“IMPLEMENTACIÓN DEL SIMULADOR DEL TEMA DE GESTIÓN DE DISPOSITIVOS DE ENTRADA Y SALIDA PARA EL LABORATORIO VIRTUAL DE LA ASIGNATURA SISTEMAS OPERATIVOS”

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

Autor: Santiago Vidal Fernández
Tutor: Ing. Carlos Yasmany Hidalgo García
Tutor: Ing. Misael Rodríguez Urrutia
Universidad de las Ciencias Informáticas

UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 3

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

TÍTULO: IMPLEMENTACIÓN DEL SIMULADOR DEL TEMA DE GESTIÓN DE DISPOSITIVOS DE ENTRADA Y SALIDA PARA EL LABORATORIO VIRTUAL DE LA ASIGNATURA SISTEMAS OPERATIVOS

AUTOR: Santiago Vidal Fernández

TUTOR: Ing. Carlos Yasmany Hidalgo García

TUTOR: Ing. Misael Rodríguez Urrutia

La Habana, Julio de 2011.

"Año 53 de la Revolución"

Dedicatoria

A mi familia y amigos.

A la dirección de la Facultad 3 y la FEU de la UCI.

Agradecimientos

A mi Papá por servirme de ejemplo en todo.
A mi Mamá por ser la mejor madre del mundo.
A mi Lil' Sister por estar ahí siempre para mí.
A mi Abuela por estar preguntándome siempre por mi tesis.
A todos mis amigos dentro y fuera de la UCI que me acompañaron por este viaje, para los que terminaron y para los que lo terminaran, Javier, Yailen (La Flaka), Saballo, Sandro, Yoiner, Barlia, Misael, Cucho, Alema, Yadira, Omarito, Rainel, el Chino, Yoena y a todos los que no puse y deberían estar aquí, a los que compartieron conmigo de alguna forma u otra y a los que me ayudaron o me preguntaban por mi tesis.
Gracias.

Declaración de autoría

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____	_____
Santiago Vidal Fernández	Ing. Carlos Yasmany Hidalgo García

Resumen

En la actualidad las Tecnologías de la Información y las Comunicaciones (TICs) forman parte de la cultura tecnológica que rodea el mundo y con la que se debe convivir. La gama de servicios que brindan, posibilita la creación de modelos, procesos y programas; lo que permite controlar y simular actividades reales o virtuales. Es por ello que el presente trabajo está enmarcado en desarrollar un simulador que apoye la visualización de los contenidos del tema Gestión de Dispositivos de Entrada y Salida en la asignatura Sistemas Operativos, puesto que la misma posee cierta complejidad, 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, además de la simulación por computadora, un 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.

Posteriormente se desarrolló una herramienta que permite simular los diferentes algoritmos de planificación así como otros ejercicios del tema de Gestión de Dispositivos de Entrada y Salida de la asignatura de Sistemas Operativos, donde los estudiantes pueden encontrar un entorno de apoyo en su aprendizaje del tema.

Finalmente se aplicaron pruebas dirigidas a evaluar la calidad del sistema, obteniéndose resultados satisfactorios.

PALABRAS CLAVES:

Laboratorio Virtual, Simulación, Gestión de Dispositivos de Entrada y Salida, Disco, Sistemas Operativos, Tecnología.

Índice

Tablas y figuras

Introducción

La informática es una ciencia que se encuentra en continua evolución y está presente en cada uno de los sectores sociales del mundo, es por ello que con el aumento de su conocimiento han ido surgiendo medios que le aportaron creatividad como las Tecnologías de la Información y las Comunicaciones (TIC), dándole un nuevo giro al mundo y proporcionando oportunidades en el desarrollo científico. Uno de sus aportes facilitó una revolución al proceso enseñanza-aprendizaje generando nuevas vías para la educación a distancia.

Durante un largo período fueron apareciendo disímiles sistemas informáticos y definiciones que fueron dando a luz a una nueva tendencia: los Laboratorios Virtuales, que con la llegada de las comunicaciones digitales de alta velocidad conjuntamente con los costos decrecientes de los servicios fijos, han permitido su desarrollo con mayor eficiencia y menor movimiento de personal, sin embargo, no se considera que vaya a suplantar a los Laboratorios Tradicionales ni competir con ellos, pero si abren nuevas perspectivas que eran poco posibles de explorar dentro de estos a un precio asequible.

Cuba no se encuentra ajena a las nuevas tendencias provocadas por las TICs y en función de insertarse como productores de software ha ido vinculando las ramas: económicas, políticas y sociales al mundo informático, consideradas de interés primordial para el Estado Cubano, no solo por los beneficios que trae desde el punto de vista del desarrollo de sistemas para el uso interno, sino también con el objetivo de introducirse en el mercado a escala mundial aprovechando sus perspectivas económicas. El país desarrolló una serie de estrategias que tienen como objetivo aumentar la cultura y el conocimiento de la informática en el país. Algunas de estas estrategias son la creación de los Joven Clubs y el surgimiento de la universidad de nuevo tipo llamada la Universidad de las Ciencias Informáticas (UCI), la cual se encuentra inmersa en profundos cambios para ayudar a elevar la calidad del proceso de enseñanza-aprendizaje, implementando el nuevo modelo de formación docencia-producción-investigación, permitiendo a los estudiantes responder a las exigencias actuales demandadas por la dirección del país.

Los Sistemas Operativos (SO) surgen para facilitar la interacción entre persona y computadora y es por ello que se manifiesta la necesidad de estudiarse como parte de una asignatura. La misma es impartida en la UCI dentro del Programa Analítico del (P1) formando parte del conjunto de disciplinas brindadas en el tercer año de la carrera.

La asignatura se divide en tres temas fundamentales, uno de estos temas es la gestión de dispositivos de entrada y salida, concerniente a la administración de recursos como función fundamental de un SO real. La enseñanza de este tema se torna profundamente compleja debido a su gran componente teórico y al alto nivel de abstracción que requiere por parte de los estudiantes para adquirir un buen entendimiento del funcionamiento interno sobre cada administrador involucrado como el Planificador de Disco.

Las observaciones realizadas durante la impartición de las clases de SO han arrojado resultados acerca de la comprensión por parte de los estudiantes, los cuales resultaron muy positivos cuando el profesor más que limitarse a la simple explicación, hace uso de medios didácticos basados en las Tecnologías de la Información y las Comunicaciones (TICs), que aunque no tenían toda la calidad necesaria, sí permitieron simular el funcionamiento de los diferentes algoritmos de planificación de disco de una manera más práctica.

Luego de un análisis detallado se observó que la asignatura SO carece de una adecuada integración con las TICs provocando que las actividades sean mayormente presenciales, entrando en contradicción con el nuevo modelo de formación, donde la semipresencialidad y la amplia y adecuada integración con las TIC son los componentes principales para lograr la unificación (docencia – producción – investigación).

A raíz de los elementos anteriormente expuestos surge la idea por parte del colectivo de la asignatura, la implementación y puesta en práctica de una herramienta de calidad que funcione como Laboratorio Virtual Simulador del tema de Gestión de Dispositivos de Entrada y Salida, para apoyar las clases del profesor y favorecer el aprendizaje de los estudiantes acerca del funcionamiento interno real de un sistema operativo moderno.

Teniendo en cuenta lo antes planteado se identifica el siguiente **problema de la investigación**: Los medios de apoyo usados por la asignatura Sistemas Operativos para ilustrar el contenido de Gestión de Entrada y Salida no son suficientes para favorecer el aprendizaje del tema. Este problema se enmarca en el **objeto de estudio**: Proceso de desarrollo de software, cuyo **campo de acción** es el proceso de desarrollo de software para la creación de simuladores. En este sentido se define como objetivo general: desarrollar un simulador que apoye la visualización de los contenidos del tema Gestión de Entrada y Salida para el Laboratorio Virtual de la asignatura Sistemas Operativos.

Como **idea a defender** se define que con el desarrollo de un simulador que apoye la visualización de los contenidos del tema Gestión de Entrada y Salida en la asignatura Sistema Operativo se incrementarán los medios que se utilizan en la asignatura para el aprendizaje de los estudiantes, permitiendo a los mismos lograr una mejor comprensión de la materia.

Para garantizar el desarrollo exitoso del objetivo general se han definido una serie de **objetivos específicos** que lograrán el desarrollo eficiente de esta investigación:

- ✓ Elaborar el marco teórico de la investigación.
- ✓ Analizar los artefactos generados durante fases anteriores, así como seleccionar los estándares de codificación y la arquitectura.
- ✓ Implementar las funcionalidades del sistema y validar la aplicación.

Métodos de investigación:

Métodos teóricos:

- *Análisis Histórico – Lógico*: Para profundizar en los antecedentes de la utilización de las Tecnologías de la Información y las Comunicaciones 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 auto gestionado y la introducción de las TIC en el proceso de enseñanza-aprendizaje.

Métodos empíricos:

- *Entrevista:* Se utilizó 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 Sistemas Operativos I en la Facultad 3 de la Universidad de las Ciencias Informáticas.

A lo largo del presente trabajo se muestran 3 capítulos fundamentales los cuales se enuncian a continuación.

Capítulo 1: Fundamentación teórica. En este capítulo se resume el estudio realizado sobre los Laboratorios Virtuales y la simulación por computadora. Se aborda el tema de las metodologías de desarrollo de software 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. Se describen además los requisitos establecidos como historias de usuario. Se describen las herramientas y lenguajes seleccionados para el modelado y desarrollo. También se describen las principales clases y funcionalidades a implementar.

Capítulo 3: Validación de la solución propuesta. En este capítulo se aborda el tema referente a la validación de la solución propuesta. Las pruebas estructurales o de caja blanca y las pruebas de caja negra aplicadas al sistema. Finalmente se obtiene un software que permita la modelación de los procesos que ocurren en el administrador de disco, y a la vez hacer más comprensible este proceso a ojos de los estudiantes.

CAPÍTULO 1: Fundamentación teórica

1.1. Introducción

En el siguiente capítulo se tratan diferentes temas entre los que se encuentran: qué es un laboratorio virtual, la variedad que existe de los mismos, así como aquellos afines a la investigación. Se presenta un análisis de las propiedades significativas dentro del proceso de enseñanza con sus ventajas y desventajas. De conjunto se estudia la simulación por computadora y los beneficios que aportan al estudiante. Partiendo del uso del paradigma de la programación orientada a objetos se realiza un estudio sobre metodologías de desarrollo, herramientas CASE y lenguajes de implementación que se utilizaran en el desarrollo de este trabajo.

1.2. ¿Qué es un Laboratorio Virtual?

La estructuración de información mediante sistemas hipermedia y multimedia, y las redes de comunicación de área extendida, es decir, Internet, son herramientas valiosas en la creación de sistemas de apoyo al aprendizaje de materias dotadas de una componente práctica no muy fuerte. Una de las soluciones de e-learning más interesantes son los e-laboratorios. 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.

