Software (RUP). 33

1.6 Lenguajes de Modelado 36

1.6.1 UML (2.0) 36

1.7 Herramientas CASE 37

1.7.1 Visual Paradigm (VP) 5.0. 38

1.8 Lenguajes de Programación 38

1.8.1 Java 39
1.9 Herramientas de desarrollo 39
1.9.1 NetBeans. 7.1 39
1.10 Conclusiones Parciales 40
Capítulo 2: Descripción y validación de la solución propuesta 41
2.1 Introducción 41
2.2 Modelo de Dominio 41
2.2.1 ¿Por qué modelo de dominio? 41
2.2.2 Conceptos del Modelo del dominio del sistema: 42
2.2.3 Diagrama del Modelo del Dominio: 43
2.3 Especificación de los requerimientos del software 43
2.3.1 Requisitos funcionales: 44
2.3.2 Requisitos no funcionales 45
2.4 Modelo de Casos de Uso del Sistema 46
2.4.1 Actores del Sistema 46
2.4.2 Diagrama de Casos de Uso del Sistema 47
2.4.3 Patrones de Casos de Uso del Sistema: 48
2.4.4 Descripción de Casos de Uso del Sistema 48
2.5 Análisis del Sistema 50
2.6 Diseño del sistema 51
2.6.1 Diagrama de clases del diseño 51
2.6.2 Diagrama de Interacción 52
2.7 Arquitectura del Software: 53

2.7.1 Vista lógica de la arquitectura: 54
2.8 Patrones de diseño 55
2.9 Conclusiones Parciales 56
Capítulo 3: Implementación y Validación de los artefactos generados 57
3.1 Introducción 57
3.2 Implementación 57
3.2.1 Estándares de codificación de JAVA 57
3.2.2 Diagrama de Componentes 59
3.3 Resultados Obtenidos 60
3.4 Validación de los artefactos obtenidos 60
3.4.1 Métricas para la Calidad de la Especificación de los Requisitos de Software 61
3.4.2 Métricas para la evaluación de la calidad en los diagramas de casos de usos 63
3.4.3 Tamaño de clase (TC) 66
3.4.4 Pruebas de Software 68
3.5 Conclusiones parciales 70
CONCLUSIONES 71
BIBLIOGRAFÍA REFERENCIADA 73
BIBLIOGRAFÍA 74
ANEXOS 76
Anexo 1 Descripción de los Casos de Usos del sistema: 76
Anexo 2 Diagrama de clases del diseño 86
Anexo 3 Diagramas de Interacción 88

ÍNDICE DE FIGURAS

Figura 1 Modelo de Dominio *43*
Figura 2 Modelo de Casos de usos del sistema *47*
Figura 3 Diagrama de Clases del Diseño *52*
Figura 4 Diagrama de secuencia del Caso de Uso Evaluar Algoritmos de Reemplazos *53*
Figura 5 Vista lógica de la Arquitectura *54*
Figura 6 Diagrama de Componentes *59*
Figura 7 Interfaz Principal del Evaluador para el LAVSO *60*
Figura 9 Juego de datos para escenario #2 del Caso de Prueba" Evaluar Algoritmos de Reemplazos" *69*

ÍNDICE DE GRÁFICAS

Gráfica 1 Calidad Especificación de Requisitos *62*
Gráfica 2 Resultados de la evaluación de la calidad en los diagramas de casos de usos del sistema *66*
Gráfico 3 Número de clases por categorías *67*

ÍNDICE DE TABLAS

Tabla 1 Conceptos del modelo de dominio *42*
Tabla 2 Requisitos Funcionales *45*
Tabla 3 Actores del sistema *47*
Tabla 4 Descripción de Casos de usos del sistema *50*
Tabla 5 Nombre del equipo de revisores del software *61*
Tabla 6 Métricas para la calidad de la funcionalidad del diagrama de casos de uso de sistema *64*
Tabla 7 Resultados de las métricas para la calidad de la funcionalidad del diagrama de casos de uso de sistema *65*
Tabla 8 Métrica tamaño de clases aplicada a las clases del diseño *67*
Tabla 9 Valores de los umbrales *67*
Tabla 10 Diseño del caso de prueba Evaluar Algoritmos de Reemplazos *69*

INTRODUCCIÓN

La Educación Superior en Cuba no ha estado al margen del uso de tecnologías en aras de mejorar la calidad del proceso de Enseñanza y Aprendizaje. Con el desarrollo vertiginoso en las últimas décadas de la informática y con ella la aparición y utilización de las Tecnologías de la Información y la Comunicación (TIC) en todas las esferas de la sociedad actual, ha permitido con su aplicación en las Universidades alcanzar un eslabón superior para este tipo de enseñanza.

Siendo parte de esta ideología la Universidad de las Ciencias Informáticas (UCI) creada bajo el concepto de universidad productiva, se enfrenta a la difícil tarea de formar profesionales capacitados para ser parte activa y fundamental del reto que constituye el desarrollo de la industria cubana del software y la informatización de nuestra sociedad. Actualmente se está aplicando un nuevo modelo de formación centrado en el aprendizaje, donde el estudiante se compromete más en adquirir sus conocimientos y se persigue que obtengan herramientas para aprender, en el cual tiene el Entorno Virtual de Aprendizaje (EVA) un uso protagónico.

El EVA, cuyo concepto viene asociado a un nuevo paradigma en el que convergen las tendencias actuales de la educación, permite la gestión de manera no presencial del proceso de enseñanza-aprendizaje en el cual están inmersos tanto educandos como educadores. Su uso en la UCI es creciente ya que ha permitido utilizar otros recursos de apoyo, materiales didácticos y entornos de simulación entre los que se encuentran los Laboratorios Virtuales, este último le proporciona al educando el uso de prácticas y ejercicios como métodos de consolidación de sus conocimientos.

La asignatura Sistemas Operativos (SO) es fundamental en el plan de estudio del tercer año de la carrera de Ingeniería en Ciencias Informáticas, la cual está dividida en tres temas fundamentales: Gestión de Procesos, Gestión de Memoria, y Gestión de Entrada y Salida de Información.

El tema de Gestión de Memoria es un poco complejo tanto para impartirlo y evaluarlo el profesor como para recibirlo los estudiantes, puesto que cuenta con un componente teórico muy fuerte, resultando difícil para los estudiantes la realización de ejercicios prácticos. Se ha podido observar que esta asignatura en la actualidad carece de una fuerte integración con el desarrollo de las TIC ya que la misma no cuenta

con herramientas didácticas que le permitan al estudiante la consolidación de sus conocimientos y habilidades.

La carencia de materiales didácticos en la asignatura conlleva a que los estudiantes no cuenten con los medios necesarios para aumentar la comprensión de los contenidos vistos en el aula, así como cuenten con una forma de poner en práctica los contenidos teóricos que se imparten por parte de los profesores.(1)

En los últimos tiempos se han realizado modificaciones en los contenidos de esta materia que responden al nuevo modelo de formación que se desarrolla en la UCI por lo que se trabaja en la creación de medios didácticos que contribuyan a la formación del estudiantado de manera amena, flexible y atractiva.

En virtud de satisfacer las necesidades existentes de materiales didácticos y sistemas que apoyen el aprendizaje de la asignatura de SO, se está desarrollando un laboratorio virtual que abarcará todos los temas de la misma, el cual contará con un módulo de ejercicios donde el estudiante fortalecerá los conocimientos adquiridos en los respectivos temas de la materia. Sin embargo, dentro de su concepción ninguna de las aplicaciones le proporcionan al alumno elementos que les permita evaluarse y poner en práctica el contenido estudiado específicamente en el tema de Gestión de Memoria.

Dada esta situación polémica se plantea el siguiente **problema investigativo:** ¿Cómo dotar al Laboratorio Virtual de SO (LAVSO) de un medio para evaluar los ejercicios correspondientes al tema de gestión de memoria posibilitándole al estudiante la comprobación de los contenidos adquiridos en clases?

Luego de un análisis del problema antes planteado se tiene como **objeto de estudio** de esta investigación el proceso de desarrollo de software, restringiendo el **campo de acción** al análisis, diseño e implementación de un evaluador del contenido de Gestión de Memoria para el LAVSO.

Como **objetivo general** se plantea Desarrollar un Evaluador que posibilite al estudiante la evaluación y comprobación de los contenidos adquiridos en clases.

Para el cumplimiento del objetivo propuesto se trazan los siguientes **objetivos específicos**:

- Elaborar el marco teórico de la investigación de manera que permita enfocar la misma a los términos que serán utilizados.

- Realizar los modelos de análisis y diseño de la solución.
- Realizar la implementación de la solución diseñada y validar los resultados obtenidos.

Se define como **idea a defender**: El desarrollo de un Evaluador del tema de Gestión de Memoria para el LAVSO le posibilitará al estudiante la evaluación y comprobación de los contenidos adquiridos en clases.

Para cumplir los objetivos específicos antes expuestos se realizarán a lo largo de todo el proceso investigativo las siguientes **tareas**:

1. Realización de un estudio referente al tema de Gestión de Memoria específicamente los algoritmos de remplazo.
2. Estudio de la metodología, lenguajes, herramientas y tecnologías necesarias para el desarrollo del Evaluador de Memoria para el LAVSO.
3. Estudio de los elementos relacionados con la Ingeniería de Requisitos, el diseño y la implementación del Evaluador de Memoria para el LAVSO.
4. Elaboración del documento Especificación de Requisitos.
5. Elaboración del documento modelo de Casos de Uso del Sistema.
6. Realización de los diagramas de secuencia, clases del diseño y diagrama de componentes
7. Implementación de la aplicación.
8. Aplicación de métricas, métodos que serán usados para medir la calidad de los artefactos obtenidos y pruebas para revisar el buen funcionamiento de la aplicación.

Para la realización del trabajo fue necesario la utilización de los siguientes **métodos y técnicas de investigación:**

Métodos teóricos:

- Análisis Histórico – Lógico: Para profundizar en los antecedentes de la utilización de las TIC en los procesos de enseñanza – aprendizaje y sus tendencias actuales.

- Analítico - Sintético: Se utiliza en la revisión bibliográfica, el estudio de reportes e informes sobre el estado de la infraestructura tecnológica de la UCI, la consulta de documentos rectores de la política de la UCI sobre la Informatización, la tendencia actual al aprendizaje autogestionado y la introducción de las TIC en el proceso de enseñanza - aprendizaje.

Métodos empíricos:

- Entrevista: Para la recopilación de información especializada o dirigida a directivos, profesores y alumnos que interactúan con las TIC en el proceso de enseñanza – aprendizaje presencial o mixto de la asignatura de SO en la Facultad 3 de la UCI.
- **Análisis Documental**: Se emplea para profundizar las bibliografías consultadas a lo largo de la investigación.

El presente trabajo consta de tres capítulos, en los cuales se desarrollan aspectos de importancia para lograr los objetivos que se persiguen:

Capítulo 1: Fundamentación teórica. En este capítulo se resume el estudio realizado sobre laboratorios virtuales, y los evaluadores, se aborda el tema de las metodologías de desarrollo de software, de los lenguajes de modelado y se fundamenta su selección para el proceso de desarrollo. También se conocen aspectos como las herramientas de desarrollo, así como los lenguajes de programación a utilizar.

Capítulo 2: Descripción de la solución propuesta. En este capítulo se describe la solución propuesta para el problema planteado mediante un modelo de dominio. Se describen además los requisitos establecidos y se definen los actores, casos de usos con sus descripciones así como sus diagramas y todo lo referente al análisis y diseño de la aplicación.

Capítulo 3: Implementación y Prueba. En este capítulo se aborda el tema referente a la construcción de la solución propuesta, mostrando un conjunto de artefactos generados en el flujo de trabajo Implementación, y se validan todos los artefactos obtenidos aplicando métricas para validar los requisitos del sistema, los casos de uso y las pruebas de caja negra aplicadas al sistema para comprobar el funcionamiento del mismo.

CAPÍTULO 1: Fundamentación Teórica

1.1 Introducción

En el siguiente capítulo se tratan diferentes temas entre los que se encuentran: concepto de laboratorio virtual, la variedad que existe de los mismos, sus ventajas y desventajas. De conjunto se estudia la evaluación, el evaluador del LAVSO y los sistemas existentes en el mundo. Partiendo del uso del paradigma de la programación orientada a objetos se realizará una descripción de la metodología, las herramientas y tecnologías seleccionadas para llevar a cabo el desarrollo del Evaluador de Memoria para el módulo de ejercicio.

1.2 Laboratorio Virtual

En el Laboratorio Tradicional (LT), los recursos en personas y espacios son restringidos, debido a su masificación y a problemas presupuestarios se requiere la presencia física del estudiante y la supervisión del profesor. Una solución a estos problemas la encontramos en la aplicación de los avances tecnológicos a la docencia e investigación universitaria y en concreto, el uso de Laboratorios Virtuales.

El Laboratorio Virtual (LV) acerca y facilita la realización de experiencias a un mayor número de alumnos, aunque alumno y laboratorio no coincidan en el espacio. Permite simular fenómenos, conceptos abstractos, mundos hipotéticos, controlar la escala de tiempo, entre otros, ocultando el modelo matemático y mostrando el fenómeno simulado de forma interactiva.

Trasladando este entorno a la enseñanza actual, los elementos necesarios para abordar la realización de actividades prácticas son los laboratorios virtuales, accesibles a través de Intranet, Internet o ambientes computacionales donde el alumno realiza las prácticas de una forma lo más similar posible a como si estuviese en las dependencias del laboratorio tradicional, simulando e interactuando con instrumentos virtuales.(2)

1.2.1 Conceptos

Según James P. Vary, un LV es un “espacio electrónico de trabajo concebido para la colaboración y la experimentación a distancia con objeto de investigar o realizar otras actividades creativas, y elaborar y difundir resultados mediante tecnologías difundidas de información y comunicación”.(3)

Julián Monge Nájera define un LV como “simulaciones de prácticas manipulativas que pueden ser hechas por el estudiante lejos de la universidad y el docente”.(4)

Luego analizar las definiciones expuestas, se entiende que un LV no es más que un entorno de simulación y experimentación a distancia, para realizar el conjunto de prácticas de laboratorio que son necesarias para poder superar las distintas asignaturas.

1.2.2 Ventajas e Inconvenientes

El uso de LV tiene sin duda muchos beneficios, permite al estudiante buscar información, en él se pueden relacionar fenómenos con sus consecuencias, se pueden repetir los eventos o fenómenos cuantas veces se requiera, se incorporan las tecnologías de la información y comunicación en las prácticas educativas y sociales para beneficio de los estudiantes. El aprendizaje está basado en simulaciones. Pero además de esto tiene otras ventajas más significativas como son: (2)

- Acerca y facilita a un mayor número de alumnos la realización de experiencias, aunque alumno y laboratorio no coincidan en el espacio.
- Permite simular situaciones que en realidad tendría escasas posibilidades de realizarlas.
- Repetir los eventos o fenómenos cuantas veces se requiera, sin miedo a sufrir o provocar un accidente.
- Pueden asistir al laboratorio cuando ellos quieran y elegir las áreas del laboratorio más significativas para realizar prácticas sobre su trabajo.
- El estudiante accede a los equipos del laboratorio a través de un navegador, pudiendo experimentar sin riesgo alguno y además, se flexibiliza el horario de prácticas y evita la saturación por el solapamiento con otras asignaturas.
- Reducen el coste del montaje y mantenimiento de los laboratorios tradicionales, siendo una alternativa barata y eficiente, donde el estudiante simula los fenómenos a estudiar como si los observase en el LT.

No todo son ventajas en los laboratorios virtuales, también existen inconvenientes. A continuación mostramos los más destacados:

- El LV no puede sustituir la experiencia práctica altamente enriquecedora del LT. Ha de ser una herramienta complementaria para formar a la persona y obtener un mayor rendimiento.
- En el LV se corre el riesgo de que el alumno se comporte como un mero espectador. Es importante que las actividades en el laboratorio virtual vengan acompañadas de un guión que explique el concepto a estudiar, así como las ecuaciones del modelo utilizado. Es necesario que el estudiante realice una actividad ordenada y progresiva, conducente a alcanzar objetivos básicos concretos.
- El alumno no utiliza elementos reales en el LV, lo que provoca una pérdida parcial de la visión de la realidad. Además no siempre se dispone de la simulación adecuada para el tema que el profesor desea trabajar.

La creciente complejidad de las actividades en el LT y el desarrollo de las TICs y la Computación, han hecho que los laboratorio virtual evolucionen, existiendo varios tipos según la bibliografía consultada, se proponen tres clasificaciones que se mencionan a continuación.

1.2.3 Tipos de Laboratorios Virtuales

Laboratorios virtuales software.

Los laboratorios virtuales de software, son laboratorios desarrollados como un programa de software independiente destinado a ejecutarse en la máquina del usuario, y cuyo servicio no requiere de un servidor Web. Es el caso de programas con instalación propia, que pueden estar destinados a plataformas Unix, Linux, M.S. Windows e incluso necesitar que otros componentes de software estén instalados previamente, pero que no necesitan los recursos de un servidor determinado para funcionar (como bases de datos o módulos de software de servidor).

Laboratorios virtuales Web.

En contraste con el anterior, este tipo de laboratorio se basa en un software que depende de los recursos de un servidor determinado. Estos recursos pueden ser bases de datos, software que requieren

ejecutarse en su servidor o la exigencia de determinado hardware para ejecutarse. No son programas que un usuario pueda descargar en su equipo para ejecutar localmente de forma independiente.

Laboratorios remotos.

Los laboratorios remotos son aquellos que permiten operar remotamente cierto equipamiento, bien sea didáctico como maquetas específicas, o industrial, además de poder ofrecer capacidades de laboratorio virtual. En general, estos requieren de equipos servidores específicos que les den acceso a las máquinas a operar de forma remota, y no pueden ofrecer su funcionalidad ejecutándose de forma local. Otro motivo que los hace dependientes de sus servidores es la habitual gestión de usuarios en el servidor. (Herías, 2003)

Se va a trabajar con los laboratorios virtuales web a causa de que en la asignatura Sistema Operativo se están dando los primeros pasos en la realización de un laboratorio virtual, el mismo se va a incorporar en el EVA y contará con un simulador y un evaluador que será una aplicación de escritrio, facilitando que cualquier profesional o alumno interesado en el tema desarrollado pueda hacer uso del mismo y beneficiarse de sus prestaciones.

1.3 Definiciones

Evaluación: el concepto de evaluación se refiere a la acción y a la consecuencia de evaluar, un verbo cuya etimología se remonta al francés *évaluer* y que permite indicar, valorar, establecer, apreciar o calcular la importancia de una determinada cosa o asunto.

Una evaluación también puede entenderse como un examen propuesto en el ámbito escolar para que el docente califique los conocimientos, las aptitudes y el rendimiento de sus alumnos.(5)

Sistema Operativo: es el software básico de una computadora que provee una interfaz entre el resto de programas del ordenador, los dispositivos hardware y el usuario. (6)

La parte del sistema operativo que se encarga de las tareas relacionadas con la administración de la Memoria Principal se llama Gestor de memoria. Existen varios algoritmos que facilitan este proceso.

- Técnicas de Asignación: se utilizan para administrar la memoria utilizando listas ligadas, este se basa en buscar huecos libres en la memoria para asignarle procesos según la técnica deseada (primer ajuste, siguiente ajuste, peor ajuste y mejor ajuste)
- Los Algoritmos de reemplazos: en caso de existir un fallo de página en la memoria el SO debe ser capaz de elegir una página para retirarla de la memoria y hacer un espacio para la página a recuperar, se utilizan algoritmos para realizar este proceso porque la página a retirar tiene que ser la indicada porque si está siendo utilizada constantemente se deduce que esa página no puede ser reemplazada.(6)

El módulo de ejercicios del LAVSO contará con Evaluador de memoria el cual le permitirá al estudiante comprobar los conocimientos adquiridos en clases respecto al tema de Gestión de memoria, el mismo le posibilita resolver los algoritmos explicados anteriormente y se le dará una nota por los resultados introducidos. Esto le facilitará el trabajo al profesor el cual tendrá acceso a esas calificaciones y podrá saber en qué estado se encuentran sus estudiantes y tendrá un mejor control sobre ellos.

1.3.1 Soluciones existentes:

Se realizó un estudio de soluciones que pudieran utilizarse como referencia para desarrollar el Evaluador de memoria y se obtuvieron los siguientes resultados:

- **Traductor de fórmulas a lenguaje o fórmula compilada**: en el año 2010 se desarrolló un componente evaluador de expresiones matemáticas en la UCI el cual permite evaluar fórmulas matemáticas y realiza el análisis léxico, sintáctico y semántico de la misma, generando el código intermedio o Fórmula Compilada.(7)
- **Software Evaluador De Derivadas:** es un software evaluador que le permite a los alumnos, tutores y otros poner en balance sus conocimientos. Medir los conocimientos en derivadas y emplear métodos más dinámicos para su aprendizaje. (8)
- **Software Evaluador de Proyectos de Inversión (S.E.P.I.)**: permite registrar unidades de negocios y asociarles proyectos de inversión. Cada proyecto de inversión podrá a su vez registrar los insumos necesarios para poder medir y estimar el grado en que se alcanzarán los objetivos

económicos planteados dentro de la realización del proyecto, ejecutando proyecciones de rentabilidad que servirán de base para determinar la viabilidad económica del proyecto. (9)

- **Evaluador de Prácticas de Gobierno Corporativo:** con el fin de apoyar a las empresas interesadas en evaluar sus prácticas de Gobierno Corporativo**,** la Corporación Andina de Fomento **(CAF**) pone a su disposición una aplicación de software, el cual permite obtener un diagnóstico de éstas y señalar las áreas prioritarias de acción. La aplicación permite a un usuario, conocedor o no de la materia, ubicar el grado de cumplimiento de una empresa determinada respecto a los estándares de **gobierno corporativo** contenidos en los **Lineamientos para un Código Andino de Gobierno Corporativo.** (10)

Las soluciones existentes a pesar de estar concurridas de funcionalidades tienen varias desventajas que imposibilitan que se utilicen para el desarrollo del Evaluador de memoria.

- La interfaz de la primera solución no permite modificación. La forma de evaluación de esta solución no se asemeja al Evaluador de memoria que se quiere desarrollar para la asignatura de SO que se imparte en la UCI.
- Son evaluadores cuyos procesos de evaluación están adaptados específicamente al negocio que tienen.
- La mayoría son software privativo.
- Cuentan con muy poca documentación lo que imposibilita el uso de estas soluciones para desarrollar el Evaluador de memoria para el LAVSO.

1.4 Proceso de Desarrollo del Software

El software según Pressman: “es la máquina que conduce a la toma de decisiones comerciales”. Los desarrolladores capacitados, preparados y responsables de construir el software se conocen como ingenieros del software.(11)

El proceso de desarrollo del software ha progresado ha medida que evolucionan las TICS aplicadas en la industria del software, el desarrollo del software se ha convertido en un elemento importante y fundamental para las personas, la industria y la economía de nuestro país, por lo que se persigue que se

contruyan software de alta calidad, por lo que surge la necesidad de aplicar conceptos, técnicas y métodos ingenieriles para el desarrollo de sistemas de software de alta calidad, que disminuyan los plazos de entrega y de costo, esto ha dado lugar a la aparición de la Ingeniería de Software (ISW), disciplina que surge para aplicar estos principios.

La ISW es una disciplina o área de la informática, que ofrece métodos y técnicas para desarrollar y mantener software de calidad. Aporta una serie de conocimientos que permiten comprender los procesos, datos, e interrelaciones que se establecen en un modelo real. Proporciona los elementos que hacen que la interacción con el usuario se torne más amigable, sin descuidar por ello la asignación de responsabilidades y la determinación de los roles de los participantes de un proyecto. A lo largo de la última década la ISW ha sufrido una evolución considerable en una gran parte de las distintas áreas que la constituyen, convirtiéndose en una disciplina ampliamente aceptada tanto a nivel académico como industrial.(11)

La ISW contiene varias disciplinas dentro de las que se encuentran: la planificación, ingeniería de requisitos, análisis y diseño, implementación, prueba y mantenimiento, estas son utilizada para dividir el desarrollo de proyectos de software, así el trabajo se realiza de forma organizada y con mejor calidad.

1.4.1 Ingeniería de Requisitos

Para lograr un mejor entendimiento de lo que es la Ingeniería de Requisitos (IR) y lo que representa, es necesario primero comprender qué son los requisitos, cómo están definidos y para qué sirven. Estas son algunas de sus definiciones: (12)

La IEEE define a un requisito como:

I. Una condición o capacidad necesaria para un usuario para resolver un problema o alcanzar un objetivo.

II. Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal.

III. Una representación documental de una condición o capacidad como en I o en II.

Los requisitos de software pueden dividirse en 2 categorías:

Requisitos Funcionales (RF): definen las funciones que el sistema será capaz de realizar, describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. Es importante que se describa el ¿Qué? y no el ¿Cómo? se deben hacer esas transformaciones. Además, son independientes de las tecnologías usadas por el producto.(Robertson, 2006)

Requisitos no Funcionales (RNF): son aquellos requisitos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste como la fiabilidad, el tiempo de respuesta y la capacitad de almacenamiento. (Sommerville, 2005)

1.4.1.1 Etapas de la ingeniería de requisitos.

La IR se divide en etapas lógicamente definidas, las cuales especifican los pasos, tareas y técnicas que se deben emplear para interactuar con los clientes y especificar correctamente el sistema.

Etapa: Elicitación de requisitos.

Es la etapa donde se adquiere el conocimiento del trabajo del cliente/usuario, se comprenden sus necesidades y se detallan las restricciones medioambientales. Como resultado de las acciones realizadas se obtiene el conjunto de los requisitos.

Esta etapa se divide en varias tareas que guían el trabajo de los especialistas, para así poder realizar una mejor recopilación de los requisitos. Primeramente se obtiene la información sobre el dominio del problema y el sistema actual, después se procede a preparar y realizar las reuniones de negociación con los clientes, con el objeto de obtener sus necesidades y resolver posibles conflictos que se hayan detectado en iteraciones anteriores, luego de ser realizadas las reuniones, se identifican y revisan los requisitos funcionales y no funcionales del sistema, y finalmente se priorizan, según su importancia para el futuro desarrollo de software.(11)

Etapa: Análisis de requisitos.

Los requisitos identificados se agrupan por categorías y se organizan en subconjuntos. Se estudia cada requisito en relación con el resto, se examinan los requisitos según su consistencia, completitud y ambigüedad, y se clasifican en base a las necesidades de los clientes/usuarios.(13)

Normalmente se realiza un análisis después de tener un bosquejo inicial del documento de requisitos; en esta etapa se leen los requisitos, se conceptúan, se investigan, se intercambian ideas con el resto del

equipo, se resaltan los problemas, se buscan alternativas y soluciones, y luego se van fijando reuniones con el cliente para discutir los requisitos identificados.

Etapa: Especificación de requisitos.

Una especificación puede ser un documento escrito, un modelo gráfico, un modelo matemático formal, una colección de escenarios de uso, un prototipo o una combinación de lo anterior. A través de la especificación de requisitos se puede negociar concretamente lo que el sistema debe hacer y cumplir para evitar retrasos y costes agregados por la mitigación de errores. Es el punto de partida de la estimación de costo, tiempo y esfuerzo del proyecto y tiene una incidencia marcada en los procesos de gestión de la calidad del producto a entregar.(13)

Etapa: Validación de requisitos.

La validación de requisitos tiene como misión demostrar que la definición de los requisitos define realmente el sistema que el usuario necesita o el cliente desea. Examina las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados hayan sido corregidos, y que el resultado del trabajo se ajusta a los estándares establecidos para el proceso, el proyecto y el producto.(13)

Etapa: Administración de Requisitos:

La administración de requisitos es un conjunto de actividades que ayudan al equipo de trabajo a identificar, controlar y seguir los requisitos y los cambios en cualquier momento. La administración de los requisitos incluye todas las actividades que mantienen la integridad y exactitud de los mismos a medida que el proyecto progresa.

En esta etapa se enfatizan:(13)

- Control de los cambios de los requisitos que están sobre la línea base definida.
- Control de versiones tanto de requisitos individuales como del documento de requisitos.

1.4.1.2 Técnicas de recopilación de información.

Las técnicas de recopilación de requisitos (TRI) son utilizadas para identificar las necesidades del cliente, las mismas requieren una adecuada planificación y preparación, existen diversas (TRI) se realizó un estudio de las que más se utilizan actualmente:

Observación: mediante esta técnica se observa al futuro usuario del sistema en su entorno de trabajo para saber qué tareas realiza actualmente y cuáles debe cubrir el sistema a desarrollar. Esta observación se hace especialmente importante y obligatoria si desconocemos el negocio del cliente o la operativa habitual para este tipo de usuarios, el cual dice siempre que es todo muy sencillo y se le van a olvidar esas particulares que luego serán muy difíciles de implementar.

Se pueden realizar filmaciones del lugar de trabajo, para luego observarlos cuidadosamente y obtener resultados del análisis. Siempre se debe estar atento a lo que sucede en el entorno de la organización, ver cómo resuelven un problema que surge, modo en que fluyen las operaciones.(11)

Entrevistas: dentro de las técnicas utilizadas para recopilar información, las entrevistas ocupan un lugar preponderante en consideración con el tiempo que ocupan y el objetivo que tienen. Por lo general, son la mayor fuente de información del analista. La entrevista es una forma de conversación, no de interrogación.

Las entrevistas formales o informales con los Stakeholders[1] del sistema son parte de la mayoría de los procesos de la IR. En estas entrevistas el equipo hace preguntas a los Stakeholders sobre el sistema que utilizan y sobre el sistema a desarrollar. Los requisitos provienen de las respuestas a estas preguntas.

Las entrevistas pueden ser de dos tipos.(14)

- **Entrevistas cerradas**: donde los Stakeholders responden a un conjunto predefinido de preguntas.
- **Entrevistas abiertas:** donde no hay un programa predefinido. El equipo examina una serie de cuestiones con los Stakeholders del sistema y, por lo tanto, desarrolla una mejor comprensión de sus necesidades.

[1]Stakeholders: cualquier persona o grupo que se verá afectado por el sistema, directa o indirectamente.(1)

Cuestionarios: requieren que el analista conozca el ámbito del problema en el que está trabajando. Consiste en redactar un documento con preguntas cuyas respuestas sean cortas y concretas, o incluso cerradas por unas cuantas opciones en el propio cuestionario. Este cuestionario será cumplimentado por el grupo de personas entrevistadas o simplemente para recoger información en forma independiente de una entrevista. (15)

Tormenta de ideas: es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios. Suele estar formada por un número de cuatro a diez participantes, uno de los cuales es el jefe de la sesión, encargado más de comenzar la sesión que de controlarla. Puede ayudar a generar una gran variedad de vistas del problema y a formularlo de diferentes formas, sobre todo al comienzo del proceso de elicitación, cuando los requisitos son todavía muy difusos. (11)

Prototipo de interfaz no funcional: Un cliente, a menudo, define un conjunto de objetivos generales para el software, pero no identifica los requisitos detallados de entrada, proceso y salida. En esta y en otras muchas situaciones, la construcción de prototipos puede ofrecer un mejor enfoque. El prototipo lo evalúa el cliente /usuario y se utiliza para refinar los requisitos del software a desarrollar. (11)

Sistemas existentes: Esta técnica consiste en analizar distintos sistemas ya desarrollados que estén relacionados con el sistema a ser construido. Por un lado, podemos analizar las interfaces de usuario, observando el tipo de información que se maneja y cómo es manejada, por otro lado también es útil analizar las distintas salidas que los sistemas producen (listados, consultas, entre otros), porque siempre pueden surgir nuevas ideas sobre la base de estas. (15)

1.4.2 Patrones

Los patrones son soluciones simples compuestos por una pareja problema/solución, fundamentadas en la experiencia para problemas específicos y comunes y que se ha demostrado que funcionan y pueden emplearse en diferentes contextos. (16) Ejemplo de estos: los patrones de casos de usos y del diseño.

1.4.2.1 Patrones de Casos de Usos

Después de ver estudiado la IR, es preciso realizar un estudio de algunos patrones de casos de uso, ya que los mismos son utilizados generalmente como plantillas que describen como debería ser estructurados y organizados los casos de uso.

Concordancia (Commonality): extrae una subsecuencia de acciones que aparecen en diferentes lugares del flujo de Casos de Uso y es expresado por separado. Este patrón se deriva en los siguientes patrones.

- Concordancia de Reusabilidad: consta de 3 Casos de Uso. El primero llamado subsecuencia común, modela una secuencia de acciones que aparecerán en múltiples Casos de Uso en el modelo. Los otros Casos de Uso modelan el uso del Sistema que comparte la subsecuencia común de acciones. De manera que deben existir al menos dos de ellos. Es un patrón de estructura.

- Concordancia de Adición: en el caso de este patrón alternativo, la subsecuencia común de Casos de Uso, extiende los Casos de Uso compartiendo la subsecuencia de acciones. Los otros Casos de Uso modelan el flujo que será expandido con la subsecuencia. Este patrón es preferible usarlo cuando otros Casos de Uso se encuentran propiamente completos, o sea, que no requieren de una subsecuencia común de acciones para modelar los usos completos del Sistema. Este es un patrón de estructura.

Extensión o Inclusión Concreta (Concrete Extension or Inclusion): el patrón es aplicable cuando un flujo puede extender o incluir el flujo de otro Caso de Uso, así como también puede ser realizado por el mismo. Ambos patrones derivados de este, son de tipo de estructura.

- Extensión Concreta: es de estructura, consiste en dos casos de uso y una relación de extensión. El caso de uso extendido es concreto, es decir, puede ser instanciado por su cuenta como por el caso de uso base. Es aplicable cuando un flujo puede extender el flujo de otro caso de uso, lo que significa que puede ocurrir el proceso del caso de uso base, o puede ocurrir el del caso de uso base con su caso de uso extendido.

- Inclusión Concreta: es de estructura, consiste en dos casos de uso y una relación de inclusión entre el caso de uso base y el caso de uso incluido. Este último puede ser instanciado por sí solo. El caso de uso base puede ser concreto o abstracto. Se utiliza cuando un flujo de datos puede ser incluido en el flujo de otro caso de uso y también puede ejecutarse por sí solo.

CRUD (Creating, Reading, Updating, Deleting): este patrón se basa en la fusión de Casos de Uso simples para formar una unidad conceptual. Tiene dos tipos de patrones el Parcial y el Completo, ambos son patrones de estructura.

- CRUD Completo: es un caso de uso llamado Información de CRUD o Administrar Información, que modela las diferentes operaciones que pueden realizarse en un pedazo de información de un cierto tipo, como crear, leer, actualizar, y eliminar. Debe usarse cuando todos los flujos contribuyen al mismo valor del negocio y estos son cortos y simples.

- CRUD Parcial: patrón alternativo que modela uno de las alternativas del caso del uso como un caso de uso separado. Es preferible cuando uno de las alternativas del caso del uso es más significativa, grande, o mucho más compleja que las otras alternativas.

Múltiples actores (Multiple Actors): Consiste de un Caso de Uso y por lo menos dos actores. Tiene dos variantes dado el rol de dichos actores.

- Múltiples actores Rol Común: puede suceder que los dos actores jueguen el mismo rol sobre el CU. Este rol es representado por otro actor, heredado por los actores que comparten este rol. Es aplicable cuando, desde el punto de vista del Caso de Uso, solo exista una entidad externa interactuando con cada una de las instancias del Caso de Uso.

- Múltiples actores Roles distintos: consiste en un caso de uso y al menos dos actores. Es usado cuando dichos actores juegan diferentes roles en relación con el caso de uso, es decir, ellos interactúan diferentemente con el caso de uso.

Estos patrones brindan la posibilidad de realizar un mejor y más entendible modelado del sistema. Esto es de gran importancia, pues este modelo es una entrada fundamental para realizar el diseño del

sistema, que es otra de las disciplinas en que se divide el desarrollo del software, en la misma se hará uso también de patrones de diseño para confeccionar los artefactos más entendibles para el programador.

1.4.2.2 Patrones de Diseño

Los patrones de diseño proponen soluciones exitosas a problemas comunes que se pueden presentar durante el diseño. Un patrón de diseño es una abstracción de una solución en un nivel alto. Los patrones solucionan problemas que existen en muchos niveles de abstracción. Hay patrones que abarcan las distintas etapas del desarrollo desde el análisis hasta el diseño y desde la arquitectura hasta la implementación. (17)

Patrones GRASP:

Los patrones de asignación de responsabilidades (General Responsibility Assignment Software Patterns, GRAPS) describen los principios fundamentales de la asignación de responsabilidades a objetos. Constituyen el fundamento de cómo se va a diseñar el sistema finalmente. Es importante que el diseñador de software domine y aplique estos conocimientos durante la realización de un diagrama de interacción. (17) **Principales patrones GRASP:**

Experto:

Solución: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Problema: ¿Cuál es el principio fundamental en virtud del cual se asignan las responsabilidades en el diseño orientado a objetos?

Este patrón se usa más que cualquier otro al asignar responsabilidades. El patrón Experto ofrece una analogía con el mundo real.

Creador:

Solución: Asignarle a la clase B la responsabilidad de crear una instancia de la clase A en uno de los siguientes casos:

- B agrega los objetos de A.

- B contiene los objetos de A.
- B registra las instancias de los objetos A.
- B utiliza específicamente los objetos A.
- B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado (así B es un Experto respecto a la creación de A).

Problema: ¿Quién debería ser el responsable de crear una nueva instancia de alguna clase?

El patrón Creador 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 debemos conectar con el objeto producido en cualquier evento.

Bajo acoplamiento:

Solución: Asignar una responsabilidad para mantener bajo acoplamiento.

Problema: ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización?

El Bajo acoplamiento es un principio que debemos recordar durante las decisiones del diseño: es la meta principal que es preciso tener siempre presente.

Alta cohesión:

Solución: Asignar una responsabilidad de modo que la cohesión siga siendo alta.

Problema: ¿Cómo mantener la complejidad dentro de límites manejables?

Alta cohesión es un principio de debemos tener presente en todas las decisiones de diseño: es la meta principal que ha de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño.

Controlador:

Solución: Asignar la responsabilidad del manejo de mensajes de los eventos del sistema a una clase que represente alguna de las siguientes opciones:

- El sistema global.
- La empresa u organización global.
- Algo activo en el mundo real que pueda participar en la tarea.

- Un manejador artificial de todos los eventos del sistema de un caso de uso (controlador de casos de uso).

Problema: ¿Quién debería encargarse de atender un evento del sistema?

Patrones GOF

Estos patrones surgen a raíz del trabajo de un grupo de autores conocido como el Gang of Four (GoF). Ellos recopilaron y documentaron un grupo de patrones de diseño aplicados usualmente por diseñadores de software orientado a objetos.

El grupo GoF agrupó los patrones en tres grandes categorías de acuerdo a su propósito. Este criterio describe la función que el patrón cumple. Los patrones de diseño pueden tener propósito creacional, estructural o de comportamiento:(17)

Patrones Creacionales: Se encargan de las formas de crear instancias de objetos. El objetivo de estos patrones es abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.

- **Factory Method:** centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear.
- **Abstract Factory:** permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.
- **Builder:** abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.
- **Prototype:** crea nuevos objetos clonándolos de una instancia ya existente.
- **Singleton:** garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

Patrones Estructurales: Describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.

Patrones de Comportamiento: Nos ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos. (17)

1.4.3 Métrica para evaluar el Diseño de Software

Luego de estudiar los patrones de diseño se hace necesario usar métricas para evaluar el diseño ya que ayudan a medir de forma cuantitativa la calidad de los atributos internos del software, esto permite evaluar la calidad durante el desarrollo del sistema.

Tamaño de clase (TC)

El tamaño general de una clase se puede determinar empleando las medidas siguientes:

- El número total de operaciones (tanto operaciones heredadas como privadas de la instancia) que están encapsuladas dentro de la clase.
- El número de atributos (tanto atributos heredados como atributos privados de la Instancia) que están encapsulados en la clase.

Si existen valores grandes de TC estos mostrarán que una clase puede tener demasiada responsabilidad, lo cual reducirá la reutilización de la clase y complicará la implementación y la comprobación, por otra parte cuanto menor sea el valor medio para el tamaño, más probable es que las clases existentes dentro del sistema se puedan reutilizar ampliamente. (11)

Estas métricas permiten evaluar el diseño, así como saber con anterioridad a qué clase, subsistema y módulo se le debe prestar más atención y designar más recursos para su implementación.