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. (1)

1.2.1. Laboratorio Virtual

Muchos han sido los autores que han dado su definición de lo que es un Laboratorio Virtual, a continuación se citan algunos ejemplos:

Según James P. Vary, un laboratorio virtual 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".(2)

Julián Monge Nájera define un laboratorio virtual como "simulaciones de prácticas manipulativas que pueden ser hechas por el estudiante lejos de la universidad y el docente". (3)

Después de estudiar las definiciones dadas por varios autores, se entiende que un Laboratorio Virtual 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. Laboratorios Virtuales: Ventajas e Inconvenientes

Un laboratorio virtual (LV) es un sistema computacional que pretende aproximar el ambiente de un laboratorio tradicional (LT). Los experimentos se realizan paso a paso, siguiendo un procedimiento similar al de un LT: se visualizan instrumentos y fenómenos mediante objetos dinámicos (applets de Java o Flash, cgi-bin, JavaScript), imágenes o animaciones. Se obtienen resultados numéricos y gráficos, tratándose éstos matemáticamente para la obtención de los objetivos perseguidos en la planificación docente de las asignaturas. El uso de laboratorios virtuales 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:[4]

- Acerca y facilita a un mayor número de alumnos la realización de experiencias, aunque alumno y laboratorio no coincidan en el espacio. 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.

- Es una herramienta de auto aprendizaje, donde el alumno altera las variables de entrada, configura nuevos experimentos, aprende el manejo de instrumentos, personaliza el experimento, etc. La simulación en el LV, permite obtener una visión más intuitiva de aquellos fenómenos que en su realización manual no aportan suficiente claridad gráfica. El uso del laboratorio virtual da lugar a cambios fundamentales en el proceso habitual de enseñanza, en el que se suele comenzar por el modelo matemático. La simulación interactiva de forma aislada posee poco valor didáctico, ésta debe ser embebida dentro de un conjunto de elementos multimedia que guíen al alumno eficazmente en el proceso de aprendizaje. Se trata de utilizar la capacidad de procesamiento y cálculo del ordenador, incrementando la diversidad didáctica como complemento eficaz de las metodologías más convencionales.

- Los estudiantes aprenden mediante prueba y error, sin miedo a sufrir o provocar un accidente, sin avergonzarse de realizar varias veces la misma práctica, ya que pueden repetirlas sin límite; sin temor a dañar alguna herramienta o equipo. Pueden asistir al laboratorio cuando ellos quieran y elegir las áreas del laboratorio más significativas para realizar prácticas sobre su trabajo.

- En Internet encontramos multitud de simulaciones de procesos físicos (en forma de applets de Java y/o Flash). Con estos objetos dinámicos, el docente puede

preparar actividades de aprendizaje que los alumnos han de ejecutar, contestando al mismo tiempo las cuestiones que se les plantean.

No todo son ventajas en los laboratorios virtuales, también existen inconvenientes. A continuación mostramos los más destacados:[1]

- El laboratorio virtual 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. En Internet existe demasiada información, a veces inútil. Para que sea útil en el proceso de enseñanza/aprendizaje, hemos de seleccionar los contenidos relevantes para nuestros alumnos. Son pocas las experiencias realizadas con laboratorio virtual en los centros educativos, donde aún impera el uso de recursos tradicionales, tanto en la exposición de conocimientos en el aula como en el laboratorio.

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. Laboratorios Virtuales de 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 necesiten los recursos de un servidor determinado para funcionar(como bases de datos o módulos de software de servidor). También determinados laboratorios virtuales pensados inicialmente como aplicaciones Java accesibles a través de un servidor Web se pueden considerar de este tipo si funcionan localmente y no necesitan recursos de un servidor en concreto.(5)

1.2.4. 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 un 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. (5)

1.2.5. Laboratorios Virtuales 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 le 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.(6)

La diferencia entre el Laboratorio Remoto (LR) y el resto de los Laboratorios Virtuales, reside en el tipo de computación subyacente y tratamiento del material: el LR se basa en instrumentos reales (tarjetas de adquisición de datos, instrumentos de medida, conexiones en interfaces diversas, comunicación de datos, entre otros. Mientras que en el resto de los LV sólo existen procesos de computación basados en simulaciones, ya

sean applets de Java, Flash, o bien programas o ambientes computacionales ejecutados en ordenadores aislados o en una Intranet. Los LR presentan mayores ventajas entre los LV, debido a que éstos proporcionan un mayor nivel de interactividad y el alumno entra en contacto con equipamiento real, en lugar de entrar en contacto con programas simulados. Los LR son una innovación en el campo de la Educación y habrá que prestar atención tanto a su diseño como al estudio de las ventajas e inconvenientes, desde el punto de vista didáctico. A continuación mostramos algunas ventajas importantes de los LR. [(1)]

- Permite aprovechar los recursos, tanto humanos como materiales en los LT. Al integrar, en un único ordenador, los instrumentos necesarios para la ejecución de las prácticas, el ahorro en material de laboratorio es considerable. Se podría pensar que el alumno pierde así la perspectiva real, lo cual es erróneo ya que, por un lado, los instrumentos virtuales diseñados son idénticos a los reales y, por otro, la respuesta de los sistemas es la de un sistema real y no utiliza la simulación más que para la comparación de los resultados. Donde se obtiene un gran aprovechamiento, es a través de una Intranet o a través de Internet, ya que supone no tener que duplicar los materiales y poder acceder a ellos a través de la red como si se estuviese en el mismo puesto.

En la asignatura Sistemas Operativos se están dando los primeros pasos en la realización de un laboratorio virtual web que se pretende incorporar al EVA, que va a tener integrado el trabajo de diploma realizado que forma parte de otro de sus módulos, con una aplicación de escritorio que es un simulador; facilitando que cualquier profesional o alumno interesado en el tema desarrollado pueda hacer uso del mismo y beneficiarse de sus prestaciones.

1.3. ¿Qué es la Simulación por Computadora?

La simulación es el proceso de diseñar un modelo de un sistema real y llevar a término experiencias con él, con la finalidad de comprender el comportamiento del sistema o evaluar nuevas estrategias, dentro de los límites impuestos por un cierto criterio o un conjunto de ellos, para el funcionamiento del sistema.

Además es de mucha utilidad para el aprendizaje de los estudiantes en la asignatura Sistema Operativo, ya que la integración de un software educativo permitirá la incorporación de la tecnología informática. Aportando herramientas que favorezcan el desempeño profesional de los alumnos, estos podrán experimentar en entornos que representan la realidad, a través de modelos de la misma, pero de una forma más interactiva y constructivista. También fomenta la creatividad, el aprendizaje por descubrimiento y la enseñanza individualizada. Al mismo tiempo le es de mucha ayuda al estudiante en su proceso de auto aprendizaje porque puede observar paso a paso lo que quiere aprender de una forma más amena y apreciable.[7]

1.3.1. Tipos de Simuladores

- *Simulador de conducción:* Permiten a los alumnos de autoescuela enfrentarse con mayor seguridad a las primeras clases prácticas, además de permitirles practicar de manera ilimitada situaciones específicas (aparcamientos, incorporaciones desde posiciones de escasa visibilidad, conducción en condiciones climatológicas adversas). Uno de estos simuladores es SIMESCAR, desarrollado por la firma SIMUMAK.
- *Simulador de conducción:* Simulador de SIMPRO creado en la UCI.
- *Simulador de carreras:* Es el tipo de simulador más popular; se puede conducir un automóvil, motocicleta, camión. Ejemplos: rFactor, GTR, GT Legends, toca racer.
- *Simulador de vuelo o de aviones:* Permite dominar el mundo de la aviación y pilotar aviones, helicópteros. Ejemplo: Microsoft Flight Simulator, X-Plane.
- *Simulador de trenes:* Permite controlar un tren. Ejemplo: Microsoft Train Simulator, Trainz BVE Trainsim.
- *Simulador de vida o de dinámica familiar:* Permite controlar una persona y su vida. Ejemplo: El video juego “Los Sims”.
- *Simulador de negocio:* Permite simular un entorno empresarial. Es posible jugar diferentes roles dentro de las funciones típicas de un negocio. Ejemplo: EBSims, Market Place, Flexsim.
- *Simulador político:* Permite hacer el rol de político. Ejemplo: Las Cortes de Extremapol, Política XXI, Simupol, Dolmatovia.
- *Simulador de redes:* permite simular redes. Ejemplo: Omnet++, ns2.

- *Simulador clínico médico:* permite realizar diagnósticos clínicos sobre pacientes virtuales. El objetivo es practicar con pacientes virtuales casos clínicos, bien para practicar casos muy complejos, preparando al médico para cuando se encuentre con una situación real o bien para poder observar como un colectivo se enfrenta a un caso clínico, para poder sacar conclusiones de si se está actuando correctamente, siguiendo el protocolo de actuación establecido. Ejemplo: Simulador clínico Mediteca.
- *Simulador musical:* permite reproducir sonidos con un instrumento de juguete. Ejemplo, Guitar Hero, Dj Hero, Band Hero de Activision Blizzard y Rock Band de Harmonix.

1.3.2. Simuladores en la educación

Los simuladores constituyen un procedimiento, tanto para la formación de conceptos y construcción en general de conocimientos, como para la aplicación de éstos a nuevos contextos a los que por diversas razones, el estudiante no puede acceder desde el contexto metodológico donde se desarrolla su aprendizaje. De hecho, buena parte de la ciencia puntera se basa cada vez más en el paradigma de la simulación, más que en el experimento en sí.

Mediante los simuladores se puede por ejemplo, desarrollar experimentos de química en el laboratorio de informática con mayor seguridad. Así como si a un estudiante se le ocurre agregar más de un determinado líquido y esto cause una explosión, solo será una simple "simulación" y cuando vaya a realizarlo en la práctica él estará informado de las consecuencias de este proceso.

Características en la educación:

- Apoyan aprendizaje de tipo experimental y conjetural.
- Permiten la ejercitación del aprendizaje.
- Suministran un entorno de aprendizaje abierto basado en modelos reales.
- Alto nivel de interactividad
- Tienen por objeto enseñar un determinado contenido.
- El usuario trata de entender las características de los fenómenos, cómo controlarlos o qué hacer ante diferentes circunstancias.

- Promueven situaciones excitantes o entretenidas que sirven de contexto al aprendizaje de un determinado tema.
- El usuario es un ser activo, convirtiéndose en el constructor de su aprendizaje a partir de su propia experiencia.

Después de haber estudiado acerca de los simuladores y el rol importante que juegan estos en la educación, se puede concluir que en la actualidad no existe un simulador el cual pueda dominar y simular los contenidos de gestión de dispositivos de entrada y salida de la asignatura Sistemas Operativos, lo que conlleva a desarrollar una herramienta que permita realizar ejercicios, así como simular el comportamiento de los distintos algoritmos de planificación presentes en el tema.

1.4. Metodologías de desarrollo

El desarrollo de software no es sin dudas una tarea fácil. Como resultado a este problema ha surgido una alternativa desde hace mucho tiempo: la Metodología. Las metodologías imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar inspirado por otras disciplinas de la ingeniería, también su propósito es establecer un contrato social entre todos los participantes en un proyecto para conseguir la solución más eficaz con los recursos disponibles.

Debido a que el proyecto no se considera de gran envergadura, el equipo de desarrollo es pequeño y no se requiere la generación de gran cantidad de artefactos; se decide seleccionar una metodología ágil y no una metodología tradicional.
Cada una de las metodologías agiles tiene características propias y hace hincapié en algunos aspectos más específicos. A continuación se hace un estudio de las principales metodologías agiles

1.4.1. SCRUM

Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de

software se realiza mediante iteraciones, denominadas *sprints*, con una duración de 30días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente.

La segunda característica importante son las reuniones a lo largo proyecto. Éstas son las verdaderas protagonistas, especialmente la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.(8)

1.4.2. Crystal Methodologies

Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo (de ellas depende el éxito del proyecto) y la reducción al máximo del número de artefactos producidos. Han sido desarrolladas por Alistair Cockburn. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo CrystalClear (3 a 8 miembros) y Crystal Orange (25 a 50 miembros).(9)

1.4.3. XP

La Programación eXtrema se basa en la simplicidad, la comunicación y el reciclado continuo de código, para algunos no es más que aplicar una pura lógica. Es utilizada para proyectos de corto plazo. Es la más destacada de las metodologías ágiles de desarrollo de software, estas ponen más énfasis en la adaptabilidad que en la previsibilidad.

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como

especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. (10)

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Kent Beck, el padre de XP, describe su filosofía (11) sin cubrir los detalles técnicos y de implantación de las prácticas. Posteriormente otras publicaciones de experiencias se han encargado de dicha tarea. A continuación presentaremos las características esenciales de XP, organizadas en los apartados siguientes:

Las Historias de Usuario

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. (10)

Prácticas XP

- **El juego de la planificación:** Esta práctica se puede ilustrar como un juego, donde existen dos tipos de jugadores: Cliente y Programador. El cliente establece la prioridad de cada historia de usuario, de acuerdo con el valor que aporta para el negocio. Los programadores estiman el esfuerzo asociado a cada historia de usuario. Se ordenan las historias de usuario según prioridad y esfuerzo, se define el contenido de la entrega y/o iteración, apostando por enfrentar lo de más valor y riesgo cuanto antes.

- **Entregas pequeñas:** La idea es producir rápidamente versiones del sistema que sean operativas, aunque obviamente no cuenten con toda la funcionalidad pretendida para el sistema, pero sí que constituyan un resultado de valor para el negocio.

- **Metáfora:** En XP no se enfatiza la definición temprana de una arquitectura estable para el sistema. Dicha arquitectura se asume evolutiva y los posibles inconvenientes que se generarían por no contar con ella explícitamente en el

comienzo del proyecto se solventan con la existencia de una metáfora. El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema.

- **Diseño simple:** Se debe diseñar la solución más simple que pueda funcionar, la complejidad innecesaria y el código extra deben ser removidos inmediatamente.

- **Pruebas:** La producción de código está dirigida por las pruebas unitarias. Estas pruebas son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del sistema. Los clientes escriben las pruebas funcionales para cada historia de usuario que deba validarse.

- **Refactorización (Refactoring):** La refactorización es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios.

- **Integración continua:** Cada pieza de código es integrada en el sistema una vez que esté lista. Así el sistema puede llegar a ser integrado y construido varias veces en un mismo día.

- **40 horas por semana:** Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. El trabajo extra desmotiva al equipo.

- **Cliente in-situ:** El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Gran parte del éxito del proyecto XP se debe a que es el cliente quien conduce constantemente el trabajo y los programadores pueden resolver de manera inmediata cualquier duda asociada.

- **Estándares de programación:** XP enfatiza la comunicación de los programadores a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación.

- **Comentarios respecto de las prácticas:** El mayor beneficio de las prácticas se consigue con su aplicación conjunta y equilibrada puesto que se apoyan unas en otras.

Históricamente las metodologías tradicionales han intentado abordar la mayor cantidad de situaciones de contexto del proyecto, exigiendo un esfuerzo considerable para ser adaptadas sobre todo en proyectos pequeños y con requisitos muy cambiantes. La metodología XP ofrece una solución casi a medida para una gran cantidad de proyectos que tienen estas características. Una de las cualidades más destacables en la metodología XP es su sencillez, tanto en su aprendizaje como en su aplicación, reduciéndose así los costos de implantación en un equipo de desarrollo. (10)

No existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo de software. Toda metodología debe ser adaptada al contexto del proyecto (recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, entre otros).

La Tabla 1 (12), compara las distintas aproximaciones ágiles en base a tres parámetros: vista del sistema como algo cambiante, tener en cuenta la colaboración entre los miembros del equipo y características más específicas de la propia metodología como son simplicidad, excelencia técnica, resultados, adaptabilidad y prácticas de colaboración.

	SCRUM	Crystal Methodologies	XP
Sistema como algo cambiante	5	4	5
Colaboración	5	5	5
Características Metodologías (CM)			
- Resultados	5	5	5
- Simplicidad	5	4	5
- Adaptabilidad	4	5	3
- Excelencia técnica	3	3	4
- Prácticas de colaboración	4	5	5

Media CM	4.2	4.4	4.4
Media Total	4.7	4.5	4.8

Tabla 1. Ranking de "agilidad".

Después de un estudio de las principales tendencias en metodologías agiles de desarrollo se decide escoger la metodología XP para la realización del simulador, ya que muchas de sus características son aplicables al contexto de realización del proyecto. Principalmente que es ideal para equipos pequeños (en este caso solo una persona), la simplicidad en el código, la frecuente integración del programador y el cliente. Sigue un desarrollo iterativo e incremental, el cual irá dando una visión poco a poco y cada vez más clara de cómo va quedando la herramienta. Es una metodología diseñada para adaptarse a los cambios de requisitos surgidos una vez empezado el proyecto. Exige pocos artefactos como también hace menos énfasis en la arquitectura del software.

1.5. Lenguajes de Modelado

Un sistema tanto del mundo real como en el mundo del software es bastante complejo, por ello es necesario dividir el sistema en partes o fragmentos si se quiere entender y administrar su complejidad. Estas partes se pueden representar como modelos que describan sus aspectos esenciales. Por tanto, un paso útil en la construcción de un sistema de software es el de crear modelos que organicen y comuniquen los detalles más importante de la vida real con que se relacionan y del sistema a construir.

1.5.1. IDEF0

IDEF0 es una técnica de modelación concebida para representar de manera estructurada y jerárquica las actividades que conforman un sistema o empresa y los objetos o datos que soportan la interacción de esas actividades. Un modelo IDEF0 se compone de una serie jerárquica de diagramas que permiten mediante niveles de detalle, describir las funciones especificadas en el nivel superior. En las vistas superiores del modelo la interacción entre las actividades representadas visualiza los procesos fundamentales que sustentan la organización. Los elementos gráficos utilizados para la construcción de los diagramas IDEF0 son cuadros y flechas.

Representa el proceso cronológicamente. Se describe el flujo orientado al cliente final de ese negocio, cruzando todas las actividades de la organización que dan cumplimiento a la solicitud de producto o servicio que realiza el cliente, representando así la "cadena de valor" de la empresa. Permite incorporar en el flujo los datos que entran y salen de las actividades, así como las reglas del negocio y los actores, todo en la misma vista.[3]

1.5.2. BPMN (1.0)

BPMN define un Diagrama de Procesos de Negocio (BPD, del inglés Business ProcessDiagram), que se basa en una técnica de grafos de flujo para crear modelos gráficos de operaciones de procesos de negocio. Un modelo de procesos de negocio, es una red de objetos gráficos, que son actividades (trabajo) y controles de flujo que definen su orden de rendimiento. El BPMN se compone de varios conjuntos de elementos que abarcan la representación, tanto de los Objetos del flujo y sus conexiones como los instrumentos de ayuda que son las Bandas (Swimlanes) y los Artefactos. Además está diseñado para cubrir muchos tipos de modelados y para permitir la creación de segmentos de procesos, así como procesos de negocio end-to-end con diferentes niveles de fidelidad. [13]

1.5.3. Lenguaje de Modelado Unificado (UML 2.0)

El Lenguaje de Modelado Unificado UML es un lenguaje estándar para escribir planos de software. Puede utilizarse para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra gran cantidad de software. Es un lenguaje que ayuda a interpretar grandes sistemas mediante gráficos o texto, obteniendo modelos explícitos que contribuyen a la comunicación durante el desarrollo, ya que al ser estándar pueden ser interpretados por personas que no participaron en su diseño. En este contexto, UML sirve para especificar modelos no ambiguos y completos. UML es un método formal de modelado. Esto aporta las siguientes ventajas:

- Mayor rigor en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se puede automatizar determinados procesos y permite generar código a partir de los modelos y a la inversa. Esto hace que el modelo y el código estén

actualizados, con lo que siempre se puede mantener la visión en el diseño de más alto nivel de la estructura de un proyecto. (14)

UML propone diagramas con la finalidad de presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. Un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementarlo.

Luego de analizadas estas notaciones de modelado se decide utilizar el lenguaje UML, ya que brinda la posibilidad de revisar y modificar con facilidad los modelos entregados en fases anteriores. Además hace que se obtenga una documentación que es válida durante todo el ciclo de vida de un proyecto. Al mismo tiempo es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad para modelar los artefactos creados durante el proceso de desarrollo de software.

1.6. Herramientas CASE

Las herramientas de desarrollo de software han desempeñado un importante papel en el desarrollo de aplicaciones. Como consecuencia del avance tecnológico éstas han experimentado también continuos cambios. Estas herramientas favorecen el apoyo al desarrollo de software, proporcionando un conjunto de programas de asistencia a los analistas para la Ingeniería de Software durante todo el ciclo de vida del desarrollo del sistema.

1.6.1. Rational Rose Enterprise Edition (v7.0)

Rational Rose es la herramienta CASE desarrollada por los creadores de UML que cubre todo el ciclo de vida de un proyecto, desde la fase de inicio, formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases. Permite establecer una trazabilidad real entre el modelo (análisis y diseño) y el código ejecutable. Dentro de sus características principales se puede encontrar un avanzado modelado de UML para trabajar en diseños de bases de datos, con capacidad de representar la integración de los datos y los requisitos de aplicación a través de diseños lógicos y físicos. Posee además una fuerte capacidad para integrarse con cualquier sistema de control de versiones. (3)

Es importante resaltar que a pesar de esto Rational Rose presenta una necesidad de alta capacidad de procesamiento y se necesita tener habilidad y conocimiento de esta herramienta para trabajar con ella.