1.4.4 Pruebas de Software

La implementación se inicia a partir de los resultados obtenidos en el diseño y es casi la etapa final del desarrollo de software, por lo que para entregar un producto que cumpla con todas las necesidades requeridas por el cliente; es preciso realizar pruebas al Sistema exhaustivamente a través de una ejecución controlada antes de ser entregado al cliente.

1.4.4.1 Prueba de Caja Blanca

Las pruebas de caja blanca (también conocidas como pruebas de caja de cristal o pruebas estructurales) se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. (11)

Mediante la prueba de la caja blanca el ingeniero del software puede obtener casos de prueba que:

1. Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
2. Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
3. Ejecuten todos los bucles en sus límites operacionales.
4. Ejerciten las estructuras internas de datos para asegurar su validez.

Es por ello que se considera a la prueba de Caja Blanca como uno de los tipos de pruebas más importantes que se le aplican a los software, logrando como resultado que disminuya en un gran porciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad.

1.4.1.2 Pruebas de Caja Negra

Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

La prueba de Caja Negra no es una alternativa a las técnicas de prueba de la Caja Blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la Caja Blanca. Muchos autores consideran que estas pruebas permiten encontrar: (11)

1. Funciones incorrectas o ausentes.
2. Errores de interfaz.
3. Errores en estructuras de datos o en accesos a las Bases de Datos externas.
4. Errores de rendimiento.
5. Errores de inicialización y terminación.

Para preparar los casos de pruebas hacen falta un número de datos que ayuden a la ejecución de los estos casos y que permitan que el sistema se ejecute en todas sus variantes, pueden ser datos válidos o inválidos para el programa según si lo que se desea es hallar un error o probar una funcionalidad. Los

datos se escogen atendiendo a las especificaciones del problema, sin importar los detalles internos del programa, a fin de verificar que el programa corra bien.

1.5 Metodologías de desarrollo

La implantación de una metodología es necesaria si se quieren gestionar adecuadamente los proyectos. Aplicar una metodología de desarrollo de Software en una organización es una tarea difícil y compleja. El éxito en su utilización depende de múltiples factores, antes de decidirse por una en función de sus características se debe reflexionar acerca de sus aspectos determinantes, así como las ventajas sobre las otras.

Dichas metodologías pretenden guiar a los desarrolladores al crear un nuevo software, pero los requisitos de un software a otro son tan variados y cambiantes, que ha dado lugar a que exista una gran variedad de metodologías para la creación del software.

Se han desarrollado dos corrientes en lo referente a los procesos de desarrollo, los llamados métodos pesados y los métodos ligeros.

La diferencia fundamental entre ambos es que mientras los métodos pesados intentan conseguir el objetivo común por medio de orden y documentación, los métodos ligeros (también llamados métodos ágiles) tratan de mejorar la calidad del software por medio de una comunicación directa e inmediata entre las personas que intervienen en el proceso.(16)

1.5.1 Proceso Unificado de Desarrollo de Software (RUP).

El proceso unificado de desarrollo de software (RUP, del inglés Rational Unified Process) es uno de los procesos más generales de los existentes actualmente, es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo y trabajo de muchas metodologías utilizadas por los clientes, está pensado para adaptarse a cualquier proyecto y no tan solo de software.

RUP es una metodología pesada, centrada en la arquitectura, iterativa e incremental y dirigida por casos de uso, el cual designa tareas y responsabilidades. Asegura la producción de Software de alta calidad, capaz de ajustarse a las necesidades de los usuarios finales con un costo y un calendario predecible.

RUP sería entonces un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas, utilizando el Lenguaje Unificado de Modelado (Unified Modeling Language, UML) para preparar todos los esquemas de un sistema de software.

RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades.

Para la realización de un proyecto RUP se divide en 4 fases:(18)

- Fase de Inicio (Objetivos): se describe el negocio, se delimita el proyecto describiendo sus alcances con la identificación de los casos de uso del sistema, se identifican y priorizan los riesgos más importantes, se planifica en detalle la fase de elaboración y se estima el proyecto de manera aproximada.
- Elaboración (Arquitectura):se especifican en detalle la mayoría de los casos de uso del producto, se desarrollan los casos de usos más críticos que se identificaron en la fase de comienzo y se diseña la arquitectura.
- Construcción (Funcionalidad Operativa): se obtiene un producto listo para su utilización, conteniendo todos los casos de usos que fueron acordados entre las partes implicadas para la elaboración del software, documentado y con un manual de usuario.
- Transición (Release o Liberación del Sistema): se corrigen problemas o se incorporan mejoras sugeridas, para lograr que el producto de software se encuentre listo para su instalación en condiciones reales.

RUP está compuesto por 9 flujos de trabajo, los 6 primeros de ingeniería y los 3 últimos de soporte.(18)

1. Modelamiento del negocio: de los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
2. Requerimientos: define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
3. Análisis y Diseño: describe cómo el sistema será realizado a partir de los requisitos, por lo que indica con precisión lo que se debe programar.

4. Implementación: define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.
5. Prueba (Testeo): busca los defectos a lo largo del ciclo de vida.
6. Instalación: produce release del producto y realiza actividades (empaque, instalación, asistencia a usuarios.) para entregar el software a los usuarios finales.
7. Administración del proyecto: involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.
8. Administración de configuración y cambios: describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, entre otros.
9. Ambiente: contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.

Ventajas:

1. Evaluación en cada fase que permite cambios de objetivos.
2. Funciona bien en proyectos de innovación.
3. Es sencillo, ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el software.
4. Seguimiento detallado en cada una de las fases.

Desventajas:

1. La evaluación de riesgos es compleja.
2. Excesiva flexibilidad para algunos proyectos.
3. Estamos poniendo a nuestro cliente en una situación que puede ser muy incómoda para él, ya que deberá ser capaz de describir y entender a un gran nivel de detalle para poder acordar un alcance del proyecto con él.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

1.6 Lenguajes de Modelado

No basta con la selección de la metodología y seguir los pasos que estas sugieren para poder llevar a cabo el desarrollo de un software, es preciso graficar las acciones que se identifican durante la captura de requisitos para una mejor comprensión de estas, es por ello que se requiere de un lenguaje para poder modelar estas funciones, el cual pueda ser comprendido por todas las personas que intervienen en el proceso de desarrollo del Software.(15)

En cualquier proyecto de ingeniería como la construcción de un gran edificio, un avión, la construcción de un procesador de textos o un software de comunicaciones para Internet, requieren de etapas de modelado[2] que permitan experimentar y visualizar el sistema que se construirá.

1.6.1 UML (2.0).

UML[3] es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. UML es un grupo de especificaciones de notación orientadas a objeto, las cuales están compuesta por diversos elementos gráficos que se combinan para conformar distintos diagramas, que representan las diferentes etapas del desarrollo de un proyecto de software.

La clave está en organizar el proceso de diseño de tal forma que los analistas, clientes, desarrolladores y otras personas involucradas en el desarrollo del sistema lo comprendan y convengan con él. El UML proporciona tal organización.

El UML está constituido por un conjunto de diagramas, y proporciona un estándar que permite al analista de sistemas generar un anteproyecto de varias facetas que sean comprensibles para los clientes, desarrolladores y todos aquellos que están involucrados en el proceso de desarrollo. Es necesario contar

[2] **Modelado:** Es el diseño de un proyecto antes de su construcción, es la visualización de lo que se quiere construir. (Geraldo)

[3] **UML:** Lenguaje Unificado de Modelado

con todos esos diagramas dado que cada uno se dirige a cada tipo de persona implicada en el sistema. Un modelo UML indica qué es lo que supuestamente hará el sistema, mas no cómo lo hará. (19)

Ventajas:

- Es un lenguaje de modelado de propósito general que pueden usar todos los modeladores, no es un método de desarrollo. No te va a decir cómo pasar del análisis al diseño y de este al código.
- Es independiente del ciclo de desarrollo que vayas a seguir, soporta metodologías de desarrollo de Software como por ejemplo RUP, pero no especifica en sí mismo, qué metodología o proceso utiliza.
- No tiene propietario y está basado en el común acuerdo de gran parte de la comunidad informática.
- Pretende abordar los problemas actuales del desarrollo de software, tales como gran tamaño, distribución, concurrencia, patrones, y desarrollo en equipo.
- Puede ser utilizado con cualquier metodología a lo largo del proceso del desarrollo del software y en cualquier plataforma tecnológica donde se implementara, ya sea Unix, Windows. (19)

Los lenguajes y notaciones utilizados en la modelación de procesos juegan un papel de suma importancia pues son los encargados de hacer posible que los artefactos sean entendidos por todas las personas que intervienen desde los diseñadores, analistas, desarrolladores y los que los controlan y dirigen.

1.7 Herramientas CASE[4].

Las herramientas CASE son diversas aplicaciones informáticas que sirven de apoyo en todo el ciclo de vida del proceso de desarrollo de software. Estas herramientas además de agilizar el trabajo, mejorar y

[4] **CASE:** Computer Assisted Software Engineering, Ingeniería de Software Asistida por Ordenador

estandarizar la documentación y facilitar la realización de prototipos; contribuyen a mejorar la calidad y la productividad en el desarrollo de software. (19)

1.7.1 Visual Paradigm (VP) 5.0.

Es una herramienta CASE profesional que soporta el ciclo de vida completo del desarrollo de Software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue.

Soporta modelado con UML 2.0 y todos sus diagramas, permite mediante ingeniería directa, la generación de código en varios lenguajes de programación como C#, VB.NET. Además Visual Paradigm for UML permite la ingeniería inversa para archivos, entre otros.(20)

Características de VP 5.0

1. Diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.
2. Permite el modelado de base de datos y la confección de los prototipos de interfaz.
3. Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
4. Capacidades de ingeniería directa e inversa.
5. Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
6. Disponibilidad de múltiples versiones, para cada necesidad.
7. Disponibilidad de integrarse en los principales IDEs.
8. Disponibilidad en múltiples plataformas.
9. Esta versión genera modelos y reportes con una calidad mejor a las versiones anteriores.

1.8 Lenguajes de Programación

Para desarrollar un software es necesario una serie de requisitos, entre ellos se encuentra el conjunto de símbolos y de reglas tanto sintácticas como semánticas que indican al ordenador qué acciones desarrollar, a esta aglomeración de instrucciones se le llama lenguaje de programación. (Almaguer, 2009).

1.8.1 Java

Es un lenguaje de programación orientado a objetos desarrollado a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Características principales:(21)

- Orientado a Objetos: Java organiza sus programas en una colección de objetos. Esto nos va a permitir estructurar los programas de una manera más eficiente y en un formato más fácil de comprender.
- Distribuido: Java dispone de una serie de librerías para que los programas se puedan ejecutar en varias máquinas y puedan interactuar entre sí.
- Robusto: Java está diseñado para crear software altamente fiable.
- Seguro: Java cuenta con ciertas políticas que evitan que se puedan codificar virus con este lenguaje, sin olvidar además que existen muchas otras restricciones que limitan lo que se puede o no se puede hacer con los recursos críticos de una máquina.
- Interpretado y compilado a la vez: Java puede ser compilado e interpretado en tiempo real, ya que cuando se construye el código fuente este se transforma en una especie de código de máquina.
- Independiente de la Arquitectura: El código compilado de Java se va a poder usar en cualquier plataforma.
- Multiejecución: Java permite elaborar programas que permitan ejecutar varios procesos al mismo tiempo sobre la misma máquina.

1.9 Herramientas de desarrollo

Las herramientas de desarrollo son todos aquellos programas o aplicaciones que tienen cierta importancia en el desarrollo de un programa (programación), ya que con la ayuda y posibilidades que brindan se desarrollará mejor el programa o software.

1.9.1 NetBeans. 7.1

NetBeans es un proyecto de código abierto fundado y patrocinado por SunMicroSystems, es un IDE[5] que permite escribir, compilar, corregir errores y ejecutar programas. Permite crear aplicaciones de escritorio, web y para dispositivos móviles. Soporta además lenguajes dinámicos como PHP, Java Script, Groovy y Ruby. Admite desarrollar aplicaciones usando la plataforma J2EE. Puede ser usado en varias plataformas entre las que se encuentran Windows, Linux, Mac OS X y Solaris.

Después de hacer un minucioso estudio de las herramientas, para el desarrollo de la aplicación se hará uso de NetBeans debido a que es libre y de código abierto, tiene una interfaz muy amigable e intuitiva, característica fundamental en un IDE, tiene todas las herramientas para crear aplicaciones profesionales ya sean de escritorio, empresariales, web, móviles y aplicaciones SOA, no solo en Java sino también en C/C++ y Ruby, puedes hacer diagramas UML y es de múltiples lenguajes.

1.10 Conclusiones Parciales

Para el desarrollo del Capítulo se obtuvo que los LV en especial el Evaluador de Memoria va a ser una parte muy importante del proceso de enseñanza-aprendizaje en la UCI, específicamente para la asignatura de SO ya que le permitirá a los estudiantes la comprobación de los conocimientos adquiridos.

Se realizó un estudio del proceso desarrollo del software que se va a realizar donde se determinó:

- Aplicar las etapas de elicitación, análisis, especificación y validación de los requisitos, desarrollando cada una de las actividades que estas proponen en la ingeniería de requisitos.
- Aplicar patrones de casos de uso, permitiendo la elaboración del diagrama de casos de uso, además permitirán resolver problemas que se puedan presentar a la hora de modelar el sistema de forma rápida y ágil.
- Aplicar los patrones de diseño necesarios para no cometer errores tradicionales en el diseño.

Se profundizó en las principales metodologías y herramientas de desarrollo que se definieron en el documento de la Arquitectura del LAVSO, las mismas posibilitarán una mayor productividad para el desarrollo del Evaluador.

[5] **IDE:** integrated development environment (Entorno de Desarrollo Integrado)

Capítulo 2: Descripción y validación de la solución propuesta

2.1 Introducción

En el presente capítulo se realiza el modelo de dominio del cual se obtiene una mayor claridad de lo que se quiere automatizar y definir conceptos claves del negocio para que exista un lenguaje único entre los involucrados. También se especifican los requisitos funcionales y los no funcionales por los cuales se regirá el sistema propuesto, se identificarán mediante el diagrama de casos de uso del sistema las relaciones entre sus actores y los casos de uso así como la descripción de los mismos, además en el análisis definido por la metodología de desarrollo escogida se tuvieron en cuenta los requisitos funcionales durante la confección de los diagramas de clases. El diseño consolidará el análisis propuesto con los diagramas de clases y la aplicación de patrones de diseño como una manera más práctica de describir ciertos aspectos; teniendo en cuenta las cualidades o propiedades que el sistema debe tener.

2.2 Modelo de Dominio

La metodología seleccionada (RUP), como metodología de desarrollo propone dos opciones para realizar el modelado del negocio de un sistema, uno es por casos de usos del negocio y la otra es por clases del domino, conocido como modelo de dominio que se utiliza cuando el negocio no está bien definido entre los clientes y los ejecutores del sistema.

El modelo de dominio es una de las vías que brinda el Proceso Unificado para la comprensión del contexto del sistema, capturando los tipos más importantes de objetos o eventos dentro del entorno donde este se encontrará. Los objetos del dominio representan “cosas” que existen o los eventos que suceden en el entorno en el que trabaja el sistema. (17)

El objetivo fundamental del modelo de dominio es comprender y describir las principales clases dentro de lo que abarca el sistema, posibilitando, a los involucrados en el desarrollo del producto ya sean clientes, desarrolladores e interesados, utilizar un vocabulario común el cual permita capturar correctamente los requisitos y por ende obtener el producto deseado.

2.2.1 ¿Por qué modelo de dominio?

Se decide realizar el modelo de dominio, debido primeramente a que no existe un negocio definido, por lo cual, no se pueden determinar los procesos y roles del proceso de negocio, haciéndose engorroso y poco exacto la descripción de los mismos, segundo que este modelo ayuda a comprenderlos conceptos que utilizan los usuarios, los conceptos con los que trabajan y con los que deberá trabajar nuestra aplicación, por último no están definidas las actividades y procesos a automatizar, que el proceso en desarrollo está altamente centrado en herramientas y tecnologías informáticas. Teniendo en cuenta lo expuesto anteriormente se determina que no es viable llevar a cabo un modelo de negocio y se hace necesario realizar la descripción de la aplicación mediante un modelo de dominio o modelo de conceptos del dominio, el cual agrupa los conceptos fundamentales de objetos relacionados con el dominio, sus atributos y sus relaciones.

2.2.2 Conceptos del Modelo del dominio del sistema:

Concepto	Descripción
Usuario Común	Puede ser un estudiante, profesor o cualquier usuario autenticado por el EVA.
Laboratorio Virtual de Sistemas Operativos.	Entorno virtual que permite gestionar el proceso de enseñanza - aprendizaje para la asignatura Sistema Operativos.
Subsistema de Ejercicios.	Subsistema en el que se gestionan los ejercicios para evaluar a los estudiantes.
Simulador	Módulo que se encarga de resolver los ejercicios seleccionados por el estudiante.
Evaluador	Módulo que se encarga de comparar los resultados del simulador con los introducidos por el usuario común y evaluarlos.
Ejercicios.	Colección de ejercicios disponibles para que el estudiante ponga en prácticas sus conocimientos.
Algoritmos de Remplazos	Tipo de ejercicios que serán resueltos y evaluados por el Evaluador.
Técnicas de Asignación	Tipo de ejercicios que serán resueltos y evaluados por el Evaluador.

Tabla 1 Conceptos del modelo de dominio

2.2.3 Diagrama del Modelo del Dominio:

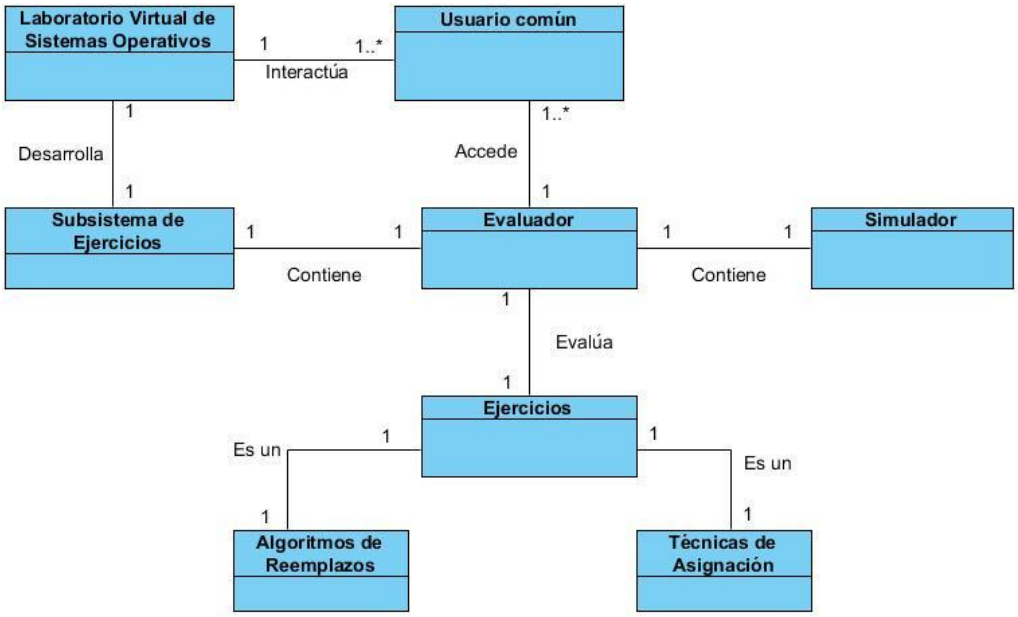


Figura 1 Modelo de Dominio

Los conceptos asociados al modelo de dominio se relacionan de la siguiente manera; el **usuario común** interactúa con el **Laboratorio Virtual** donde podrá resolver ejercicios en el **evaluador** que el mismo contienes un **simulador** que solucionará los ejercicios y luego evaluará las respuestas devueltas por el simulador con las introducidas por el usuario común, estos ejercicios a resolver solo serán algoritmos del tema de Gestión de memoria explícitamente las **Técnicas de Asignación** y los **Algoritmos de reemplazos**.

Luego de conocer los conceptos asociados al modelo de dominio, queda abierta la pregunta: ¿Qué debe hacer el sistema para que se cumplan los objetivos específicos planteados? La respuesta la podemos encontrar en una lista enumerada de requerimientos.

2.3 Especificación de los requerimientos del software

Un proyecto no puede ser exitoso sin una descripción detallada, correcta y exhaustiva de los requerimientos, estos definen lo que debe hacer un sistema y la forma en que debe hacerlo.