1.6.2. Enterprise Architect (v7.0)

Enterprise Architect (AE) es una herramienta comprensible de diseño y análisis UML, cubriendo el desarrollo de software desde el paso de los requerimientos a través de las etapas del análisis, modelos de diseño, pruebas y mantenimiento.
EA es un software propietario que permite administrar la complejidad con herramientas para rastrear las dependencias, soporte para modelos muy grandes, control de versiones, líneas base por cada punto del tiempo, además brinda la utilidad de comparar para seguir los cambios del modelo y posee una interfaz intuitiva. (15)

Esta herramienta potente fue construida en base al excepcional éxito de las versiones previas con un completo soporte para el estándar UML 2.1.
Entre sus características principales se destacan:

- Soporta transformaciones de Arquitectura avanzada, compatible con varias plataformas (Windows y Linux).
- Soporta generación y ayuda a la visualización de aplicaciones soportando ingeniería inversa de un amplio rango de lenguajes de desarrollo de software, incluyendo C++, C#, Java, Delphi, VB.Net, Visual Basic y PHP.

1.6.3. Visual Paradigm (v5.0)

Visual Paradigm es una herramienta multiplataforma de modelado visual UML y una herramienta CASE muy fácil de utilizar. Tributa una excelente interoperabilidad con otras herramientas CASE ya que tiene conexión con Rational Rose en sus archivos de proyecto (.MDL / .CAT) los cuales pueden ser importados a Visual Paradigm UML a través de esta importante característica. Visual Paradigm para UML es apoyado por un conjunto de idiomas tanto en la generación del código como en la Ingeniería Inversa por mencionar algunos ejemplos, los cuales tiene la capacidad de soporte, Se puede hablar de Java, C + +, CORBA IDL, PHP, XML Schema, Ada y Python. Admite la captura de

requisitos, análisis, diseño e implementación, también proporciona características tales como generación del código, ingeniería inversa y generación de informes. Permite dibujar todos los tipos de diagramas de clases. Apoya los estándares más recientes de las notaciones de UML. Incorpora el soporte para trabajo en equipo, permitiendo que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios hechos por sus compañeros [(3)]. Visual Paradigm presenta características esenciales, las cuales son:

- Modelado colaborativo con CVS (ConcurrentVersionsSystem) y Subversión.
- Ingeniería inversa, código a modelo, código a diagrama.
- Generación de código, modelo a código, diagrama a código.
- Editor de Detalles de Casos de Uso, entorno todo en uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Diagramas de flujo de datos.
- Generación de bases de datos, transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos, desde Sistemas Gestores de Bases de Datos existentes a diagramas de Entidad-Relación.
- Generador de informes para generación de documentación.
- Distribución automática de diagramas. Reorganización de las figuras y conectores de los diagramas UML.
- Modelo para realizar prototipos de interfaz.

Brinda la posibilidad de generar código a partir de los diagramas, para plataformas como .Net, Java y PHP, así como obtener diagramas a partir de código. [(16)]

Después de investigadas estas herramientas se decide utilizar Visual Paradigm5 ya que genera toda la documentación de lo que se realiza cumpliendo con los estándares establecidos, es una de las pocas herramientas CASE que soporta el análisis textual, siendo esta una técnica útil y práctica para la captura de los requisitos del sistema. Es una herramienta con una interfaz amigable e intuitiva. Otra de las características por la que se decide usar Visual Paradigm es su disponibilidad en múltiples plataformas, ya que no obliga al usuario a desarrollar solo en el sistema operativo Windows, sino que está disponible en otros sistemas operativos como Linux o Unix; mientras que Rational

Rose Enterprise Edition 7.0 solo se instala en la plataforma de Windows. Además influyó en la decisión el hecho de que la universidad posee una licencia de activación de Visual Paradigm.

1.7. Lenguajes de Desarrollo

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.(17)

1.7.1. C#

C# es un lenguaje de programación sencillo y moderno, orientado a objetos y con seguridad de tipos. C# combina la gran productividad de los lenguajes de desarrollo rápido de aplicaciones (RAD, Rapid ApplicationDevelopment) con la eficacia de C++. El código de C# es compilado como código administrado, lo cual le permite beneficiarse de los servicios del CommonLanguajeRuntime (CLR). Estos servicios incluyen interoperabilidad, recolección de basura, seguridad y gestión de versiones. La sintaxis y estructuración de C# es muy similar a la C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en este lenguaje a C# y su aprendizaje a los desarrolladores habituados a este. Es importante aclarar que a pesar de esto para poder instalar C# se requiere de Windows NT4 o superior y por lo menos 4 GB de espacio en el disco duro.(18)

1.7.2. C++

C++ es un lenguaje diseñado a mediados de los años 1980, por BjarneStroustrup. Es un súper conjunto de C que tiene como característica principal, el soporte para programación orientada a objetos y de plantillas o programación genérica. Se creó para añadirle cualidades y características de las que carecía C. Entre los paradigmas que abarca C++ se pueden encontrar la programación estructurada, la programación genérica y la programación orientada a objetos.

Este lenguaje mantiene la potencia para programar a bajo nivel y se la han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción. Una de las razones de programar en C++ es su increíble versatilidad. Con él pueden programarse desde las aplicaciones más simples, a las más complejas como sistemas operativos. Su portabilidad permite que un programa escrito en C++ pueda compilarse en cualquier sistema sin necesidad de cambiar apenas el código. A pesar de esto C++ es un lenguaje de programación difícil de aprender, debido principalmente al trabajo con punteros. (19)

1.7.3. Java

Java es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems 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. Utilizando Java se pueden añadir aplicaciones que vayan desde experimentos científicos interactivos de propósito educativo a juegos o aplicaciones especializadas para la tele venta. Java posee una curva de aprendizaje muy rápida. Resulta relativamente sencillo desarrollar applets interesantes desde el principio. Se puede decir que Java es un lenguaje de programación que facilita la creación de aplicaciones distribuidas ya que proporciona una colección de clases para aplicaciones en red. Una de las características más importantes que tiene Java es la indiferencia a la arquitectura ya que es compatible con los más variados entornos de red cualesquiera sean estos desde Windows 95, Unix a Windows NT y Mac, para poder trabajar con diferentes sistemas operativos. Java es muy versátil ya que utiliza bytecodes que es un formato intermedio que sirve para transportar el código eficientemente o de diferentes plataformas.

Por ser indiferente a la plataforma sobre la cual está trabajando, hace que su portabilidad sea muy eficiente, sus programas sean iguales en cualquiera de las plataformas porque especifica tamaños básicos, esto se conoce como la máquina virtual de Java. (20)

Aunque siendo Java el lenguaje requerido del cliente para desarrollar la herramienta se realizó un estudio de los lenguajes de desarrollo más utilizados, se llega a la conclusión de que Java es el más indicado para desarrollar el sistema que se desea implementar. El hecho de que es multiplataforma es muy importante, su funcionamiento es el mismo en todas las plataformas y sólo cambia la apariencia que se adapta a la del sistema operativo que lo ejecuta (Windows, Solaris, Linux, etc.). Más importante aún es la característica de su indiferencia a la arquitectura sobre la cual se está trabajando, lo que hace que su portabilidad sea muy eficiente y además que sus programas sean iguales en cualquier plataforma debido a la máquina virtual de Java que corre sobre cualquier sistema operativo.

1.8. 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. Se les conoce como un Entorno de Desarrollo Integrado (IDE) por sus siglas en inglés *(Integrated Development Environment).*

1.8.1. IntelliJ IDEA 10.0

IntelliJ IDEA es un IDE comercial de java por Jetbrains. Permite codificar sin un enganche. Practica un método no intrusivo, intuitivo para ayudarte a escribir, depurar, probar y aprender el código. Entre una de sus características principales se encuentra su edición avanzada de código ofreciendo una asistencia inteligente de codificación, generación de código, documentación, navegación y búsqueda, análisis del código sobre la marcha entre otros. Sus lenguajes admitidos son Java, JavaScript/Flex, HTML/XHTML/CSS, XML, Ruby/Jruby entre otros. IntelliJ IDEA se autodefine como un entorno inteligente para desarrollar aplicaciones Java, cliente y servidor. Permite la refactorización en varios idiomas, inspecciones y revisiones de código, análisis de la dependencia. Como requisitos exige un mínimo de 256MB de RAM y como recomendado 1GB lo cual se suma a la limitación de ser de 30 días de prueba y exigir una compra de licencia. [21]

1.8.2. Eclipse 3.4

La herramienta de desarrollo Eclipse IDE comenzó como un proyecto de International Business Machines (IBM) Canadá. Esta herramienta presenta una característica principal y es que emplea módulos (en inglés *plugin*) para proporcionar toda su funcionalidad al frente de la plataforma de cliente rico, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente al permitirle a Eclipse extenderse usando otros lenguajes de programación como son C/C++, Python, permite trabajar con lenguajes para procesado de texto como LaTeX. La arquitectura plugin permite escribir cualquier extensión deseada en el ambiente. El Software Development Kit (SDK) de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de metadata en un espacio para archivos planos, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente.
Esta herramienta presenta características fundamentales cómo son:

- Editor de Texto.
- Resaltado de Sintaxis.
- Compilación en tiempo real.
- Pruebas unitarias con Junit.
- Control de Versiones.
- Asistentes (Wizards), para la creación de proyectos, clases, test, etc.
- Refactorización.

Eclipse al tener disponibles miles de plugins para agregarle lo que le falta de serie. Para crear una aplicación de escritorio con calidad se requiere descargar varios de estos plugins, lo que se torna difícil elegir bien qué plugin usar, que sus versiones sean compatibles con el IDE y entre sí. [(22)]
También Eclipse hace un consumo de memoria mayor que otro de los IDE estudiados, el NetBeans 6.9.

1.8.3. NetBeans 6.9

NetBeans es un proyecto de código abierto fundado y patrocinado por Sun Micro System. Es un IDE 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.

La versión 6.9 que incluye soporte para Java y otras tecnologías, continúa el enfoque de entregas incrementales pequeñas y frecuentes, lo cual lo hace ideal respecto a la tecnología de desarrollo que se está usando. (XP)

Después de hacer un minucioso estudio de las herramientas anteriores, para el desarrollo de la aplicación se hará uso de NetBeans 6.9 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++. Un grupo de ventajas que hace a esta herramienta la correcta para desarrollar la solución con el lenguaje previamente seleccionado (Java) y cumpliendo con la metodología seleccionada (XP). [23]

1.9. Conclusiones parciales

Teniendo en cuenta lo antes expuesto se puede concluir que la asignatura Sistemas Operativos no posee en su estructura un análisis detallado y exhaustivo del contenido de Planificación de Disco que se ajuste a dicha información para la fácil comprensión de los estudiantes.

En el capítulo se define la metodología de desarrollo como también las herramientas a utilizar en la implementación de este trabajo. Como metodología se selecciona la XP (XtremeProgramming) ya que es diseñada especialmente para proyectos de corto plazo, con equipos de trabajo pequeños y genera pocos artefactos. Ventajas que son aprovechables para la realización de la herramienta por solo una persona.

Como lenguaje de modelado para evaluar y modificar los distintos diagramas entregados por la analista que resultaron en la fase de análisis y diseño (Tesis anterior) se escogió el lenguaje UML2.0 y para esto se seleccionó como Herramienta CASE el Visual Paradigm5 ya que cumple con los estándares establecidos. Es una herramienta con una interfaz amigable e intuitiva como también es multiplataforma y no obliga a usar la herramienta solo en el sistema operativo Windows, sino que está disponible en sistemas operativos como Windows, Linux, Unix.

Como lenguaje de desarrollo se escogió el lenguaje Java al ser uno de los requisitos del cliente, también por el hecho de ser multiplataforma, sólo cambia la apariencia que se adapta a la del sistema operativo que lo ejecuta (Windows, Solaris, Linux, etc.), gracias a la maquina virtual de Java. Es ideal para aplicaciones que vayan desde experimentos científicos interactivos de propósito educativo a juegos, lo cual tiene relación con la herramienta que se quiere desarrollar. Para el desarrollo se escogió como herramienta el NETBeans 6.9 ya que soporta el lenguaje seleccionado (Java), tiene un enfoque de entregas incrementales pequeñas y frecuentes lo cual lo hace la herramienta perfecta respecto a la tecnología de desarrollo seleccionada (XP), además de ser libre y de código abierto, lo cual cumple directrices tecnológicas de la Universidad de las Ciencias Informáticas.

CAPÍTULO 2: Descripción de la solución

2.1. Introducción

En el siguiente capítulo se realiza una valoración del diseño propuesto por el analista, además se exponen los diagramas de clases, así como la descripción de las principales clases y sus funcionalidades. Se establecen los estándares de codificación a utilizar en el desarrollo de la aplicación.

2.2. Valoración crítica del análisis y diseño propuesto.

Se obtuvo los artefactos generados durante el flujo de trabajo de análisis y diseño[(24)], los cuales permitieron aumentar la comprensión de todos los aspectos relacionados con los requisitos funcionales y componentes reutilizables.

Uno de los artefactos a tomar en cuenta son las Historias de usuario.

2.2.1. Historias de Usuario

Las historias de usuario son la técnica utilizada en la metodología ágil de desarrollo de software XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento historias de usuario pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.[(25)]

Historia de Usuario	
No. 1	**Nombre**: Asignar nuevas peticiones
Prioridad en Negocio: Alta	**Iteración Asignada:** 1era iteración

Riesgo en Desarrollo: Medio
Descripción: Para asignar nuevas peticiones a la cola de solicitudes, previamente hay que chequear si el disco ya fue inicializado y si tiene solicitudes pendientes a ser atendidas para luego incluir las que llegan a continuación de las ya existentes.
Observaciones: Ninguna

Tabla 2. HU. Asignar nuevas peticiones.

	Historia de Usuario	
No. 2	**Nombre:** *Planificar solicitudes de acceso al disco.*	
Prioridad en Negocio: Alta		**Iteración Asignada:** 1era y 2da iteración
Riesgo en Desarrollo: Alto		
Descripción: El proceso de planificar las solicitudes de acceso al disco inicialmente consiste en inicializar el disco, lo cual se desglosa en establecer el tamaño del disco, luego la cantidad de cilindros, así como la cantidad de sectores por pista, se debe conocer la posición actual en la se encuentra el cabezal y el movimiento ascendente o descendente del brazo. Para establecer el tiempo de posicionamiento es necesario conocer si las peticiones hechas arriban con un tiempo de llegada especificado o si son peticiones con tiempo fijo y a partir del tiempo promedio de acceso. Finalmente se muestran los datos del disco y se puede proceder a planificar las solicitudes seleccionando un algoritmo de planificación.		
Observaciones: Existen varios de estos algoritmos para atender las solicitudes que se le van haciendo a un disco, estos tienen el propósito de disminuir el tiempo de búsqueda de las pistas como son: • FCFS (“First Come First Served”).		

Es el algoritmo más simple de planificación en disco. Este consiste en atender las solicitudes en el orden de llegada. Es un algoritmo equitativo, pero no es el más eficiente.

- SSF o SSTF (“Shortest Seek-Time First”).

Es un algoritmo muy eficiente. Consiste en atender la solicitud cuya pista esté más cerca de la actual posición de la cabeza de lectura/escritura.

- SCAN.

En esta planificación, las cabezas de lectura/escritura comienzan en una de las pistas extremas (la inicial o la última) y se mueven hacia la otra, brindando servicio a todas las solicitudes que existan. Al llegar al otro extremo invierte el sentido del movimiento y se repite toda la operación.

- C-SCAN.

Es una variante del SCAN, se conoce como “scan” circular y siempre presta servicio en el mismo sentido, es decir, se mueve el cabezal de un extremo al otro y después se regresa al inicio para repetir la operación.
El C-SCAN parte de la lógica de que las nuevas solicitudes en el extremo inicial deben ser las que más tiempo deben llevar esperando y además de que en esta área es donde debe haber una mayor concentración de solicitudes pendientes.
En la práctica estos algoritmos no son instrumentados en esta forma, sino que comúnmente la cabeza es movida en un sentido hasta la última solicitud en dicho sentido. Tan pronto como no hay otra, se invierte el sentido del movimiento o se reinicia en la primera de las solicitudes. A estos algoritmos se conocen como:

- LOOK.
- C-LOOK.

Tabla 3. HU Planificar solicitudes de acceso al disco.

Historia de Usuario

No. 3	**Nombre:** *Calcular capacidad del disco duro.*
Prioridad en Negocio: Media	**Iteración Asignada**: 3ra iteración
Riesgo en Desarrollo: Medio	
Descripción: La capacidad de un disco depende de cuantos sectores tiene el mismo. Para averiguar la cantidad de sectores hay que realizar la siguiente operación: Número sectores = Número caras * Número cilindros * Sectores/pista. Una vez que se tiene el número de sectores, se multiplica esta cifra por el tamaño del sector de la forma en que sigue con vista a obtener la capacidad del disco: Capacidad de disco = Número de sectores * Tamaño del sector.	
Observaciones: Se expresa en Kilobytes como unidad de medida.	

Tabla 4. HU Calcular capacidad del disco duro.

Historia de Usuario	
No. 4	**Nombre**: *Calcular dirección lógica de un sector físico.*
Prioridad en Negocio: Media	**Iteración Asignada:** 4ta iteración
Riesgo en Desarrollo: Medio	
Descripción: Es recomendable aclarar que las direcciones físicas tienen la siguiente estructura: (número de cilindro, número de cara, número de sector)	

Además el número de cilindros y de sectores por pista deben conocerse como datos antes de comenzar a calcular. La numeración de las caras se sigue del siguiente modo: hasta que no se han numerado todos los sectores de una cara no se pasa a la siguiente cara del mismo cilindro y hasta que no se hayan numerado todas las caras de un cilindro no se pasa al siguiente. Suponiendo direcciones físicas de la forma (i, j, k), basta con aplicar la fórmula: NB = k + sec/pista * [j + caras/cilindro * (i − 1)].
Observaciones: Ninguna

Tabla 5. HU Calcular dirección lógica de un sector físico.

Historia de Usuario	
No. 5	**Nombre**: *Calcular dirección física de un sector lógico. .*
Prioridad en Negocio: Media	**Iteración Asignada:** 4ta iteración
Riesgo en Desarrollo: Medio	
Descripción: Al igual que para calcular la dirección lógica es recomendable aclarar que las direcciones físicas tienen la siguiente estructura: (número de cilindro, número de cara, número de sector) El número de cilindros y de sectores por pista deben conocerse como datos antes de comenzar a calcular. La numeración de las caras se sigue del siguiente modo: hasta que no se han numerado todos los sectores de una cara no se pasa a la siguiente cara del mismo cilindro y hasta que no se hayan numerado todas las caras de un cilindro no se pasa al siguiente. Si se conoce la dirección lógica, para obtener la dirección física hay que dividir este número conocido por la cantidad de sectores por pista:	

Dirección lógica/Núm. de sectores = Núm. caras llenadas.
Observaciones: Ninguna

Tabla 6. HU Calcular dirección física de un sector lógico

2.2.2. Diseño

El diagrama de clases propuesto aunque se ajusta al problema es necesario realizarle algunos cambios, como añadirle algunas funcionalidades con el fin de darle una solución simple y efectiva a las historias de usuario.

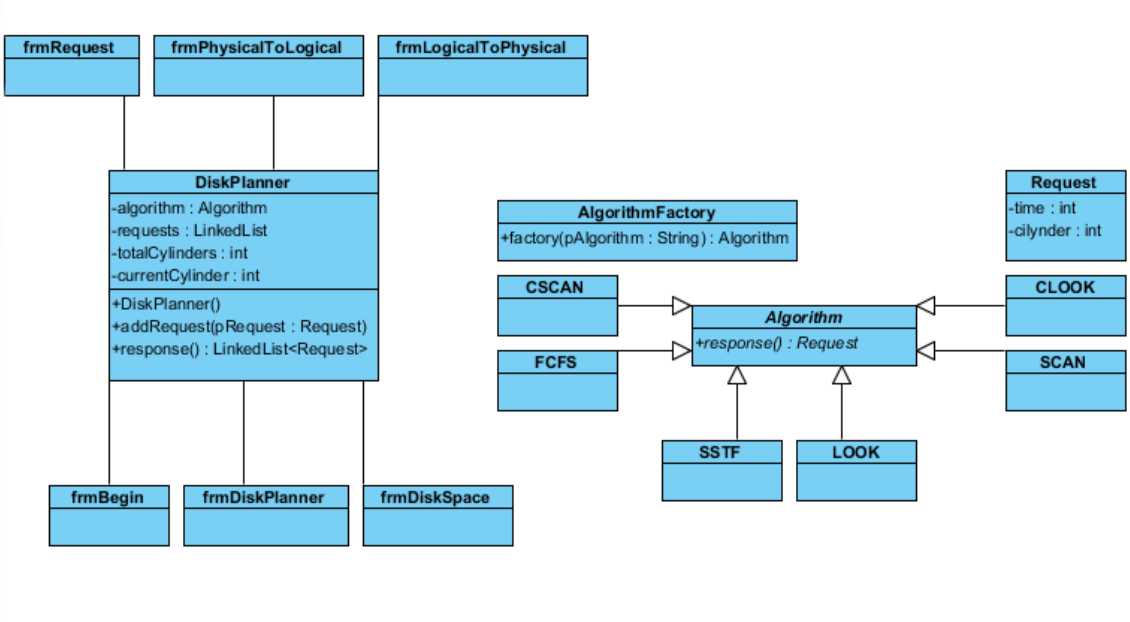


Figura 2. 1. Diagrama de clases propuesto.

2.3. Arquitectura y patrones a emplear en la solución

En los inicios de la informática, la programación se consideraba un arte y se desarrollaba como tal debido a la dificultad que entrañaba para la mayoría de las personas, pero con el tiempo se han ido descubriendo y desarrollando formas y guías generales con base a las cuales se puedan resolver los problemas. A estas se les ha denominado Arquitectura de Software, por su semejanza a los planos de un edificio o

construcción, estas indican la estructura, funcionamiento e interacción entre las partes del software.