En la etapa de identificación de requisitos se emplearon algunas de las técnicas de recopilación de información abordadas en el epígrafe 1.4.1.2 como son: entrevistas, tormenta de ideas, prototipo de interfaz no funcional, sistemas existentes.

Tanto las entrevistas como los prototipos de interfaz no funcional, permitieron generar nuevas funcionalidades que el Sistema debía poseer y permitió determinar posibles Requisitos, a través de su inspección y análisis.

2.3.1 Requisitos funcionales:

Los Requisitos Funcionales se clasifican según su prioridad en Alta (Esencial), Media (Deseado) o Baja (Opcional), en función de las necesidades del cliente. Los requisitos levantados para el desarrollo del sistema se clasificaron de prioridad alta.

No	Requisitos funcionales
1	Inicializar memoria física
2	Almacenar estado de la memoria física
3	Almacenar proceso en memoria
4	Mostrar grado de multiprogramación
5	Asignar nuevo proceso a memoria
6	Administrar huecos libres
7	Obtener lista de huecos
8	Visualizar cola de entrada de procesos y páginas
9	Mostrar la página a partir de una dirección lógica
10	Configurar memoria física
11	Configurar memoria virtual
12	Seleccionar política de reemplazo
13	Seleccionar algoritmo de reemplazo

14	Reemplazar página en memoria física
15	Capturar respuesta del estudiante
16	Comparar resultados con simulación de algoritmo
17	Visualizar errores
18	Emitir nota

Tabla 2 Requisitos Funcionales

2.3.2 Requisitos no funcionales

A continuación se muestran los requerimientos no funcionales los cuales fueron agrupados según las categorías de usabilidad, robustez, accesibilidad, fiabilidad, soporte e interfaz:

Apariencia o interfaz externa:

- **RNF1:** El diseño de la interfaz debe ser sencillo y fácil de usar con reconocimiento visual a través de elementos visibles que identifiquen cada una de sus acciones.
- **RNF2:** La combinación de colores debe ser agradable a la vista del usuario.
- **RNF3:** Debe contar con un vínculo a la ayuda en la ventana principal del trabajo.

Usabilidad:

- RNF4: El sistema puede ser usado por cualquier estudiante que posea conocimientos básicos sobre el funcionamiento interno de un Sistema Operativo.

Rendimiento:

- **RNF5:** La respuesta a solicitudes más complejas de los usuarios del sistema no debe exceder 9 segundos.
- **RNF6**: La aplicación deberá estar disponible las 24 horas del día.

Portabilidad:

- **RNF7**: Debe poder ejecutarse tanto en Sistemas Operativos desde Windows XP Profesional Service Pack 1 o Superior como en Sistemas Operativos Linux.

Hardware:

- **RNF8:** Tarjeta de memoria RAM de 128 MB o superior.
- **RNF9:** Procesador Pentium II o superior a 250 MHz como mínimo.
- **RNF10:** Computadora cliente de 40 Gb de disco duro o superior.

Restricciones en el Diseño y la Implementación

- **RNF11**: Se utiliza UML como lenguaje de modelado.
- **RNF12**: Se usará como lenguaje de programación Java.

2.4 Modelo de Casos de Uso del Sistema

El modelo de casos de uso captura todos los requisitos funcionales del sistema, permitiendo que los desarrolladores de software, usuarios e interesados lleguen a un acuerdo sobre los mismos, es decir, acordar las condiciones y posibilidades que debe cumplir el subsistema.

Los actores son el entorno del sistema, comunicándose con este mediante el envío y recepción de mensajes. Por su parte los casos de uso especifican el sistema y están diseñados para cumplir los deseos del usuario cuando utiliza el subsistema. (18)

2.4.1 Actores del Sistema

Los actores representan personas, sistemas o hardware externo que interactúan con el sistema, o sea, los actores son agentes externos al sistema los cuales tienen relación con el mismo. La identificación de los actores del sistema, determina el entorno externo del sistema. (Pressman, 2005)

Actor del Sistema	Descripción
Usuario común	El usuario común es el encargado de realizar cualquier operación dentro del sistema, como inicializar la memoria física, evaluarse en cualquier técnica de asignación o algoritmos de reemplazos. Este puede ser el estudiante, el profesor o cualquier usuario que tenga conocimientos del

	tema, requiera de su uso y esté autenticado en el EVA.

Tabla 3 Actores del sistema

2.4.2 Diagrama de Casos de Uso del Sistema

Cada forma que los actores usan el sistema se representa con un caso de uso, los mismos son fragmentos de funcionalidad del sistema y especifican una secuencia de acciones, incluyendo alternativas dentro de la secuencia, que este puede llevar a cabo, y que producen un resultado observable de valor para un actor concreto. (Pressman, 2005)

Los diagramas de casos de uso sirven para especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y otros sistemas. (Rodríguez Urrutia, y otros, 2010)

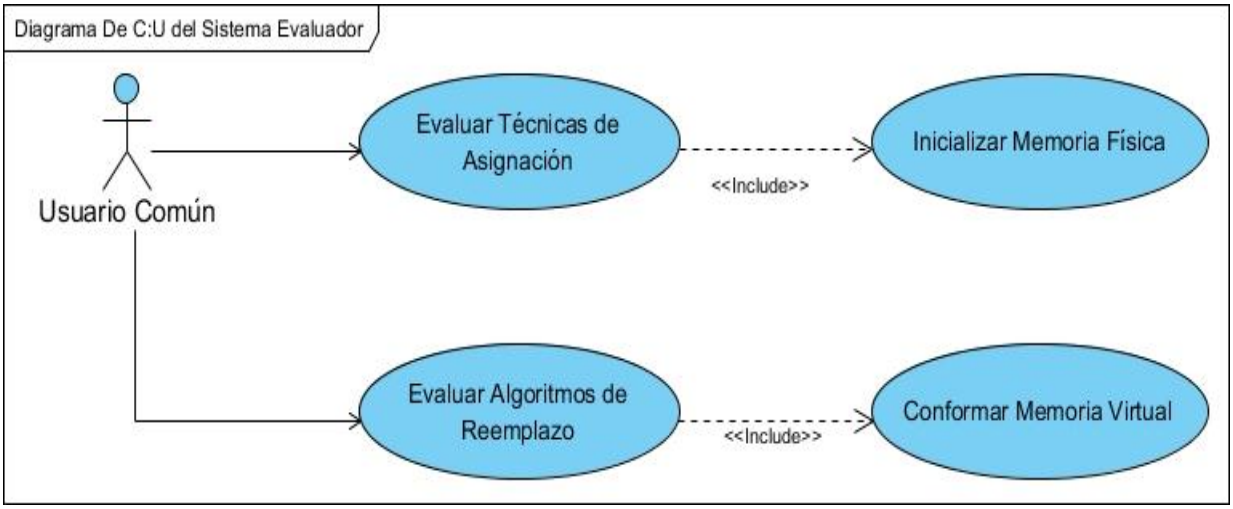


Figura 2 Modelo de Casos de usos del sistema

Para realizar el diagrama de caso de usos del evaluador se dividió en cuatro procesos:

Evaluar Técnicas de Asignación: El usuario común entrará al sistema para resolver un ejercicio referente a un técnica de asignación donde primeramente deberá Inicializar la Memoria Física que es uno de los procesos que se explicará más adelante, seguido a esto el sistema simulará internamente el ejercicio, donde luego de esto el usuario deberá introducir los resultados obtenidos para que el sistema compare los resultados del estudiante con los de la simulación donde le dará una nota, y en caso de tener errores lo mostrará la cantidad de errores y se los señalará en rojo.

Inicializar Memoria Física: Se ejecuta a través del caso de uso Evaluar Técnicas de Asignación donde el usuario inicializará la memoria virtual introduciendo los datos necesarios donde el sistema los validará inicialmente y si son correctos procederá a simular internamente la técnica de asignación seleccionada.

Evaluar Algoritmos de Reemplazos: El usuario común entrará al sistema para resolver un ejercicio referente a un algoritmo de reemplazo donde primeramente deberá Conformar la Memoria Virtual que es uno de los procesos que se explicará más adelante, seguido a esto el sistema simulará internamente el ejercicio, donde luego de esto el usuario deberá introducir los resultados obtenidos para que el sistema compare los resultados del estudiante con los de la simulación donde le dará una nota, y en caso de tener errores lo mostrará la cantidad de errores y se los señalará en rojo.

Conformar Memoria Virtual: Se ejecuta a través del caso de uso Evaluar Algoritmos de Reemplazos donde el usuario inicializará la memoria virtual introduciendo los datos necesarios donde el sistema los validará inicialmente y si son correctos procederá a simular internamente la técnica de asignación seleccionada.

2.4.3 Patrones de Casos de Uso del Sistema:

Los patrones son soluciones simples compuestos por una pareja problema/solución, fundamentadas en la experiencia para problemas específicos y comunes y que se ha demostrado que funcionan y pueden emplearse en diferentes contextos. (Overgaard, y otros, 2004).

Para realizar el diagrama de casos de usos se utilizó solamente el patrón Inclusión Concreta, que fue explicado en el epígrafe 1.4.2, debido a que no existe gran tráfico de funcionalidades, el mismo se evidencia en los procesos Evaluar Técnicas de asignación y Evaluar Algoritmos de Reemplazos, que serían los casos de usos bases, para su ejecución necesitan obligatoriamente que se ejecuten las funcionalidades Inicializar Memoria Física para el caso de que se desee ejecutar la primera funcionalidad y Conformar Memoria Virtual para el otro caso, donde estas dos funcionalidades serían los casos de uso incluidos, el uso de este patrón aumenta la productividad y disminuye la reutilización de elementos existentes (en este caso fragmentos de modelos).

2.4.4 Descripción de Casos de Uso del Sistema

Mediante las descripciones de los casos de uso se especifica la secuencia de eventos que los actores utilizan para completar un proceso a través del sistema. Aquí se muestra la descripción del caso de uso

Evaluar Algoritmos de Reemplazos, las demás descripciones de los casos de uso del sistema así como los prototipos de interfaz podrán ser consultadas los Anexos del trabajo.

Caso de Uso:	Evaluar Algoritmos de Reemplazos
Actores:	Usuario común (UC)
Resumen:	El caso de uso inicia cuando el usuario común selecciona uno de los algoritmos de reemplazo, introduce los resultados del algoritmo a evaluar y termina cuando el sistema resuelve el algoritmo y le da una nota al usuario.
Precondiciones:	Conformar Memoria Virtual
Pos condición:	Almacenar resultados del usuario común.
Casos de usos asociados:	Caso de uso incluido Conformar memoria virtual
Referencias	RF11, RF12, RF13, RF14, RF15, RF16, RF17, RF18
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El UC selecciona la opción Algoritmos de reemplazos.	2. El sistema muestra la interfaz Algoritmos de reemplazos con los botones Conformar memoria virtual y salir activados.
3. Si el UC selecciona Salir ir al paso 10 del flujo normal de eventos, de lo contrario selecciona el botón Conformar memoria virtual	4. Invoca el CU incluido Conformar Memoria Virtual.
	5. El sistema muestra una tabla donde el usuario introducirá los resultados del algoritmo y activa el botón evaluar.
6. El UC introduce los resultados y oprime evaluar.	7. El sistema evalúa los resultados introducidos por el UC.
	8. El sistema muestra un mensaje con la nota y

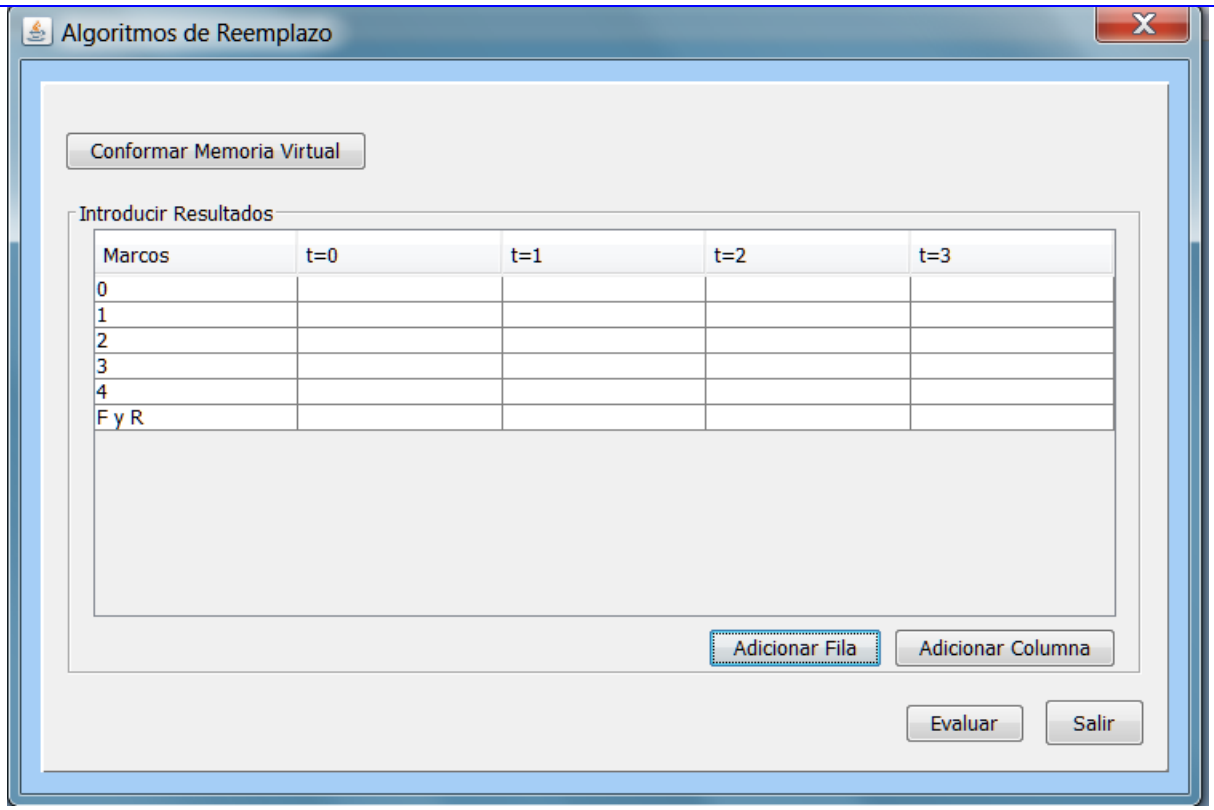
	la cantidad de errores cometidos.
	9. El sistema envía la nota del UC al módulo de ejercicios.
	10. Terminando así el Caso de Uso.
Prototipo de Interfaz	
	

Tabla 4 Descripción de Casos de usos del sistema

2.5 Análisis del Sistema

El modelo del análisis es un artefacto que usualmente se genera para entender mejor los requisitos y realizar mejor el diseño. Es típicamente un artefacto temporal, que brinda una aproximación al diseño y se realiza normalmente cuando no se tiene una idea clara de los procesos y requerimientos. (Jacobson, y otros, 2000)

Aunque el análisis ofrece una visión general del sistema, que puede ser más difícil de obtener mediante el estudio de los resultados del diseño, este no abarca el entorno de implementación que sí se tiene en cuenta en el diseño, permitiendo gestionar mejor el avance del proceso de desarrollo.

Se realiza el análisis para lograr un mejor acercamiento a lo que se desea construir, para sentar las bases del diseño. Luego de un estudio realizado, se decide no realizar el modelo de análisis pues los procesos y requerimientos están bien definidos, no contamos con un sistema de gran envergadura ni con tiempo suficiente para la construcción del mismo.

2.6 Diseño del sistema

El diseño es una de las actividades técnicas necesarias para la elaboración del software. Entre las tareas fundamentales del diseño están: producir el diseño de datos, diseño arquitectónico, diseño de interfaz y diseño de componentes. El diseño propuesto tiene que cumplir en totalidad con los requerimientos del sistema, debe ser capaz de facilitar las mejoras del software, debe especificarse, de forma tal que sea entendible por otros diseñadores y no diseñadores, servir como guía para los demás flujos de la ingeniería de software, permitir la comprobación del sistema fácilmente.

2.6.1 Diagrama de clases del diseño

Dentro de los principales elementos que se pueden encontrar en un diagrama de clases de diseño se encuentran: clases, asociaciones y atributos, interfaces, con sus operaciones y constantes, métodos, información sobre los tipos de atributos, navegabilidad y dependencias.

Los diagramas de diseño del Evaluador se dividieron en dos procesos fundamentales Evaluar Técnicas de Asignación y Evaluar Algoritmos de Reemplazos cuenta con una clase controladora ReemplazoControler la cual se dedica a simular todos los algoritmos del tema y evalúa los resultados introducidos por el estudiante. Mostrándose en la siguiente figura esta clase con sus principales relaciones pudiendo encontrarse en los Anexos del trabajo los diagramas de clases del diseño completos de los dos procesos.

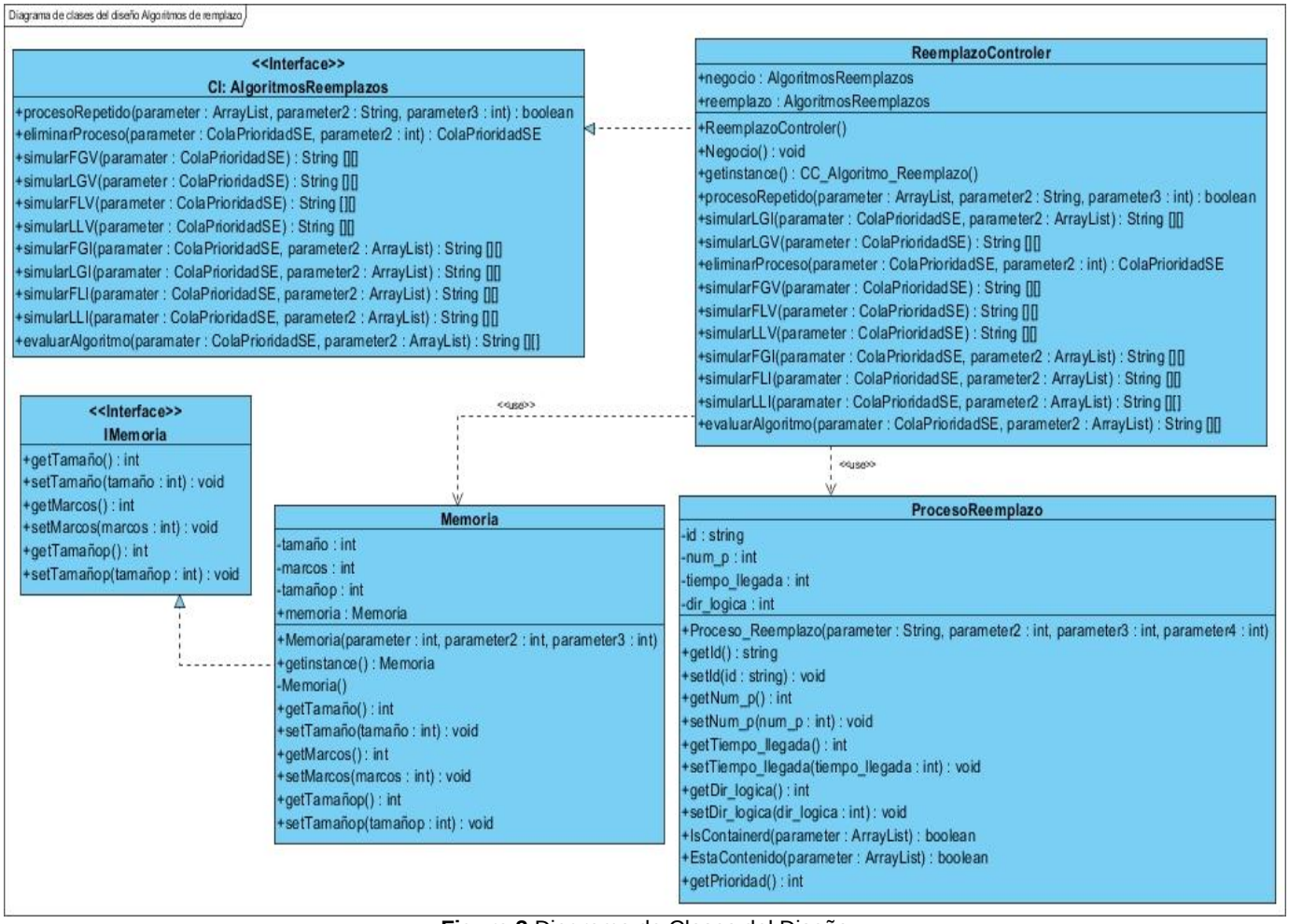


Figura 3 Diagrama de Clases del Diseño

2.6.2 Diagrama de Interacción

Los diagramas de interacción son diagramas que describen cómo grupos de objetos colaboran para conseguir algún fin. Estos diagramas muestran objetos, así como los mensajes que se pasan entre ellos dentro del caso de uso. Un diagrama de interacción capta el comportamiento de un solo caso de uso.