2.3.1. Estilo arquitectónico

La arquitectura utilizada para darle solución al problema es la arquitectura en capa o por niveles. La programación por capas es un estilo de programación en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario.

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado.

En el desarrollo del simulador para el tema de Gestión de Dispositivos de Entrada y Salida para la asignatura de Sistemas Operativos se usaran solo dos de las ya comunes tres capas, ya que el sistema no usa ninguna fuente de datos ya sea de entrada como de salida. A continuación se da una breve descripción de cada capa.

- ✓ **Capa de presentación:** es la que ve el usuario (también se la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario. Esta capa se comunica únicamente con la capa de negocio.

- ✓ **Capa de negocio:** es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación para

recibir las solicitudes y presentar los resultados. También se consideran aquí los programas de aplicación.

2.3.2. Patrones de diseño

Los **patrones de diseño** son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su **efectividad** resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser **reusable**, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

2.3.2.1 Factory Method

En diseño de software, el patrón de diseño Factory Method consiste en utilizar una clase constructora (al estilo del Abstract Factory) abstracta con unos cuantos métodos definidos y otro(s) abstracto(s): el dedicado a la construcción de objetos de un subtipo de un tipo determinado. Es una simplificación del Abstract Factory, en la que la clase abstracta tiene métodos concretos que usan algunos de los abstractos; según se use una u otra hija de esta clase abstracta, se tendrá uno u otro comportamiento.

Las clases principales en este patrón son el creador y el producto. El creador necesita crear instancias de productos, pero el tipo concreto de producto no debe ser forzado en las subclases del creador, porque entonces las posibles subclases del creador deben poder especificar subclases del producto para utilizar.

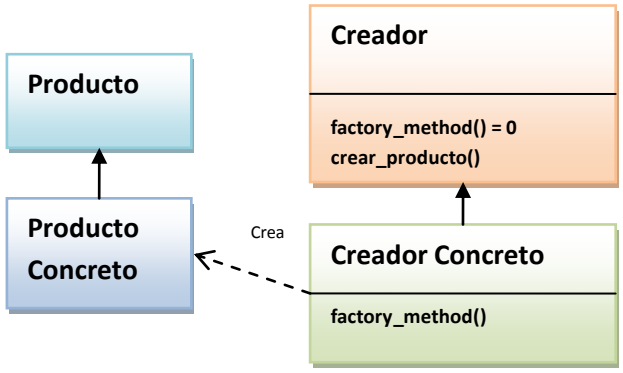


Figura 2. 2. Patrón Factory Method.

Este patrón se ve presente en la clase AlgorithmFactory (Creador) en donde el método *"factory()"* crea un objeto de tipo clase Algorithm (Producto Concreto) dependiendo del *string algorithm* obtenido del formulario.

2.3.2.2 Strategy

Una de las funcionalidades principales de la aplicación es planificar las solicitudes de acceso al disco, puesto que existen varios algoritmos de planificación y el usuario tiene la opción de usar varios de estos en tiempo de ejecución, se decide usar el patrón Strategy.

El patrón Estrategia (Strategy) es un patrón de diseño para el desarrollo de software. Se clasifica como patrón de comportamiento porque determina como se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea. El patrón estrategia permite mantener un conjunto de algoritmos, de entre los cuales el objeto cliente puede elegir aquel que le conviene e intercambiarlo dinámicamente según sus necesidades.

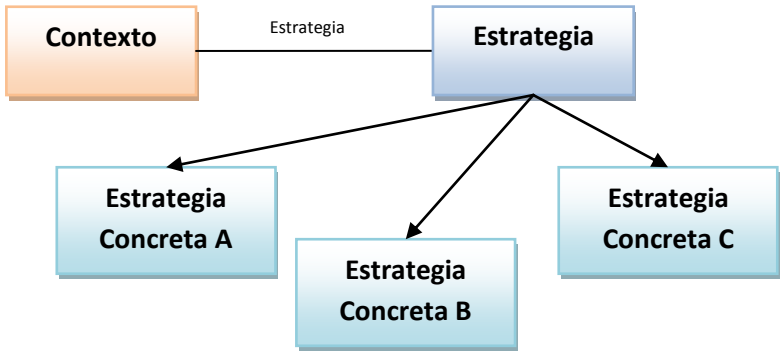


Figura 2. 3. Patrón Strategy.

Basándose en la Figura 2.3 se muestra la clase Algorithm (Estrategia) de la cual heredan las diferentes clases correspondientes a cada algoritmo de planificación: FCFS, SSTF, LOOK, entre otros (Estrategias Concretas A, B, C). La clase DiskPlanner (Contexto) en dependencia del algoritmo de planificación solicitado por el usuario adopta una estrategia diferente.

2.4. Descripción de los principales algoritmos de planificación.

El tema de Gestión de Dispositivos de Entrada y Salida (E/S) en la asignatura Sistemas Operativos se ve compuesto principalmente por el sub-tema de Disco, así como los algoritmos que intervienen en la planificación del rendimiento del manejador de disco. También se desarrollan ejercicios calculando los parámetros principales del disco, ya sean de calcular la capacidad, como la dirección lógica y la dirección física.

En la asignatura se enseñan un conjunto de técnicas que tienen como objetivo fundamental disminuir el tiempo promedio de servicio a las solicitudes. Por consiguiente, dada la importancia que reviste este tema, resulta fundamental conocer cada uno de los algoritmos de planificación para conocer la lógica del negocio.

2.4.1 Algoritmos de Planificación

Existen varios algoritmos para atender las solicitudes que se van haciendo a un disco y que como ya vimos tienen el propósito de disminuir el tiempo de búsqueda de las pistas. Estos se detallan a continuación.

2.4.1.1 FCFS *("First Come First Served")*

Es el algoritmo más simple de planificación en disco. Este consiste en atender las solicitudes en el orden de llegada. Es un algoritmo equitativo, pero no es el más eficiente.

Ejemplo: Suponga que en un momento dado se tienen solicitudes de transferencia a las siguientes pistas: 1, 36, 16, 34, 9, 12 y que el cabezal se encontraba en la pista 11. Entonces la cantidad de pistas recorridas en las búsquedas será: 10, 35, 20, 18, 25, 3 para un total de 111.
Este algoritmo tiende a ir en forma consecutiva de un extremo a otro del disco.

2.4.1.1 SSF o SSTF *("Shortest Seek-Time First")*

Es un algoritmo muy eficiente. Consiste en atender la solicitud cuya pista esté más cerca de la actual posición de la cabeza de lectura/escritura.

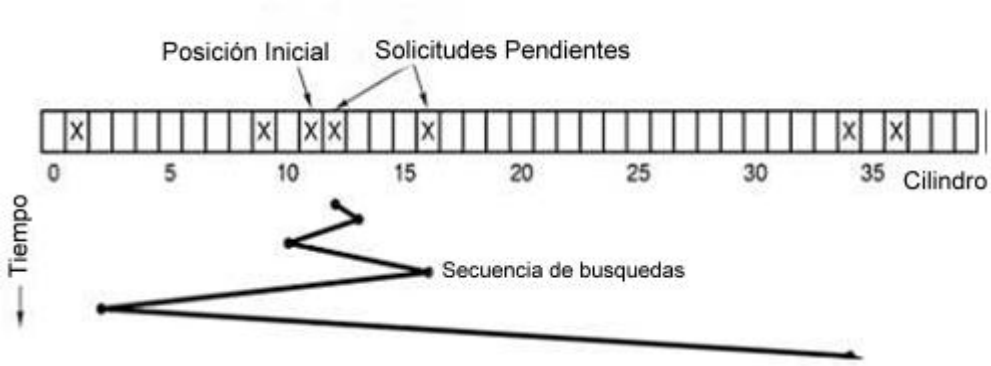


Figura 1. 1. Ejemplo de movimiento del cabezal con el algoritmo SSF

Ejemplo: con el ejemplo anterior (posición inicial del cabezal en la pista 11) se tendría la siguiente secuencia de atención: 12, 9, 16, 1, 34, 36. Entonces la cantidad de pistas recorridas en las búsquedas serán: 1, 3, 7, 15, 33, 2 para un total de 61 (Figura 1.5).

Este algoritmo tiene la desventaja que puede crear inanición (“starvation”) para las solicitudes más alejadas de la posición inicial del brazo, pues si existe un flujo continuo de nuevas solicitudes, muchas de ellas cercanas a la posición actual, las solicitudes de pistas más alejadas tardarán en atenderse.

El algoritmo del elevador. Sus variantes.

Se conoce esta política como aquella en la que el cabezal del disco recorre las pistas del disco a todo lo largo, es decir, si va en un sentido, no para hasta llegar al final y entonces es que invierte el sentido. Pero este “final” es el que varía según sea la variante que se trate.

2.4.1.2 SCAN

En esta planificación las cabezas de lectura/escritura comienzan en una de las pistas extremas (la inicial o la última) y se mueven hacia la otra, brindando servicio a todas las solicitudes que existan. Al llegar al otro extremo invierte el sentido del movimiento y se repite toda la operación.

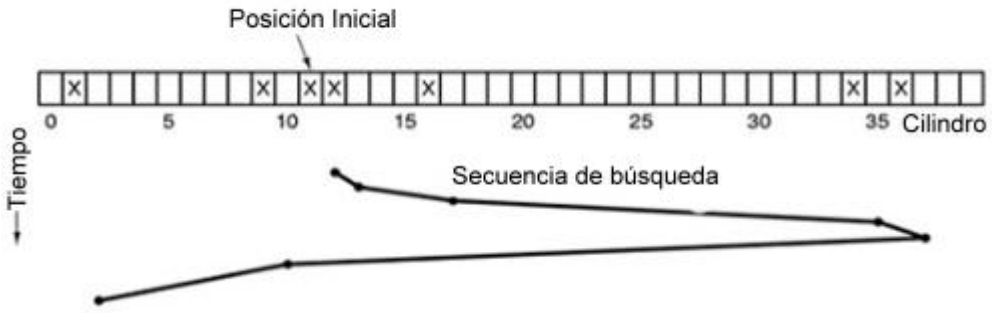


Figura 1. 2. Ejemplo de movimiento del cabezal con el algoritmo SCAN.

En la Figura 1.6 aparece representada el resultado de atención a las solicitudes aplicando esta política al caso visto con anterioridad.

Se atienden por ese orden las solicitudes: 11, 16, 34, 36, 9, 1. Se asumió que el sentido del movimiento del cabezal en el momento en que se encontraba en la pista 11 era en el sentido creciente de las pistas.

Para la instrumentación de este algoritmo se requiere tener un bit que indique el sentido del movimiento, hacia arriba (“up”) o hacia abajo (“down”).

2.4.1.3 C-SCAN

Es una variante del SCAN, se conoce como “scan” circular y siempre presta servicio en el mismo sentido, es decir, se mueve el cabezal de un extremo al otro y después se regresa al inicio para repetir la operación.
El C-SCAN parte de la lógica de que las nuevas solicitudes en el extremo inicial deben ser las que más tiempo deben llevar esperando y además de que en esta área es donde debe haber una mayor concentración de solicitudes pendientes.