Se expresan de dos formas:

- Diagramas de colaboración: Muestran las relaciones entre los objetos y los mensajes que intercambian.
- Diagramas de secuencia: muestran las interacciones de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada método de la clase. Contienen detalles de

implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos.

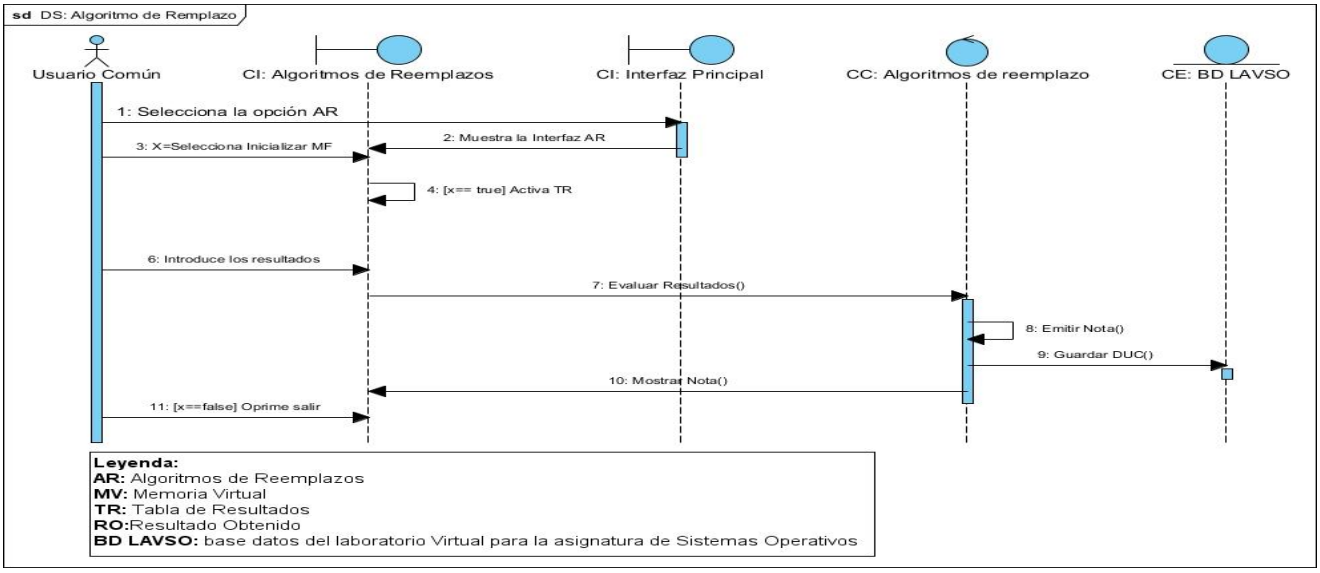


Figura 4 Diagrama de secuencia del Caso de Uso Evaluar Algoritmos de Reemplazos

Este diagrama de secuencia muestra cómo funciona el proceso Evaluar Algoritmo de Reemplazo, donde primeramente se conforma la memoria virtual, luego el usuario introduce los resultados, el sistema evalúa los resultados y emite una nota.

Para más información de los diagramas de secuencia existentes del Evaluador del tema de Gestión de Memoria se recomienda ver Anexos.

2.7 Arquitectura del Software:

La arquitectura de software le da forma al software para que soporte todos los requisitos. Establece los fundamentos para que analistas, diseñadores, programadores, etc. trabajen en una línea común que permita alcanzar los objetivos del sistema, cubriendo todas las necesidades.

Se necesita una arquitectura robusta, que guíe el proceso de desarrollo, y que defina de manera abstracta, los componentes que llevan a cabo alguna tarea, sus interfaces y la comunicación entre ellos.

Para la construcción del Evaluador de Memoria se seleccionó el estilo arquitectónico tres capas. Esta se ha vuelto muy común a la hora de construir software debido a las facilidades que brinda, pues permite la reutilización y la independencia entre las capas, se pueden realizar cambios en capas sin tener que modificar las otras, facilita la estandarización y la utilización de los recursos. Las capas que componen el sistema se listan a continuación:

2.7.1 Vista lógica de la arquitectura:

- Esta vista representa los elementos de diseño más importante para la arquitectura del sistema.
- Es una abstracción del modelo de diseño e identificación a gran escala del diseño de paquetes, subsistemas y clases.

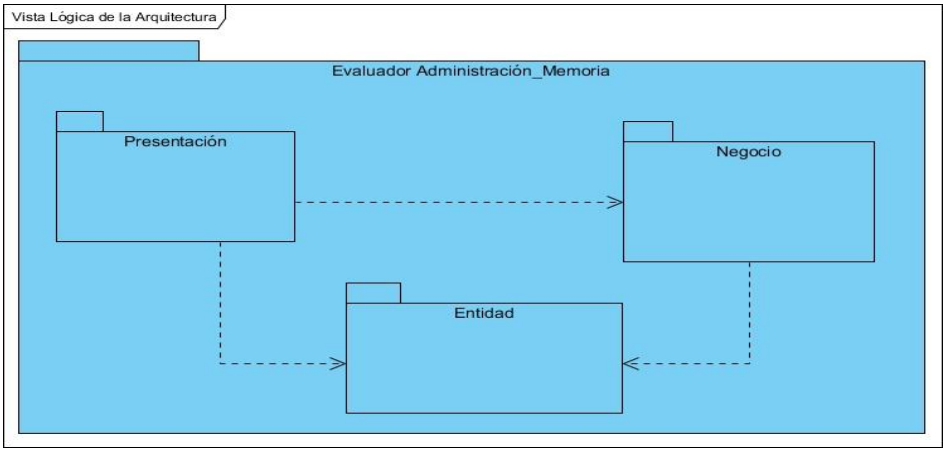


Figura 5 Vista lógica de la Arquitectura

En la vista lógica se puede observar un paquete general llamado Evaluador Administración_Memoria el mismo está compuesto por tres capas principales, (Presentación, Negocio y Entidad). Lográndose con esto un menor acoplamiento entre las clases y respondiendo así al patrón Bajo Acoplamiento. Además a la hora de darle mantenimiento al código fuente es más fácil porque un cambio en una clase no implica cambio en todas las demás clases de la aplicación.

- En la capa Presentación se encuentran todas las clases necesarias para que el usuario pueda ver la aplicación. Es aquí donde se validan los datos para que no se encuentren errores, se muestra y captura la información del usuario. Esta capa solo se comunica con la capa Negocio.

- En la capa Negocio se pueden encontrar las clases controladoras que organizan todo el trabajo de los datos de la aplicación y las de estructuras de datos. Es en este nivel donde se reciben todas las peticiones del usuario y se le dan respuestas. También esta capa recibe las solicitudes de la capa Presentación y se comunica con la capa Entidad para solicitar datos.
- En la capa Entidad se encuentran todas las clases que manejan los datos durante el funcionamiento de la aplicación. Además este nivel brinda las funcionalidades comunes a los demás niveles. Esta capa no se comunica con ninguna otra y recibe peticiones de los demás niveles.

2.8 Patrones de diseño

Durante el diseño del sistema se proponen utilizar patrones de diseño según los estudiados entre los más evidentes en el diseño propuesto se encuentran los patrones: Bajo Acoplamiento, Alta Cohesión y Singleton.

- Bajo Acoplamiento: El acoplamiento mide qué tan fuerte está una clase conectada con otras (es decir, cuántas clases conoce y necesita). Una clase con bajo acoplamiento no depende de "muchas otras" clases, lo que facilita que a la hora de darle mantenimiento al código fuente es más fácil ya que un cambio en una clase no implica cambio en todas las demás clases de la aplicación.

  Evidenciándose en la arquitectura por capas diseñada que permite agrupar las clases por tipo, dependiendo cada grupo solo de lo que necesita indispensablemente de otro.

- Alta cohesión: Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Cada elemento de nuestro diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable, una clase con baja cohesión hace muchas cosas no relacionadas o hace demasiado trabajo.

  Quedando reflejado en la clase controladora ReemplazoControler que gestiona un conjunto de entidades que realizan las funcionalidades propias por si mismas sin depender de nadie.

- Singleton: El patrón de diseño Singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Este patrón se implementa creando en nuestra clase un método que crea una instancia del objeto sólo si todavía no existe alguna.

  Poniéndose de manifiesto en la clase ReemplazoControler declarándola de tipo estática y con un método getInstance que devuelve una instancia única para cada ejecución del programa.

2.9 Conclusiones Parciales

En este capítulo se llegó a las siguientes conclusiones:

- La utilización de técnicas para la identificación de requisitos, permitió obtener resultados satisfactorios en la identificación de las funcionalidades con las que deberá contar el sistema.
- Se logró un refinamiento de los requisitos capturados, se identificaron los actores encargados de interactuar con el sistema, y se generó el diagrama de casos de uso del sistema y su especificación.
- Los artefactos creados en Visual Paradigm y UML como lenguaje de modelado, propició un mayor entendimiento entre los involucrados.
- La aplicación de patrones, tanto de casos de uso como de diseño, posibilitó obtener artefactos aceptables y evitó que se cometieran errores tradicionales.
- La construcción del modelo del diseño permitió obtener los elementos que deberán ser implementados.

Capítulo 3: Implementación y Validación de los artefactos generados

3.1 Introducción

El flujo de trabajo de Implementación tiene como objetivo definir la organización del código teniendo en cuenta los subsistemas de implementación, la implementación de los elementos de diseño en términos de ficheros fuentes, binarios y ejecutables, para poder integrar los diferentes componentes de desarrolladores o equipos y generar un ejecutable entregable o producto final. En este capítulo se presenta el artefacto diagrama de componentes, el cual muestra las dependencias entre estos. Además de la etapa de prueba, que se le realiza al software con el objetivo de asegurar completitud, correctitud, calidad, entre otros factores de gran importancia.

3.2 Implementación

La implementación se inicia a partir de los resultados obtenidos en el diseño y se implementa el sistema en términos de componentes. Estos conforman lo que se conoce como un modelo de implementación al describir los que se van a construir, su organización y dependencia entre nodos físicos en los que funcionará la aplicación. El resultado final de este flujo de trabajo es un sistema ejecutable.(Martinez, y otros)

3.2.1 Estándares de codificación de JAVA

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. Si bien los programadores deben implementar un estándar de forma prudente, este debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez.

Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad, es de gran importancia para la calidad del software y para obtener un buen rendimiento.

La adopción de un estándar de codificación sólo es viable si se sigue desde el principio hasta el final del proyecto de software. No es práctico, ni prudente, imponer un estándar de codificación una vez iniciado el trabajo.

La utilización de un estándar de codificación trae consigo ventajas, tales como:

- Asegurar la comprensión de cada una de las líneas de código.
- Facilitar el mantenimiento posterior a la creación del código.
- Facilitar la tarea de los programadores en el desarrollo del software.

Por las razones antes expuestas se decide poner en práctica el uso de estándares de codificación para la implementación del evaluador del tema de Gestión de memoria para el LAVSO.
Para la definición de este estándar es primordial mencionar dos notaciones muy importantes en el momento de codificar las cuales se mencionan a continuación.

Pascal Casing*:* la letra inicial del identificador y debe ser con mayúscula, al igual que en el caso de que sea un nombre compuesto por dos o más palabra.
Ejemplo: ReemplazoControler. Este nombre está compuesto por dos palabras (Reemplazo y Controler), la primera al igual que la segunda comienzan con mayúscula.

CamelCasing*:*Es parecido al Pascal Casing con la excepción que la letra inicial del identificador no debe estar en mayúscula.
Ejemplo: procesoRepetido. Este nombre está compuesto por dos palabras (proceso y repetido), la primera todo en minúsculas y la segunda iniciando con letra mayúscula.

Nomenclatura de las clases Los nombres de las clases comienzan con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación Pascal Casing.

Nomenclatura de las funciones El nombre a emplear para las funciones se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, y con sólo leerlo se reconoce el propósito de la misma.

Nomenclatura de las variables

El nombre a emplear para las variables se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, y comenzando con un prefijo según el tipo de datos.

3.2.2 Diagrama de Componentes

A través del diagrama de componentes el analista ilustra los elementos que serán usados para construir el sistema. UML define cinco estereotipos estándar que se le aplican a los mismos, como son: ejecutable, bibliotecas, tabla, archivo y documento. Los distintos componentes se pueden agrupar en paquetes según un criterio lógico y con vista a simplificar su implementación.

En el siguiente diagrama se muestra la relación existente entre el paquete de presentación que el mismo depende del paquete entidad y del paquete negocio y finalmente este último del paquete entidad, cada paquete contiene dentro sus respectivos paquetes y en el diagrama se muestra también la relación que existe entre cada uno de ellos.

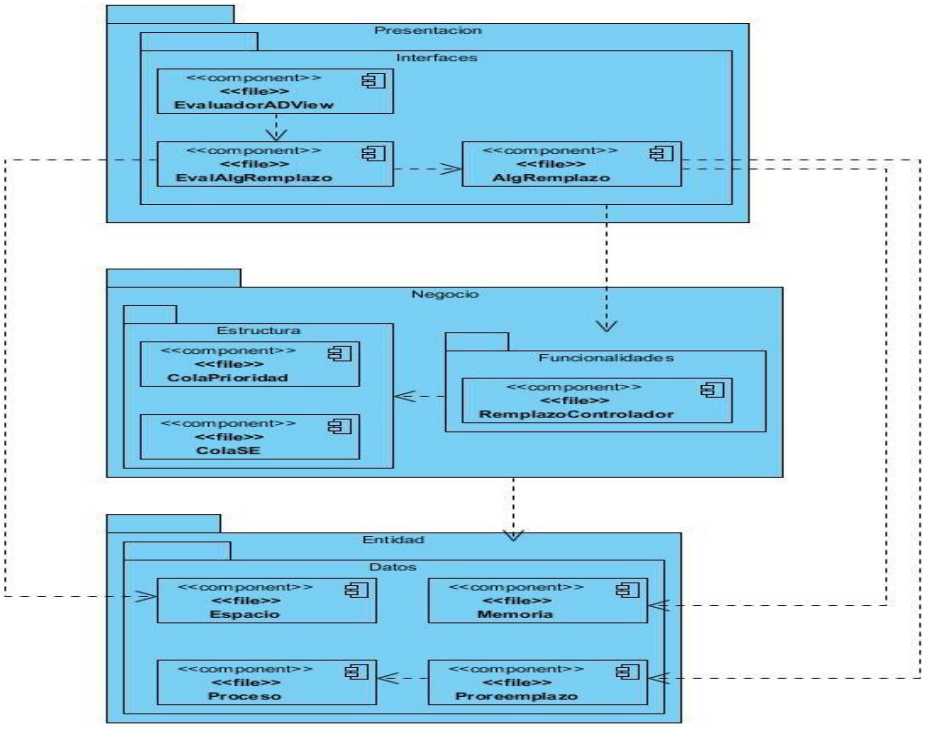


Figura 6 Diagrama de Componentes

3.3 Resultados Obtenidos

Anteriormente para la asignatura de Sistemas Operativos no existía ningún medio específicamente para el tema de Gestión de Memoria que le posibilitara al estudiante la comprobación de los conocimientos adquiridos en clases o en el EVA, por lo que se decidió desarrollar una herramienta para evaluar a los estudiantes de forma práctica haciendo uso de las TICs y cumpliendo con el nuevo modelo de formación que se está aplicando en la UCI.

Como resultado se tiene una aplicación capaz de brindarle al estudiante una forma de evaluar el contenido del tema en cuestión, posibilitándole al mismo adquirir habilidades y consolidar sus conocimientos de una forma práctica y más atractiva.

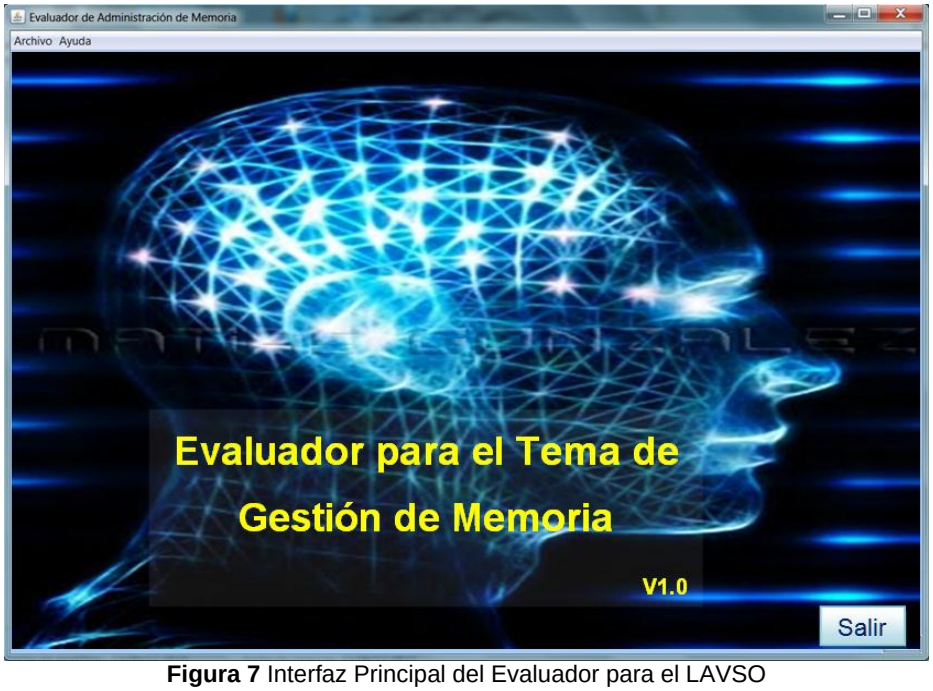


Figura 7 Interfaz Principal del Evaluador para el LAVSO

3.4 Validación de los artefactos obtenidos

Para la validación de los artefactos del sistema se aplicaron métricas para la Calidad de la Especificación de los Requisitos de Software, métricas para medir la calidad de la funcionalidad del diagrama de casos de uso del sistema y para la validación de los artefactos del modelo del diseño se aplican métricas a nivel del tamaño de clases.

La aplicación de estas métricas consistió en la realización de revisiones por parte de los miembros del equipo de trabajo, el cual está compuesto por 3 personas que fueron escogidas por estar directamente involucrados con el proceso de desarrollo del software.

Nombre y Apellidos de los revisores
Carlos Yasmiany Hidalgo García
Martín Villalón Cruzata.
Julio Antonio Villaverde

Tabla 5 Nombre del equipo de revisores del software

3.4.1 Métricas para la Calidad de la Especificación de los Requisitos de Software

Para aplicar esta métrica se realizaron dos revisiones a los requisitos especificados, por parte de los miembros del equipo de trabajo para lograr obtener requisitos fuertes y sin incoherencias, inicialmente fue necesario calcular el número total de requisitos, para así poder aplicar la métrica que hace uso de este.

Rf: Número de requisitos funcionales.

Rnf: Número de requisitos no funcionales.

Rt: Total de requisitos.

Rt = Rf + Rnf
Rt = 18 + 12
Rt = 30

Especificidad

Para determinar la especificidad (ausencia de ambigüedad) de los requisitos se empleó la métrica basada en la consistencia de la interpretación de los revisores para cada requisito; ***Q1= Rii/ Rt*.** Cuanto más cerca de 1 esté el valor de Q1(grado de especificidad de los requisitos), mayor será la consistencia de la

interpretación de los revisores para cada requisito y menor será la ambigüedad de la especificación de los requisitos. (22)

Primera Revisión:

Rii: Número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas y **Rt** el total de requisitos calculado con anterioridad, sustituyendo las variables se obtiene:

Q1= Rii/ Rt
Q1= 28/30
Q1= 0.93

Segunda Revisión:

En esta revisión se solucionaron los problemas de ambigüedad por lo que después de los cambios hechos se cuenta con iguales interpretaciones por parte de los revisores para los 30 requisitos.