En la práctica estos algoritmos no son instrumentados en esta forma, sino que comúnmente la cabeza es movida en un sentido hasta la última solicitud en dicho sentido. Tan pronto como no hay otra, se invierte el sentido del movimiento o se reinicia en la primera de las solicitudes. A estos algoritmos se conocen como LOOK y C-LOOK.

2.4.1.4 LOOK *(variante más eficiente que el SCAN)*

Si aplicamos el algoritmo LOOK a nuestro ejemplo se tendría la siguiente secuencia de atención: 12, 16, 34, 36, 9, 1. Entonces, la cantidad de pistas recorridas en las búsquedas serán: 1, 4, 18, 2, 27, 8 para un total de 60.

2.4.1.5 C-LOOK *(variante más eficiente que el C-SCAN)*

Si aplicamos el algoritmos C-LOOK a nuestro ejemplo se tendría la siguiente secuencia de atención: 12, 16, 34, 36, 1, 9. Entonces, la cantidad de pistas recorridas en las búsquedas serán: 1, 4, 18, 2, 35, 8 para un total de 68.

2.4.2 Calculo de parámetros del Disco

Otro de los puntos clave del simulador es que permita calcular los distintos parámetros del disco, como son la capacidad del disco, la dirección lógica obteniendo la dirección física y viceversa.

2.4.2.1 Capacidad del disco

Se calcula multiplicando los sectores que tiene el disco por la cifra obtenida por el tamaño del sector.

*CapacidadDisco = NúmSectores * TamañoSector*

Siendo*:*

*NúmSectores = NúmCaras * NúmCilindros * Sectores/Pista*

2.4.2.2 Calcular dirección Lógica a partir de una dirección Física

Suponemos un solo sector por bloque. Las direcciones físicas tienen la siguiente estructura:

(núm. cilindro, núm. cara, núm. sector)

Además la numeración se sigue del siguiente modo: hasta que no se han numerado todos los sectores de una cara no se pasa a la siguiente cara del mismo cilindro y hasta que no se hayan numerado todas las caras de un cilindro no se pasa al siguiente.

Para lograr esto se efectúan los siguientes pasos:

1. Se divide la dirección lógica por el número de sectores por pista.
2. Dividiremos el número de caras “completas” obtenido en el paso anterior por el número de caras por cilindro que admite el disco.

2.4.2.3 Calcular dirección Física a partir de una dirección Lógica

Suponiendo direcciones físicas de la forma (i, j, k), basta con aplicar la fórmula:

*nb = k + sec/pista * [j + caras/cilindro * (i − 1)]*

Siendo:

i = *núm. cilindro*

j = núm. cara

k = núm. sector

2.5. Descripción de las clases y funcionalidades a implementar

El Diagrama de Clases propuesto por el analista a pesar de estar ajustado a las propiedades del problema, es necesario agregarle ciertas funcionalidades a las clases con la finalidad de cumplir con todos los requerimientos de una forma simple y efectiva.

A continuación se expone el diagrama de clases con los métodos adicionados y se realiza una breve descripción de las principales clases como también de sus principales métodos.

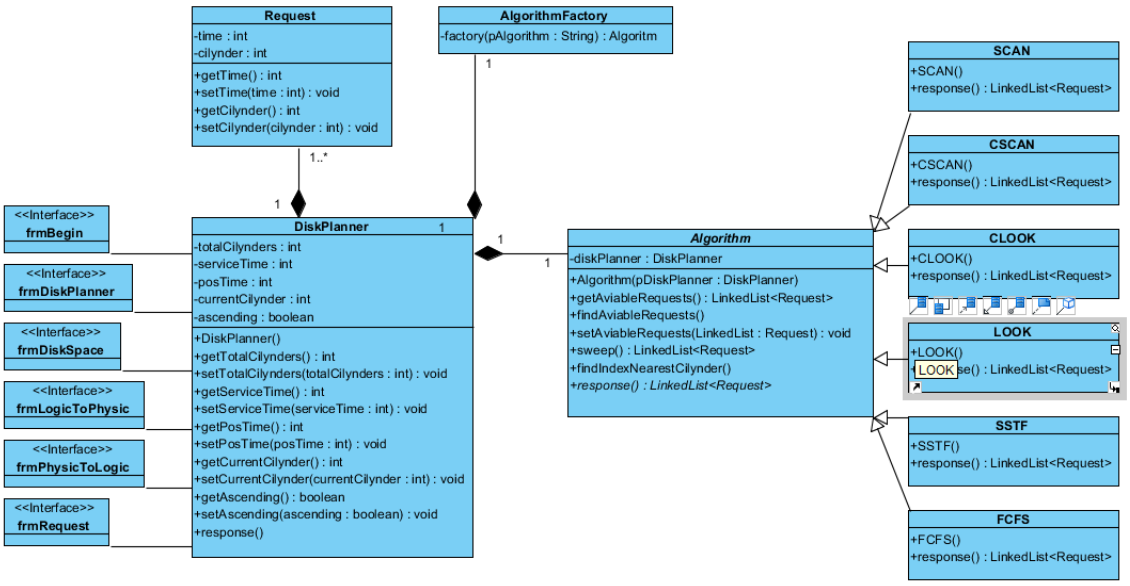
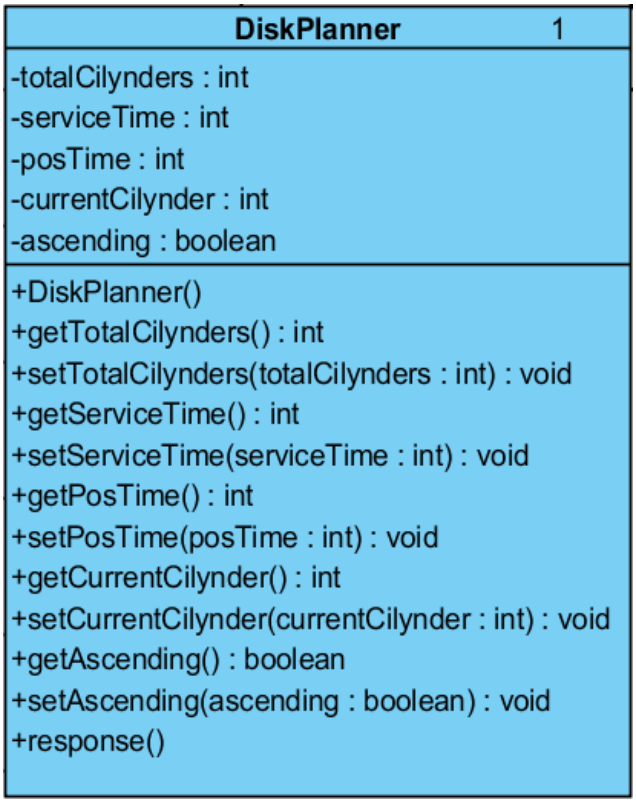
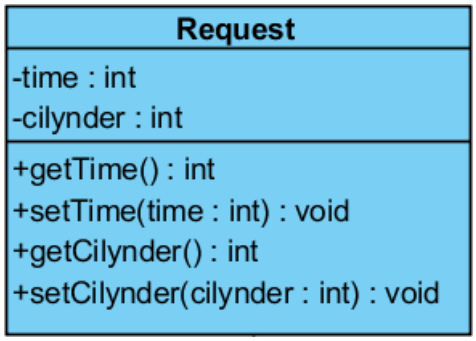


Figura 2. 4. Diagrama de clases arreglado con nuevas funcionalidades.

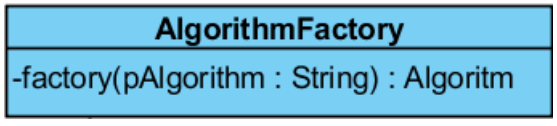
Descripción de las Principales Clases



DiskPlanner: es la clase controladora, traducida al español como "Planificador de Disco", esta clase contiene como atributos un entero *totalCilynders*(total de cilindros del disco), *intserviceTime*(Tiempo de servicio de cada solicitud), *intposTime* (Tiempo de posicionamiento, el cual no es más que el tiempo que se demora en trasladarse el brazo), *intcurrentCilynder* (Cilindro actual en el que se encuentra el brazo al ser inicializado el disco) un *boolean ascending* el cual indica en el sentido en que se mueve el brazo si ascendente o descendente. También contiene una lista enlazada (*LinkedLis*t) de Solicitudes (*requests*) y una instancia de la clase *Algorithm.* Como métodos contiene los "*get"* y *"set"* de cada uno de los atributos antes mencionados y el constructor de la clase.

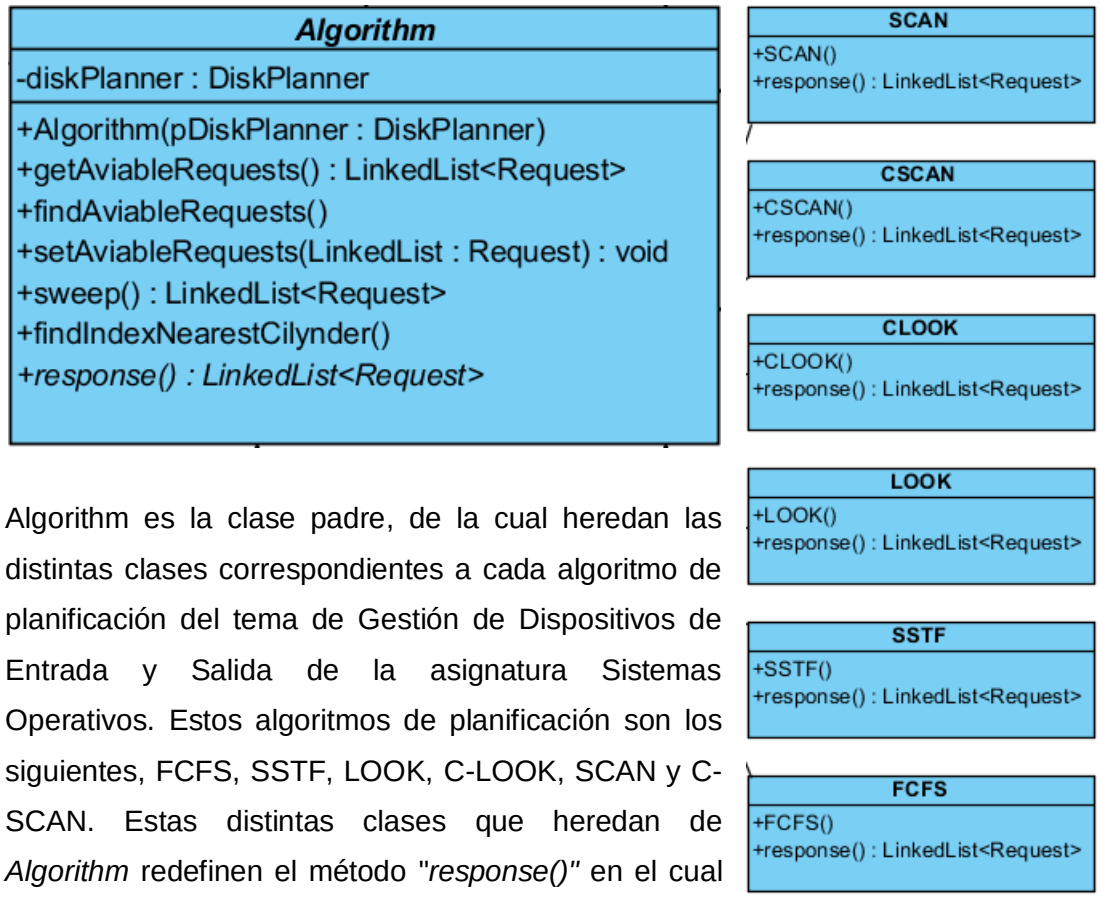


Request: es la clase "*Solicitud*", la cual contiene como atributos un entero "*Time*" como el instante de tiempo en el que la solicitud de acceso al disco entra en la lista de peticiones a servir y otro entero "*Cylinder*" que es el cilindro a servir de dicha solicitud.