Q1= 30/30
Q1=

Gráfica 1 Calidad Especificación de Requisitos

Luego de analizar las revisiones anteriores se concluye que los requisitos presentan un bajo nivel de ambigüedad, pues ya en la segunda revisión se solucionaron todos los problemas presentados en la primera. Estos resultados son bastantes satisfactorios ya que ayuda que no se atrase el desarrollo del software por error en los requisitos.

3.4.2 Métricas para la evaluación de la calidad en los diagramas de casos de usos

Las métricas para la calidad de la funcionalidad del diagrama de casos de uso de sistema definen cuatro atributos:(22)

- Completitud: permite determinar el grado en que se ha incluido de forma clara y concisa todos los elementos necesarios para la descripción del aspecto.
- Consistencia: permite definir el grado en que los elementos del artefacto representan en forma única y no contradictoria un aspecto del problema.
- Correctitud: permite establecer el grado de adecuación del artefacto para satisfacer los requerimientos establecidos.
- Complejidad: permite medir el grado de claridad y rehúso del artefacto.

Estos atributos presentan un significado determinado de acuerdo con el tipo de artefacto y al nivel de abstracción que éste describe. Cada atributo se evalúa en términos de un conjunto de factores, los cuales tendrán asociados una métrica.

A continuación se muestra la tabla de las métricas correspondientes a cada uno de los atributos especificados anteriormente:(22)

No.	Atributo.	Métricas.
1.	Completitud	Número de casos de uso que no tiene descripción resumida.
2.		Número de casos de uso que tienen requerimientos omitidos.
3.		Número de casos de uso que no poseen una descripción extendida.
4.		Número de casos de uso que tienen acciones del flujo de eventos no redactados en función del responsable.
5.		Número de casos de uso que no describen condiciones de excepción relevantes.
6.		Número de casos de uso que no han sido clasificados.
7.	Consistencia	Número de casos de uso que tienen un nombre incorrecto.
8.		Número de casos de uso que no representan una interacción observable por un actor.

9.		Número de casos de uso que tienen acciones del flujo de eventos asignados a un responsable que no le corresponde.
10.		Número de casos de uso no aceptados.
11.		Número de casos de uso complejos que no tienen separación del flujo básico y de flujos alternos.
12.		Número de casos de uso que no tienen un usuario responsable.
13.	Correctitud	Número de casos de uso en que los requerimientos representados no son comprensibles por el usuario.
14.		Número de casos de uso que deben ser modificados para adecuarlos a la funcionalidad del sistema.
15.		Número de casos de uso que deben ser modificados para mejorar el proceso actual.
16.	Complejidad	Número de elementos del diagrama que requieren reubicación.

Tabla 6 Métricas para la calidad de la funcionalidad del diagrama de casos de uso de sistema

Resultados por métricas

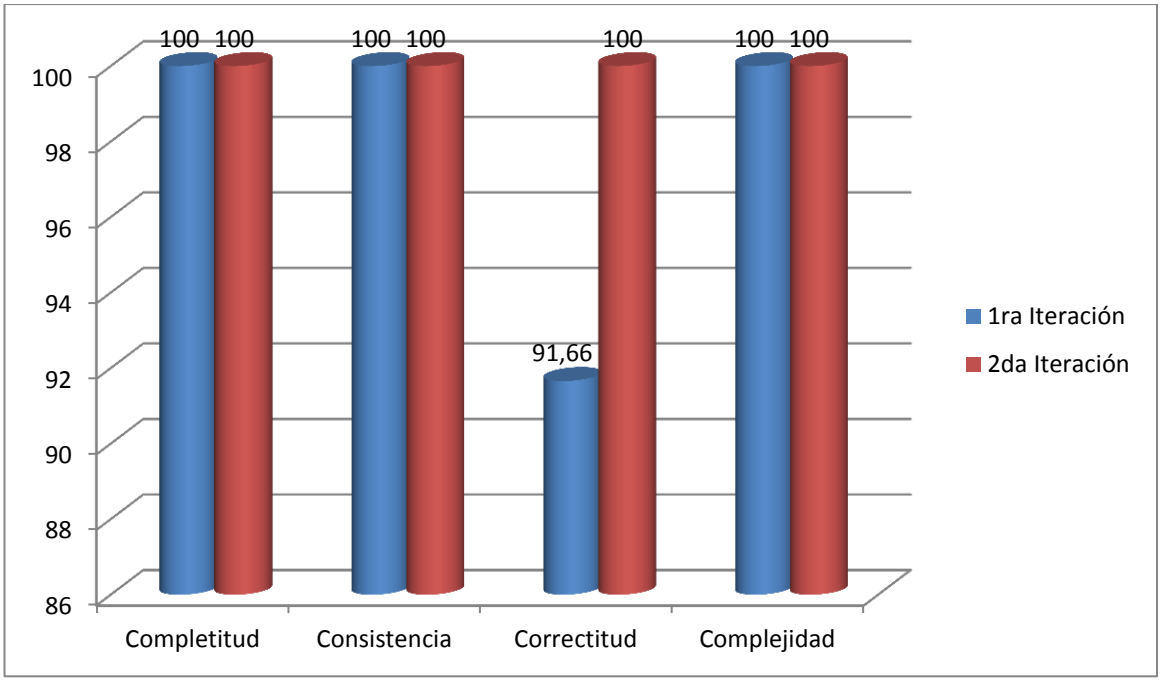
Métrica	No de casos de uso analizados	Resultado	Porcentaje de error
1	4	Todos poseen una descripción resumida.	0%
2	4	Ninguno presenta requerimientos omitidos, al menos están relacionados con 1 requerimiento.	0%
3	4	No se detectó ningún caso de uso que no presentara su descripción extendida.	0%
4	4	Todos están redactados en función del responsable que le corresponde a cada uno.	0%
5	4	No se detectó ningún caso de uso que no describiera condiciones de excepción relevantes.	0%
6	4	Todos han sido clasificados según los diferentes tipos de prioridad.	0%

7	4	Ninguno presentó problemas en cuanto al nombre dado, ya que todos los nombres representan una expresión verbal en infinitivo describiendo alguna funcionalidad relevante y significativa.	0%
8	4	Todos representan correctamente la interacción con un actor.	0%
9	4	Ninguno presentó acciones del flujo de eventos que estuvieran asignados a un responsable que no le corresponde.	0%
10	4	Todos fueron aceptados.	0%
11	4	Todos poseen sus descripciones del flujo básico y de flujos alternos separadas.	0%
12	4	Todos poseen un usuario responsable.	0%
13	4	Todos los casos de uso en que los requerimientos fueron representados son comprensibles por el usuario.	0%
14	4	Se detectó un caso de uso que debe ser modificado para adecuarlo a la funcionalidad del sistema.	8.33%
15	4	Ninguno tuvo que ser modificados para mejorar el proceso actual.	0%
16	4	Ninguno de los elementos que conforman el diagrama de casos de uso requiere reubicación.	0%

Tabla 7 Resultados de las métricas para la calidad de la funcionalidad del diagrama de casos de uso de sistema

Como resultado de aplicar esta métrica se pudo observar la calidad y funcionalidad del diagrama de casos de uso, obteniéndose un 100% tanto en completitud, consistencia y complejidad y no siendo así en Correctitud donde se obtuvo un 91.66% debido a que existe un caso de uso que tuvo que ser redefinido para que cumpla con necesidades específicas del cliente.

Se realizó una segunda revisión para arreglar los errores encontrados donde se obtuvo un 100% en la calidad de la funcionalidad del diagrama de casos de usos del sistema.



Gráfica 2 Resultados de la evaluación de la calidad en los diagramas de casos de usos del sistema

3.4.3 Tamaño de clase (TC)

Para medir el tamaño de las clases (TC), se tienen en cuenta los aspectos siguientes:

- Total de operaciones (tanto operaciones heredadas como privadas de la instancia) que están encapsuladas dentro de la clase.
- Cantidad de atributos (tanto atributos heredados como atributos privados de la Instancia) que están encapsulados en la clase.
- Promedio general de los dos anteriores para el sistema completo. (11)

Clases	No. Atributos	No. Operaciones
CI_Algoritmo_Reemplazo	0	19
Algoritmo_Reemplazo	2	21
ColaPrioridadSE	0	2
ColaSE	3	7
ExceptionColaVacia	0	1
ICola	0	5
NodoSE	2	4

Errores	2	4
Error_tecnico	2	2
Error_procedimiento	2	2
Nota	4	4
Proceso_reemplazo	4	12
Memoria	4	9
Iprioriable	0	1
IMemoria	0	6
Tecnicas_Asignacion	0	12
CI_ Tecnicas_Asignacion	0	8
Proceso	2	5
IProceso	0	2
Espacio	4	10
IEspacio	0	7
My Panel	3	4
Jpanel	0	2

Tabla 8 Métrica tamaño de clases aplicada a las clases del diseño

Umbral	Tamaño	Cantidad de Clases
<=20	Pequeño	22
>20 y <=30	Mediano	1
>30	Grande	0

Tabla 9 Valores de los umbrales

Para evaluar las métricas son necesarios los valores de los umbrales. Las clases se clasifican en tres grupos, según su tamaño, los que se presentan en la tabla #8 junto con los umbrales seleccionados para su clasificación. Donde se graficó el resultado obtenido:

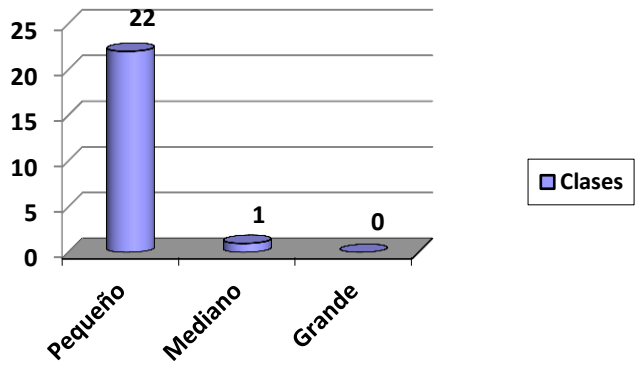


Gráfico 3 Número de clases por categorías

Con esta métrica se pudo comprobar que atendiendo a la cantidad de operaciones, el 96.5 % de las clases diseñadas están consideradas como pequeñas, lo que facilitará el proceso de construcción del sistema.

3.4.4 Pruebas de Software

Se aplicaron las pruebas de caja negra que la mismas examinan algunos aspectos del modelo, fundamentalmente del sistema, sin tener mucho en cuenta la estructura interna del software. Se centran principalmente en los requisitos funcionales del software. Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa.

Escenario	Descripción	Varia-bles	Respuesta del sistema	Flujo central
EC 1.1 Evaluar Algoritmo de Reemplazos	El usuario introduce los resultados al evaluador y el sistema compara esos resultados con los obtenidos en la simulación del ejercicio	Jtable Tabla	El sistema evalúa los resultados introducidos por el estudiante sin mostrar ningún error y emite nota máxima.	5. El sistema muestra una tabla donde el usuario introducirá los resultados del algoritmo y activa el botón evaluar. 6. El UC introduce los resultados y oprime evaluar.
EC 1.2 Valores erróneos	El usuario introduce resultados incorrectos	Jtable	El sistema señala en rojo los	

	al evaluador	Tabla 1	errores y muestra un mensaje con la cantidad de errores por tipo y la nota obtenida.	7. El sistema evalúa los resultados introducidos por el UC. 8. El sistema muestra un mensaje con la nota y la cantidad de errores cometidos. 9. El sistema envía la nota del UC al módulo de ejercicios. 10. Terminando así el Caso de Uso.

Tabla 10 Diseño del caso de prueba Evaluar Algoritmos de Reemplazos

Descripción de variables

Tabla =

Marcos	t=0	t=1
0	P1,0,t=0	P1,0,t=0
1		P2,0,t=1
F y R	F	F

Figura 8 Juego de datos para escenario #1 del Caso de Prueba" Evaluar Algoritmos de Reemplazos"

Tabla1=

Marcos	t=0	t=1
0	P1,0,t=0	P1,0,t=0
1		P2,0,t=1
F y R	R	F

Figura 98 Juego de datos para escenario #2 del Caso de Prueba" Evaluar Algoritmos de Reemplazos"

Las pruebas realizadas al componente obtuvieron resultados satisfactorios, concluyendo que es un producto de calidad y apto para el uso. Los demás casos de pruebas realizados se encuentran en el artefacto diseño de casos de pruebas.

3.5 Conclusiones parciales

Se considera que los objetivos trazados para el presente capítulo han sido cumplidos y como resultado se obtuvo el Evaluador del tema de Gestión de Memoria para el LAVSO.

- Con la aplicación de métricas a los requisitos y a los casos de usos se logró que los requisitos identificados estén con un 100% de especificación y los diagramas de casos de uso del sistema presenten una buena calidad y funcionalidad.
- Al aplicar las métricas para medir la calidad del diseño, se pudo constatar que el diseño propuesto tiene poca complejidad. Además las clases que lo componen por lo general son reusables y fáciles de comprobar, de manera que se puede concluir que el sistema posee un diseño adecuado.
- Las pruebas efectuadas arrojaron resultados satisfactorios, comprobándose así el correcto funcionamiento del código. Además al realizar las pruebas de aceptación se dio respuesta oportuna a las incidencias detectadas, quedando de esta forma el sistema listo para el despliegue.

CONCLUSIONES

- Con el estudio y comprensión de los LV y los principales Evaluadores existentes en el mundo, se determinó que ningunas de las soluciones existentes sirven de aporte al Evaluador de Memoria.
- El análisis y diseño del problema planteado permitió crear las bases para una solución novedosa que aportaría productividad durante el desarrollo del LAVSO.
- Mediante la implementación de la solución se obtuvo un producto interactivo que cumple con los requisitos establecidos por el cliente y con la aplicación de las métricas a los artefactos y las pruebas realizadas al sistema se validaron los resultados obteniendo un software listo para usarse.
- Se cumplió el objetivo general trazado para este Trabajo de Diploma: Desarrollar un Evaluador para el tema de Gestión de Memoria que posibilite al estudiante y al profesor la evaluación y comprobación de los conocimientos adquiridos para el LAVSO

Recomendaciones

Tomando en cuenta los resultados obtenidos:

- El evaluador de memoria solo permite la evaluación de 2 Algoritmos de Reemplazos por lo que se propone el aumento del cúmulo de funcionalidades que brinde la posibilidad de evaluar otros algoritmos para el LAVSO.
- Integrar el Evaluador de memoria al Módulo de ejercicios perteneciente al LAVSO.

BIBLIOGRAFÍA REFERENCIADA

Herías, Francisco Andrés Candelas. 2003. Propuesta de Portal de la Red de Laboratorios Virtuales y Remotos de CEA. [Online] noviembre 27, 2003.

1998.*IEEE, IEEE Guide for developing System Requirements Specifications.* s.l. : [Online] 1998., 1998.

Jacobson, Ivar, Booch, Grady and Rumbaugh, James. 2000.*El proceso unificado de desarrollo de software.* Madrid : Pearson Educación, 2000.

Martinez, Alejandro and Martinez, Raul.*Guia de Rational Unified Process.* s.l. : http://www.dsi.uclm.es/asignaturas/42551/trabajosAnteriores/Trabajo-Guia%20RUP.pdf.

Pressman, Rogen S. 2005.*Ingeniería de Software"Un enfoque Práctico".* s.l. : 5ta Edición, 2005.

Robertson, Suzanne y Robertson Jame. 2006.*Masering the Requirements Process.* s.l. : Second Editon, 2006.

Rodríguez Urrutia, Misael and Figueredo Jiménez, Lianni Yadira. 2010. Simulador para la asignatura Sistemas Operativos. Ciudad Habana : s.n., 2010.

BIBLIOGRAFÍA

1. **Colectivo de asignatura de la Facultad.***Informe semestral de la asignatura de Sistemas Operativos .* Ciudad Habana : s.n., 2012.

2. **Herreros, L. Rosado y J.R.** Nuevas aportaciones didácticas de los laboratorios virtuales y remotos en la enseñanza de la Física. . [En línea] 14 de abril de 2005. http://www.formatex.org/micte2005/286.pdf.

3. **Vary, James P.** Informe de la reunión de expertos. . [En línea] 10 de febrero de 2000. http://unesdoc.unesco.org/images/0011/001191/119102s.pdf..

4. **Estrada, Monge, Nájera, Julian y Hugo, Méndez,.Victor.** Ventajas y desventajas de usar laboratorios virtuales en educación a distancia: la opinión del estudiantado en un proyecto de seis años de duración. [En línea] 13 de Febrero de 2011. http://www.latindex.ucr.ac.cr/edu31-1/edu-31-1-05.pdf..

5. Definición .de. [En línea] http://definicion.de/evaluacion/.

6. **William, Stallings.***Sistemas Operativos Modernos I.*

7. **Santos Sanabria, Cesar y Colomina Curbelo, Luisdey .***[Trabajo de Diploma] Componente Evaluador de Expresiones Matemáticas.* Ciudad Habana : s.n., 2010.

8. Evaluador de Derivadas. [En línea] 10 de marzo de 2005.

9. **Rostro Hernández, Perla Esperanza , Herrera Cabral, Joel y Ramírez Barajas, Alejandro.** Software Evaluador de Proyectos de Inversión. [En línea] 1 de Junio de 2010.

10. Programa de Gobierno Corporativo. [En línea] http://gc.caf.com/interna.asp?pg=17.

11. **Pressman, Rogen S.***Ingeniería de Software"Un enfoque Práctico". s.l. : 5ta Edición,.* 2005.

12. *IEEE Guide for developing System Requirements Specifications.*

13. **Durán, Toro, Amador y Bernárdez, Jiménez, Beatriz.***Metodología para la Elicitación de Requisitos de Sistemas Software.* 2000.

14. **Sommerville, Ian.***Ingeniería de software.* 2005.

15. **Fernandez, Romero, Dailin.***Título: Desarrollo de los requisitos de software de los procesos Solicitudes de Copias, Legalizaciones de Firmas y Notas Marginales que se llevan a cabo en los Registros Principales de la República Bolivariana de Venezuela.* Habana : Trabajo de Diploma, 2011.

16. **Molpeceres, Alberto.** Procesos de desarrollo: RUP, XP y FDD. [En línea] 2002. http://www.willydev.net/descargas/articulos/general/cualxpfddrup.PDF..

17. **Larman, Graig.***UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* 2004.

18. **Jacobson, Ivar, Booch, G y Rumbaugh, J.***El Proceso Unificado de Desarrollo de Software. .* Madrid, España : s.n., 2000.

19. **Schmuller, Joseph.***Aprendiendo UML en 24 horas.* 2000.

20. Boost Productivity with Innovative and Intuitive Technologies. [En línea] 2007. http://www.visual-paradigm.com/aboutus/10reasons.jsp..

21. **Wilson, Leslie B.***Comparative Programming Languages. 2da Edición.*.

22. **Overmyer, Davis S.***Identifying and measuring quality in software requirements specification.* Los Alamitos, California. : s.n..

ANEXOS

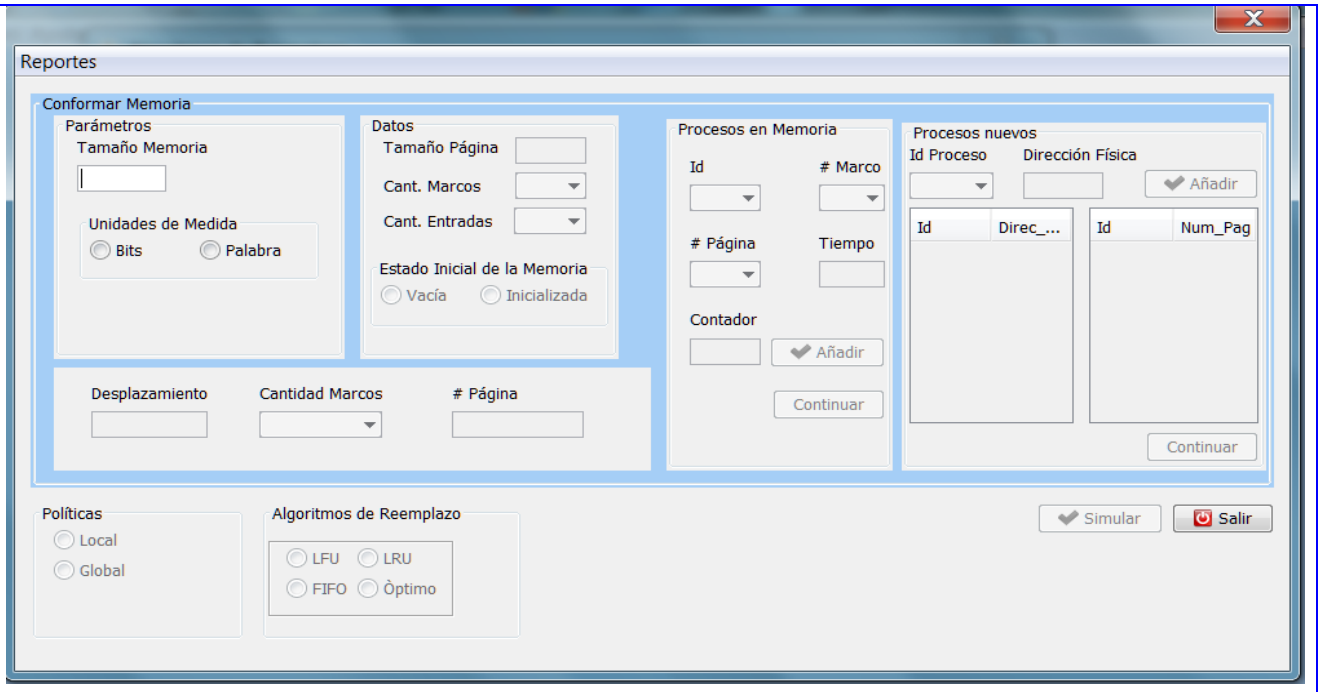
Anexo 1 Descripción de los Casos de Usos del sistema:

CU: Conformar Memoria Virtual

Caso de Uso:	Conformar Memoria Virtual
Actores:	Usuario común
Resumen:	Para conformar la memoria virtual el usuario introduce los datos del ejercicio, luego se validan los mismos y de estar correctos se procede a simular.
Precondiciones:	
Pos condición:	
Casos de usos asociados:	
Referencias	RF8, RF9, RF11, RF12, RF13, RF14,
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra la interfaz Conformar Memoria Virtual activada y con los siguientes aspectos a llenar : • Tamaño de Memoria • Unidades de Medida (bits o palabra)
2. El UC introduce datos de memoria	3. Si la unidad de medida entrada por el UC es en bit **ir al flujo alterno 1** de lo contrario si el tamaño de la memoria es en palabra el sistema activa los siguientes campos: • Tamaño de Pagina • Cantidad de Marcos • Cantidad de entrada a la tabla de páginas. • Estado de la Memoria (vacía o iniciada)

4. El UC introduce los datos	5. Si la memoria está inicializada **ir al flujo alterno 2** si está vacía se muestra las siguientes opciones activadas para añadir los procesos: • Id del proceso • Dirección lógica • Tamaño de Página
6. El UC entra los procesos del ejercicio selecciona la opción añadir y cuando termina de introducir los procesos selecciona la opción terminar.	7. El sistema activa la Opción “Seleccionar Política de Reemplazo” con los siguientes campos: • Política Local • Política Global
8. El UC selecciona la política de reemplazo.	9. El sistema muestra los Algoritmos de Reemplazo a seleccionar por el UC: • OPT (Algoritmo óptimo). • FIFO (First Come First Out). • LRU (Least Recently Used). • LFU (Least Frequently Used).
10. El UC selecciona el algoritmo de reemplazo deseado y selecciona el botón simular.	11. Si existen campos vacíos **ir al flujo alterno 3** de lo contrario el sistema determina si existen errores en los datos **ir al flujo alterno 4.**
	12. El sistema realiza la simulación en dependencia del algoritmo seleccionado y la política de reemplazo escogida.
	13. El sistema almacena los resultados de la simulación en una variable.
	14. Terminando así el Caso de Uso.
Prototipo de Interfaz	

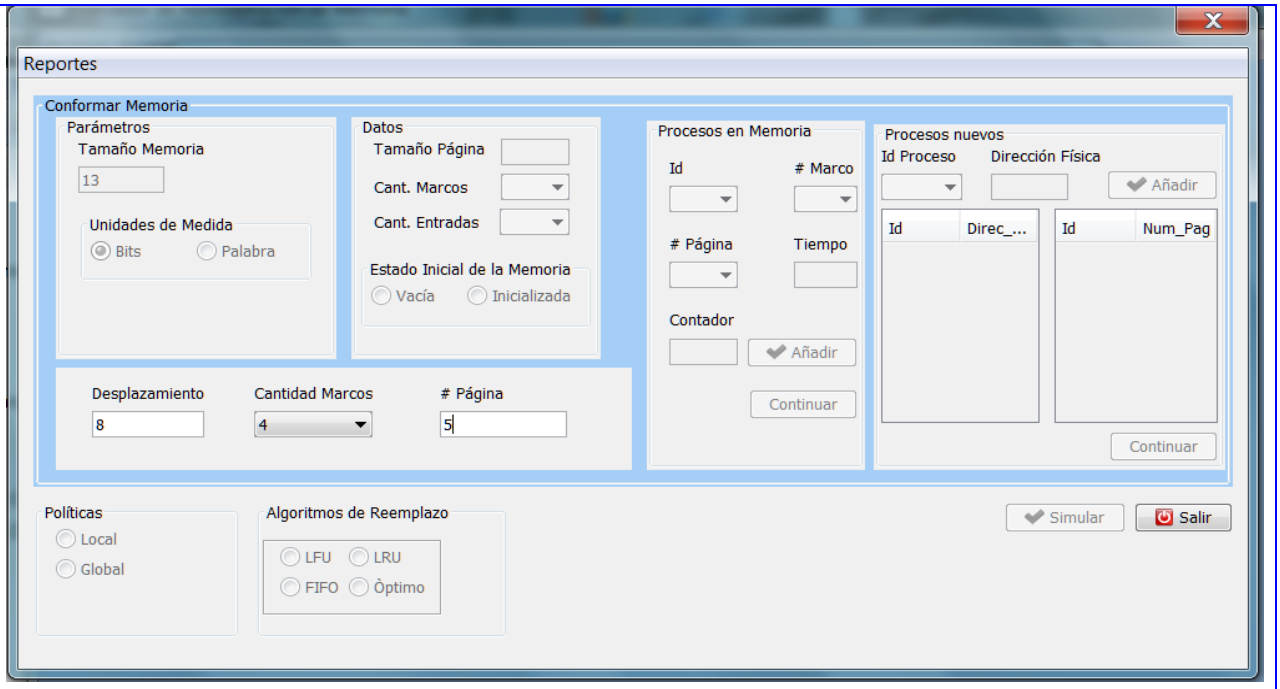


Flujos Alternos

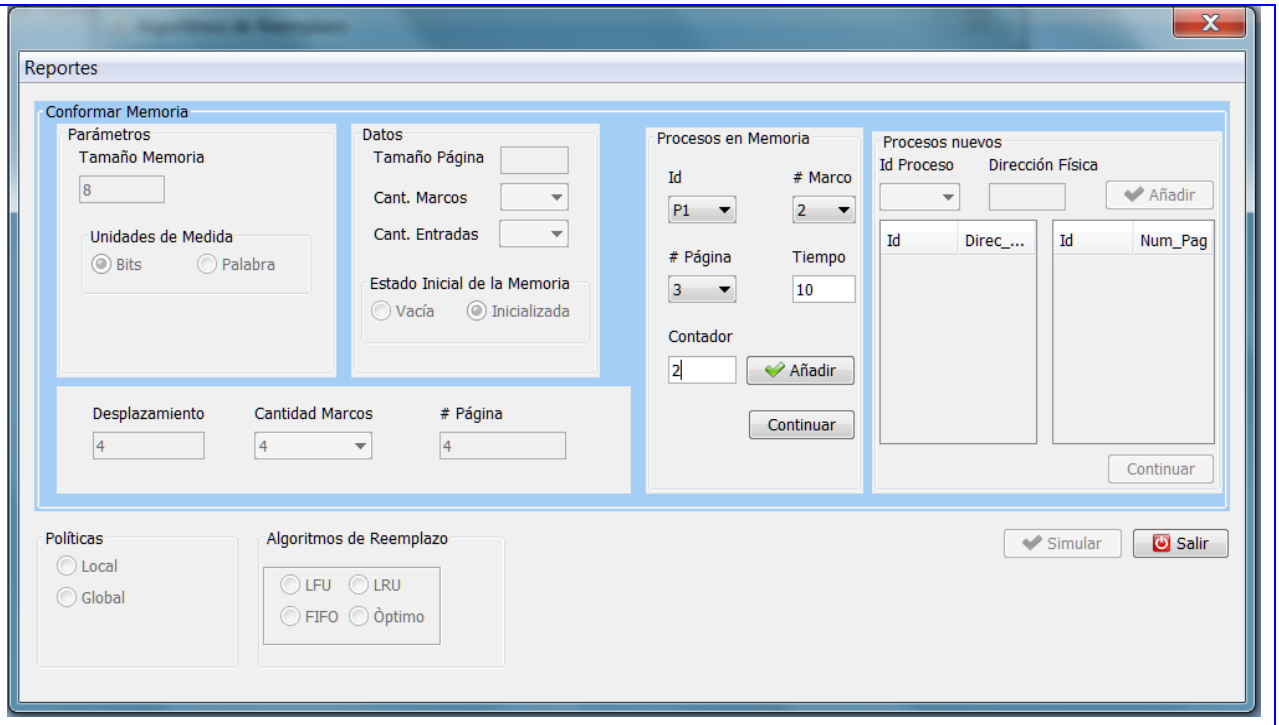
Flujo Alterno 1

Acción del Actor	Respuesta del Sistema
	1. Activa las siguientes opciones: • Desplazamiento • Cantidad de Marcos • Número de Página
2. El UC introduce los valores correspondientes	3. Ir al paso 5 del Flujo Normal de Eventos.

Prototipo de Interfaz



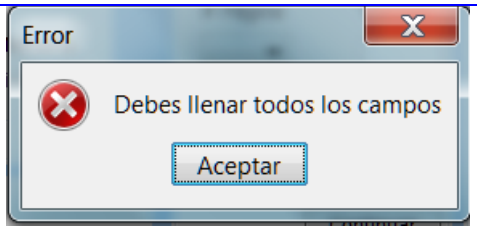
Flujo Alterno 2	
Acción del Actor	**Respuesta del Sistema**
	1. Se activan los siguientes campos para que el UC entre los procesos que están en la Memoria: • ID del proceso • No. De marco que se encuentra • No. De Página • Tiempo • Contador
	2. Regresar al paso 6 del Flujo Normal de Eventos.
Prototipo de Interfaz	



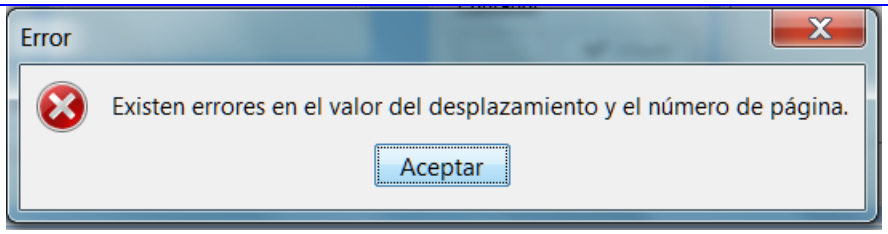
Flujo Alterno 3

Acción del Actor	Respuesta del Sistema
	1. El sistema muestra un mensaje “Debe llenar todos los campos”
2. El usuario llena los campos vacíos y selecciona el botón aceptar.	3. Ir al paso 10 del flujo normal de eventos.

Prototipo de Interfaz



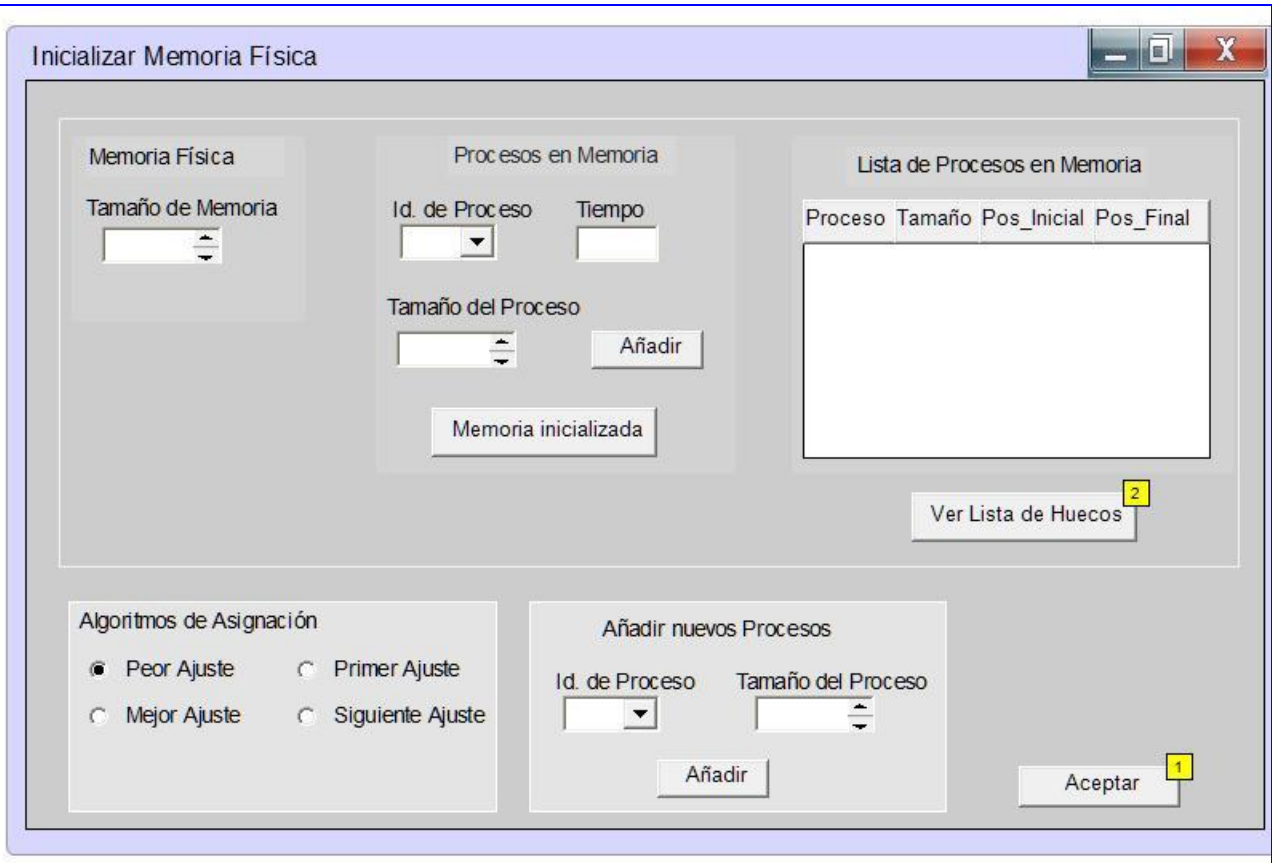
Flujo Alterno 4

Acción del Actor	Respuesta del Sistema
	1. El sistema muestra un mensaje con el error.
2. El usuario corrige los errores y oprime el botón aceptar.	3. Ir al paso 11 del flujo normal de eventos.
Prototipo de Interfaz	
	

CU: Inicializar Memoria Física

Caso de Uso:	Inicializar Memoria Física
Actores:	Usuario común
Resumen:	Para inicializar la memoria física el usuario introduce los datos del ejercicio, luego se validan los mismos y de estar correctos a añadir los procesos en memoria.
Precondiciones:	
Pos condición:	
Casos de usos asociados:	
Referencias	RF1, RF2, RF3, RF4
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra la interfaz Inicializar Memoria Física con la opción Tamaño de Memoria

	activada.
2. El UC introduce los datos y oprime el botón aceptar.	3. El sistema muestra la opción inicializar memoria con los siguientes campos activos: • ID del Proceso • Posición inicial • Tamaño del Proceso
4. El UC selecciona la opción memoria inicializada.	5. El sistema muestra una tabla con una lista de los Procesos que hay en memoria y la opción ver lista de huecos (Ir al flujo alterno 1).
	6. El sistema activa las técnicas de asignación a evaluar: • Peor Ajuste • Mejor Ajuste • Primer Ajuste • Siguiente ajuste
7. El UC selecciona la Técnica de Asignación	8. El sistema activa la opción Agregar nuevos procesos a la memoria con los siguientes acápites: • Id. Del Proceso • Tamaño del Proceso
9. EL UC agrega los procesos deseados en la memoria selecciona añadir.	10. Si existen errores en los datos ir al flujo alterno 2 de lo contrario continuar con el flujo normal de eventos.
	11. El sistema añade los procesos en memoria en dependencia de la técnica de asignación escogida.
	12. El sistema almacena los resultados en una variable.
	13. Terminando así el Caso de Uso.
Prototipo de Interfaz	



Flujos Alternos	
Flujo Alterno 1	
Acción del Actor	**Respuesta del Sistema**
1. El UC selecciona opción Lista de Hueco	2. El sistema muestra en la interfaz Lista de Huecos en memoria una la lista de huecos en la memoria.
3. Selecciona salir	4. Volver al paso 6 del Flujo Normal de Eventos
Flujo Alterno 2	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema resalta el error en rojo y le da la

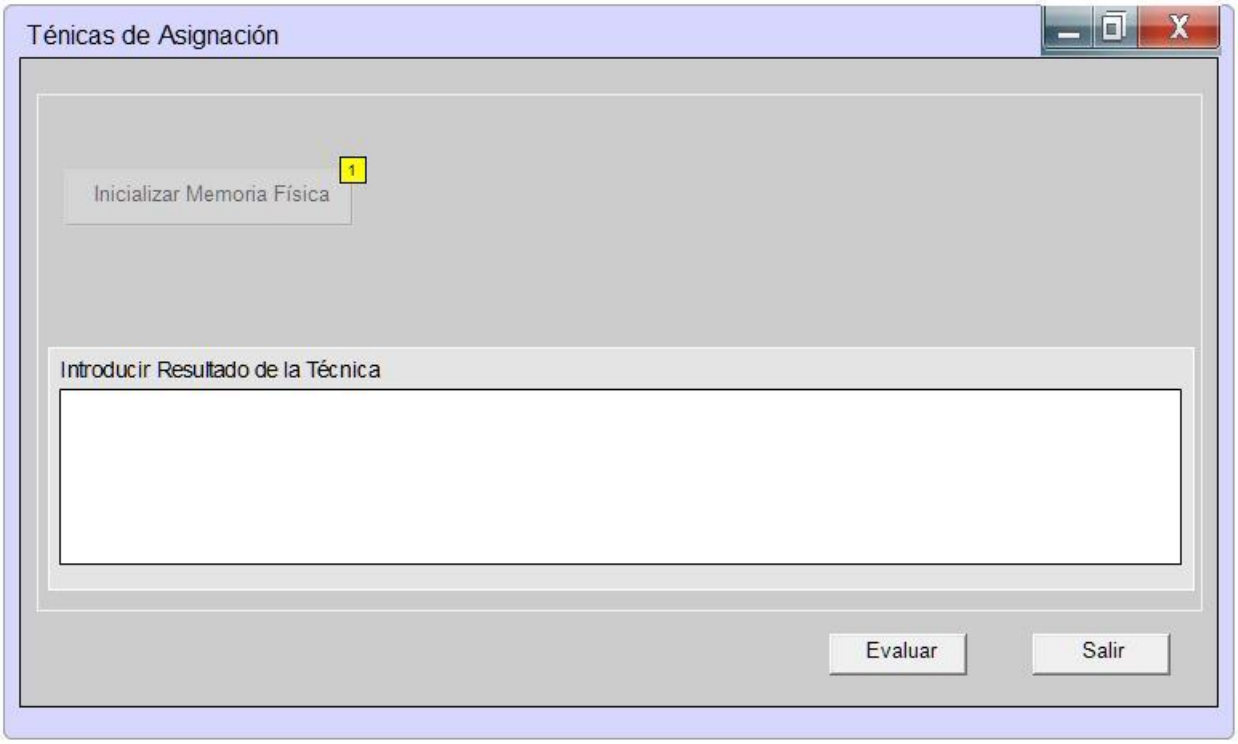
	posibilidad al usuario de corregirlo.
	2. El sistema guarda la cantidad de errores técnicos en una variable.
3. El usuario corrige los errores y oprime el botón aceptar.	4. Ir al paso 12 del flujo normal de eventos.
Prototipo de Interfaz	

CU: Evaluar Técnica de Asignación

Caso de Uso:	Evaluar Técnica de Asignación
Actores:	Usuario común
Resumen:	El caso de uso inicia cuando el usuario común selecciona la opción Técnicas de Asignación, después de inicializada la memoria física el usuario introduce los resultados de la técnica y termina cuando el sistema da una nota al usuario.
Precondiciones:	
Pos condición:	
Casos de usos asociados:	Inicializar Memoria Física (caso de uso incluido)
Referencias	RF5, RF6, RF7, RF10
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El UC selecciona la opción Evaluar Técnicas de Asignación.	2. El sistema muestra la interfaz Técnicas de Asignación con las siguientes opciones activadas: • Inicializar Memoria Física • Salir
3. Si el UC selecciona Salir ir al paso 17 del flujo	4. Invoca el CU incluido Inicializar Memoria Física.

normal de eventos, de lo contrario selecciona el botón Inicializar Memoria Física	
	5. El sistema muestra una tabla donde el usuario introducirá los resultados del algoritmo y activa el botón evaluar.
6. El UC introduce los resultados y selecciona el botón evaluar.	7. El sistema evalúa los resultados introducidos por el UC.
	8. El sistema muestra un mensaje con la nota y la cantidad de errores cometidos.
	9. El sistema envía la nota del UC al módulo de ejercicios.
	10. Terminando así el Caso de Uso.

Prototipo de Interfaz



Anexo 2 Diagrama de clases del diseño

Diagrama de clases “Evaluar Algoritmos de Reemplazos”

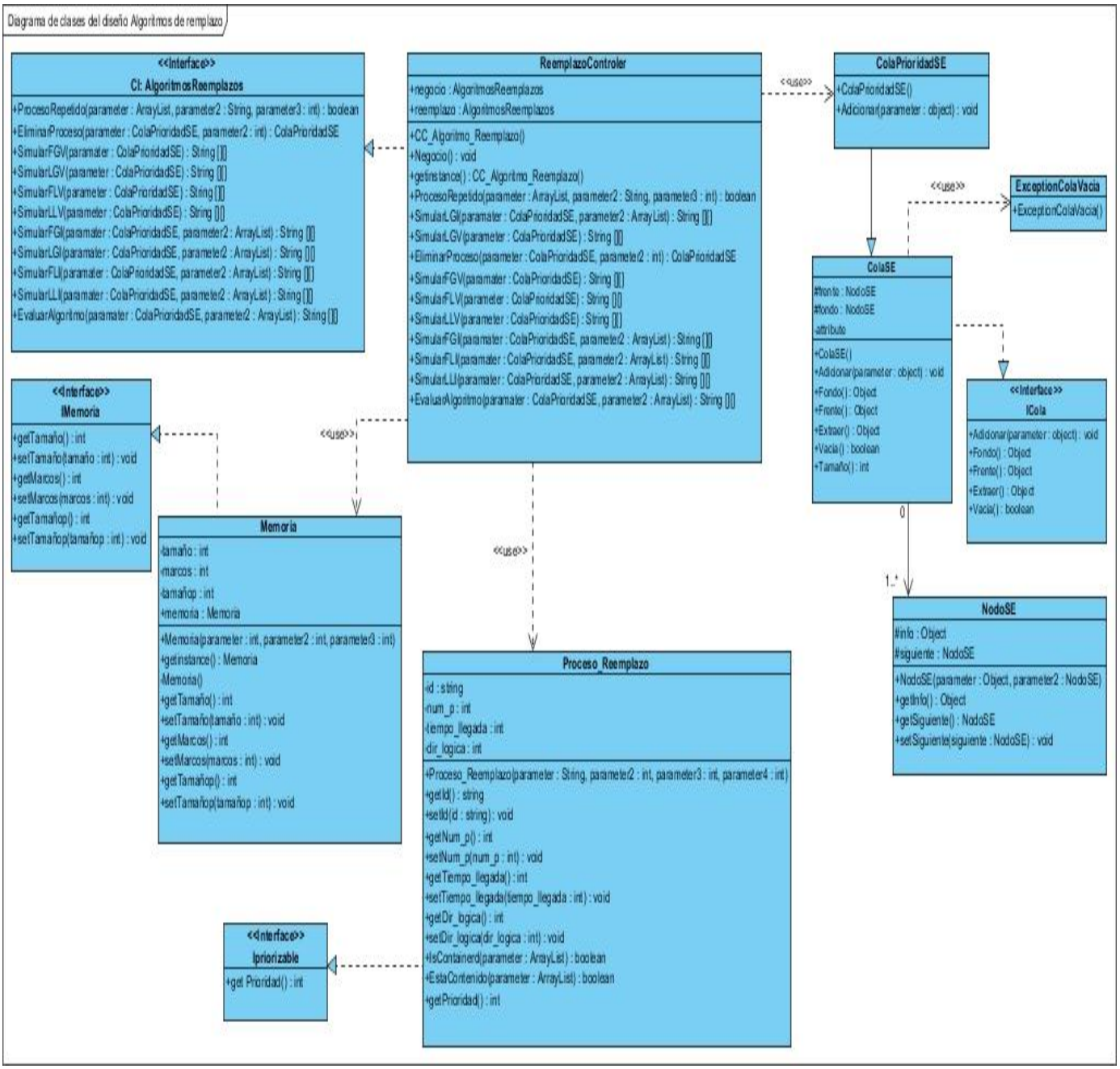
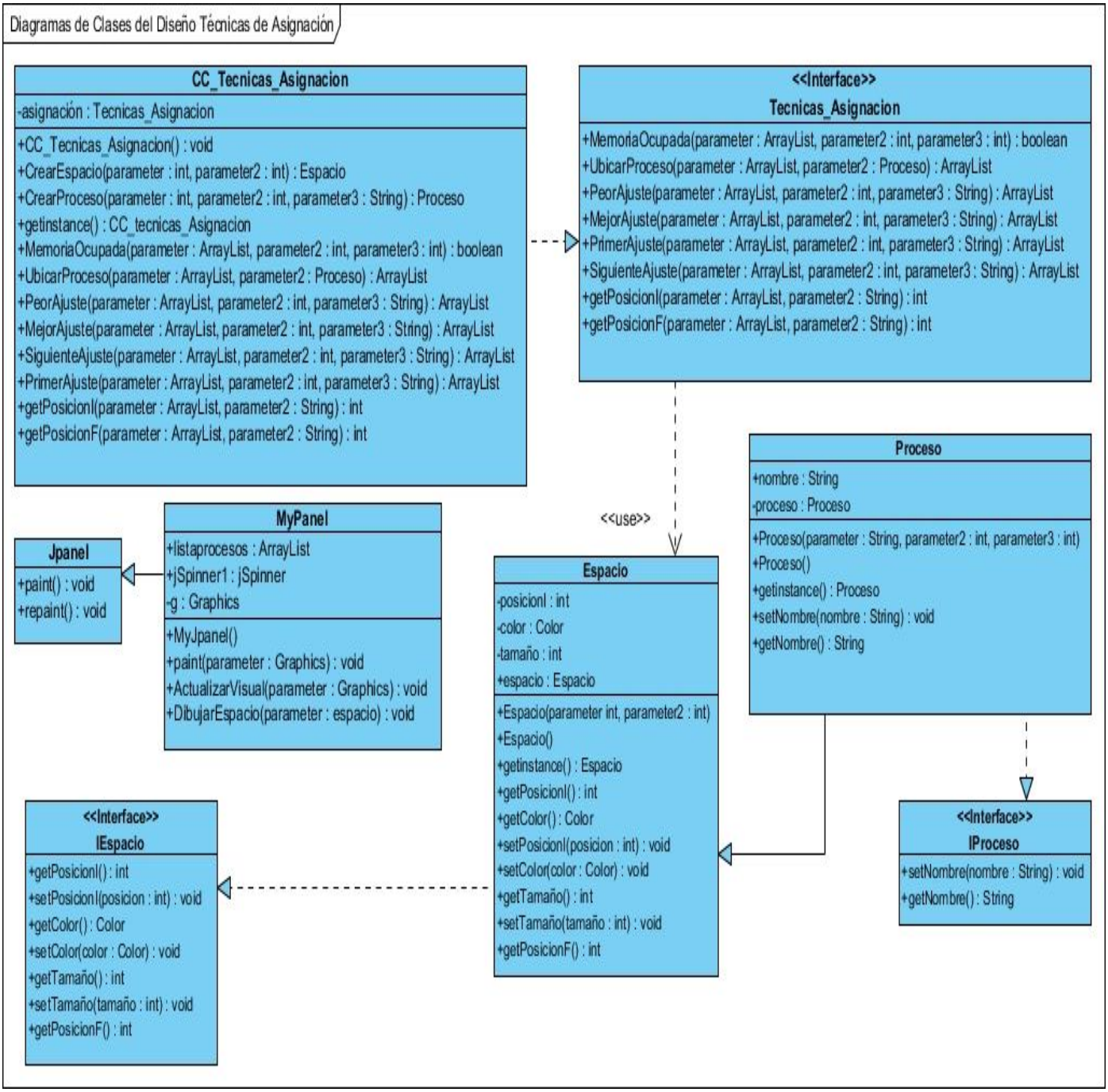
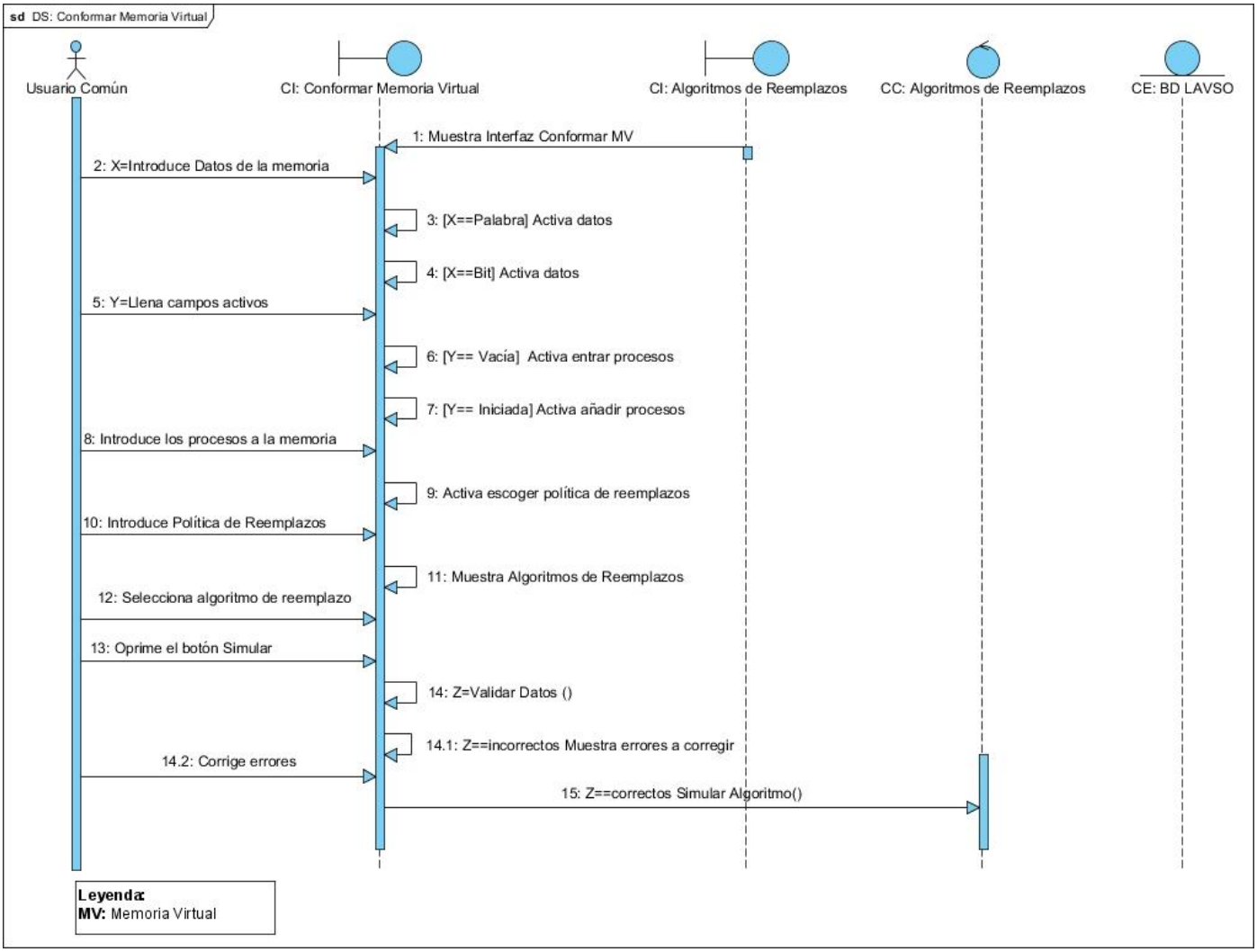


Diagrama de clases “Evaluar Técnicas de Asignación”

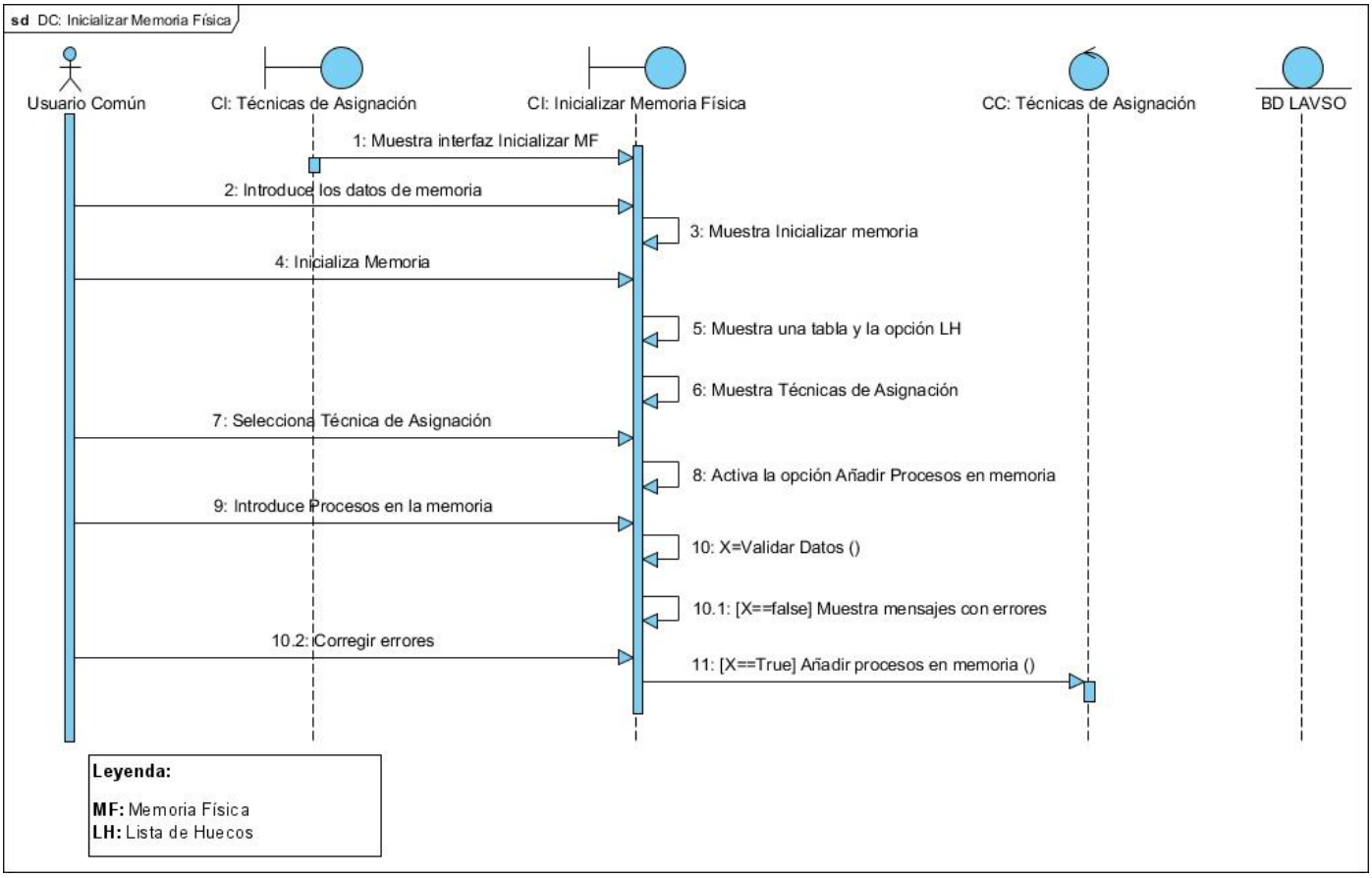
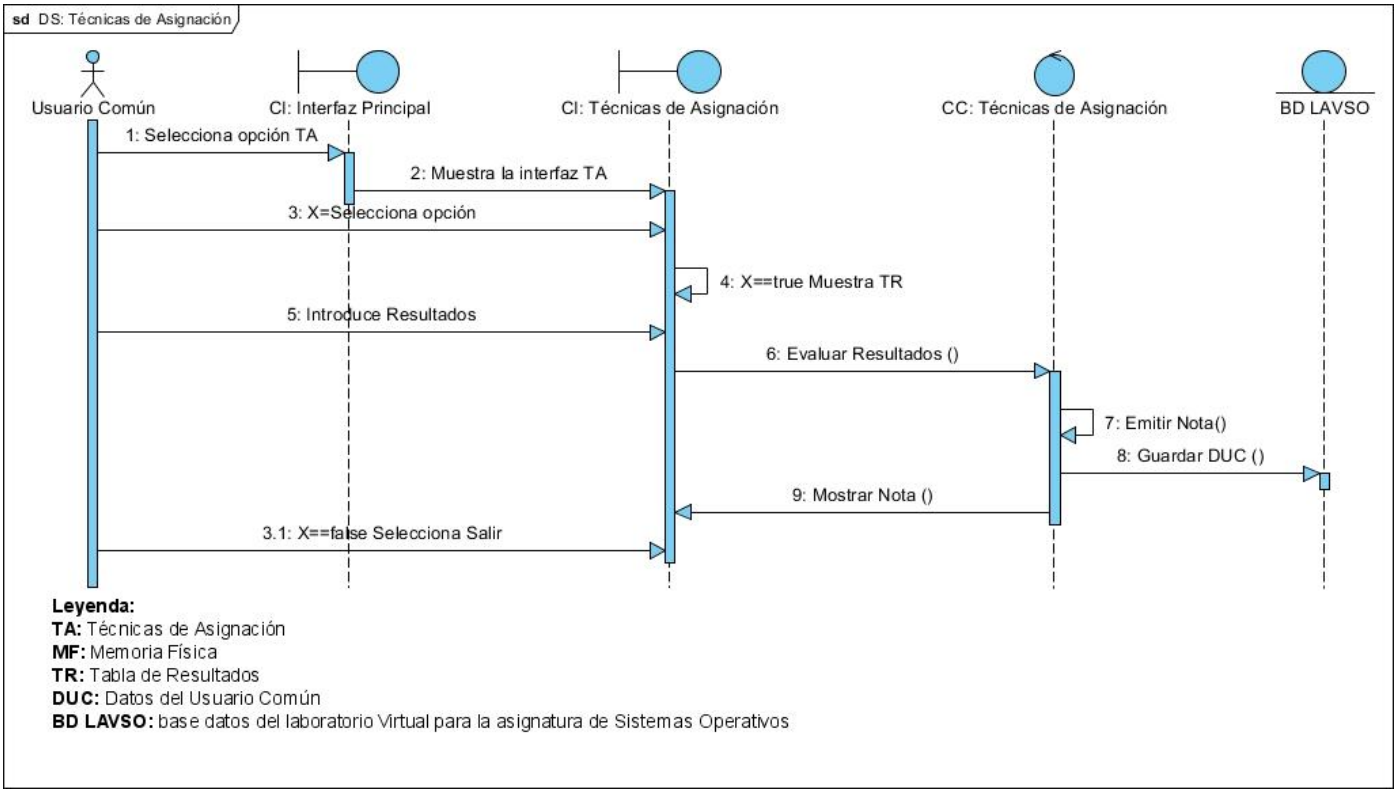


Anexo 3 Diagramas de Interacción

Diagramas de Interacción “Conformar Memoria Virtual”



Diagramas de Interacción “Inicializar Memoria Física”



Diagramas de Interacción “Evaluar Técnicas de Asignación”