AlgorithmFactory: Ésta clase se encarga de transformar el Algoritmo seleccionado en formato "*string*" obtenido del formulario y

construir su clase correspondiente quien hereda de Algorithm.



Algorithm es la clase padre, de la cual heredan las distintas clases correspondientes a cada algoritmo de planificación del tema de Gestión de Dispositivos de Entrada y Salida de la asignatura Sistemas Operativos. Estos algoritmos de planificación son los siguientes, FCFS, SSTF, LOOK, C-LOOK, SCAN y C-SCAN. Estas distintas clases que heredan de *Algorithm* redefinen el método "*response()"* en el cual se le introduce una lista "*linkedList*" de solicitudes a servir "*getRequest*" y retorna una lista con las solicitudes ya planificadas.

Sus métodos fundamentales son:
findAviableRequests() : este método se encarga de acumular en una lista las peticiones disponibles, es decir que su instante de tiempo le permita entrar en la lista de peticiones a servir. Finalmente las guarda en la lista llamada "*aviableRequests*", la cual posee sus respectivos métodos *getaviableRequests()* y *setaviableRequests().*

Otro de los métodos importantes de esta clase es el "*sweep()",* como su nombre en inglés lo describe el método esencialmente lo que realiza es un barrido por la lista "*aviableRequests"* desde un "cilindro actual" previamente seleccionado y en dependencia del sentido ascendente o descendente del brazo del disco, busca la

petición con el cilindro a servir más cercano al actual usando el método "*findIndexNearestCilynder ()*".

2.6. Estándares de codificación

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, éste 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 simulador para el tema de gestión de Entrada y Salida de la asignatura de Sistemas Operativos.

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: DiskPlanner. Este nombre está compuesto por dos palabras (Disk y Planner), la primera al igual que la segunda comienzan con mayúscula.

Camel Casing: Es parecido al Pascal Casing con la excepción que la letra inicial del identificador no debe estar en mayúscula.
Ejemplo: algorithmFactory. Este nombre está compuesto por dos palabras (Algorithm y Factory), 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.

2.7. Conclusiones parciales

Mediante el desarrollo de este capítulo se obtuvo la solución del simulador de discos para el laboratorio virtual de la asignatura de Sistemas Operativos con el objetivo de apoyar la visualización de los contenidos del tema Gestión de Entrada y Salida.

Con la revisión del diseño propuesto y de todos los artefactos generados se puede concluir que dicho diseño facilita e implanta las bases para la implementación gracias a la buena estructura y organización que posee.

Se garantizó además, a partir de la arquitectura propuesta, el intercambio de información entre los componentes existentes y se sentaron las bases para la incorporación de nuevos componentes que incluyan nuevas funcionalidades que mejoren en un futuro la herramienta.

Se realizó un análisis de los estándares de codificación lo que permitió lograr una mayor uniformidad en el código y se hizo una breve descripción de algunas de las clases más importantes del sistema que permite ver internamente las funcionalidades que estas brindan.

CAPÍTULO 3: Validación de la solución propuesta

3.1. Introducción

Durante el proceso de desarrollo de un software, los errores pueden comenzar a aparecer incluso desde el mismo momento en que fueron definidos los objetivos y estos a su vez especificados de forma errónea e imperfecta; de la misma forma en los posteriores pasos del diseño y desarrollo. A raíz de la incapacidad humana de trabajar y comunicarse de forma perfecta, el desarrollo del software debe ser acompañado de una actividad que garantice su calidad. En el desarrollo de este capítulo se valida mediante pruebas estructurales y funcionales si el software realizado ha cumplido el objetivo planteado con la calidad requerida.

3.2. Aplicación de pruebas de software

Las pruebas de software son un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación [(26)]. Están constituidas por técnicas, métodos y herramientas que contribuyen a que un programa tenga un correcto funcionamiento. Las pruebas de software pueden ser clasificadas en dependencia del tipo de prueba que se le haga a la aplicación aunque todas se ejecutan para obtener el mismo resultado, encontrar errores o defectos en la aplicación. Algunas de ellas pueden ser: Pruebas de caja blanca: Están dirigidas a probar las funciones internas, es decir, el código del sistema. Pruebas de caja negra: Están dirigidas a ejecutar los elementos pertenecientes a la interfaz de usuario para estudiar la entrada de datos y respuestas del sistema.

3.2.1. Pruebas estructurales o de caja blanca

Las llamadas pruebas de caja blanca, también conocidas como técnicas de caja transparente o de cristal, proponen una manera de diseñar los casos de prueba centrándose en el comportamiento interno y la estructura del programa. De esta manera se examina solo la lógica interna del programa, dejando fuera los aspectos de rendimiento del mismo.

El empleo de este tipo de pruebas permitirá diseñar casos de prueba que comprueben que todas las sentencias del sistema se ejecuten al menos una vez, además de todas

las condiciones, tanto verdaderas como falsas. Para esto primero se procede a enumerar las sentencias del código y a partir del mismo se construye el grafo de flujo asociado.

```
public int findIndexNearestRequest ()  {
    int result = 0; int diff = 999; int i = 0;

    for (Request request : aviableRequests) {
        if (Math.abs(diskPlanner.getCurrentCilynder() - request.getCilynder()) < diff) {
            diff = Math.abs(diskPlanner.getCurrentCilynder() - request.getCilynder());
            result = i;
        }

        i++;
    }

    return result;
}
```

Figura 3. 1. Código fuente del método findIndexNearestRequest().

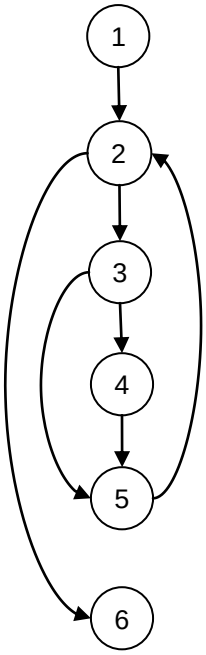


Figura 3. 2. Grafo de flujo asociado a la funcionalidad findIndexNearestRequest().

Luego de haber construido el grafo se realiza el cálculo de la complejidad ciclomática mediante las tres fórmulas descritas a continuación, las cuales deben arrojar el mismo resultado para asegurar que el cálculo de la complejidad es correcto.

1. V (G) = (A – N) + 2

Siendo “A” la cantidad total de aristas y “N” la cantidad total de nodos.

V (G) = (7 - 6) + 2

V (G) = 3

2. V (G) = P + 1

Siendo “P” la cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).

V (G) = 2 + 1

V (G) = 3

3. V (G) = R

Siendo “R” la cantidad total de regiones, para cada formula “V (G)” representa el valor del cálculo.

V (G) = 3

El cálculo efectuado mediante las tres fórmulas ha dado el mismo valor, dando como resultado 3, lo que indica que existen 3 posibles caminos por donde el flujo puede circular, y determina el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. Seguidamente es necesario representar los caminos básicos por los que puede recorrer el flujo:

1+2+6

1+2+3+5+2+6

1+2+3+4+5+2+6

Casos de prueba para el Camino básico #1:

Descripción: La lista de aviableRequest no contiene ningún elemento.

Condición de ejecución: La lista aviableRequest está vacía.

Entrada: *No recibe parámetros*

Resultados esperados: Retorna el valor 0.

Casos de prueba para el Camino básico #2:

Descripción: La lista de aviableRequest tiene al menos un elemento y el cálculo de la diferencia entre el cilindro actual y el cilindro de la petición es mayor que el cálculo anterior

Condición de ejecución: La lista de aviableRequest debe tener al menos un elemento y el cálculo de la diferencia entre el cilindro actual y el cilindro de la petición debe ser mayor que el cálculo anterior

Entrada: *No recibe parámetros*

Resultados esperados: Retorna la posición del cilindro de la primera petición.

Casos de prueba para el Camino básico #3:

Descripción: La lista de aviableRequest tiene al menos un elemento y el cálculo de la diferencia entre el cilindro actual y el cilindro de la petición es menor que el cálculo anterior

Condición de ejecución: La lista de aviableRequest debe tener al menos un elemento y el cálculo de la diferencia entre el cilindro actual y el cilindro de la petición debe ser menor que el cálculo anterior

Entrada: *No recibe parámetros*

Resultados esperados: Retorna la posición del cilindro más cercano.

Luego de aplicar los distintos casos de pruebas, se pudo comprobar que el flujo de trabajo de la función es correcto ya que cumple con las condiciones necesarias que se habían planteado.

3.2.2. Pruebas funcionales o de caja negra

Caso de Pruebas de Aceptación: Están dirigidas a verificar si el sistema cumple con las especificaciones que el cliente definió, es decir, si cumple con las historias de usuario.

Caso de Prueba de Aceptación	
Código:CPA1.1	**Historia de Usuario:** *No.1 Asignar nuevas peticiones*
Nombre: Adicionar Peticiones	
Descripción: Insertar una petición introduciendo los datos correctamente	
Condiciones de Ejecución: Ninguna	
Entrada / Pasos de ejecución: Se selecciona en el menú **Peticiones > Adicionar** Se introducen los datos correctamente Se presiona **Aceptar**	
Resultado Esperado: Se muestra la petición en la lista de solicitudes.	

Evaluación de la Prueba: Satisfactoria	
Código:CPA1.2	**Historia de Usuario:** *No.1 Asignar nuevas peticiones*
Nombre: Adicionar Peticiones	
Descripción: Insertar una petición introduciendo los datos incorrectamente, Se presiona **Aceptar**	
Condiciones de Ejecución: Ninguna	
Entrada / Pasos de ejecución: Se selecciona en el menú **Peticiones > Adicionar** Se introducen los datos incorrectamente Se presiona **Aceptar**	
Resultado Esperado: Se muestra mensaje “Valor no válido”	
Evaluación de la Prueba: Satisfactoria	
Código:CPA1.3	**Historia de Usuario:** *No.1 Asignar nuevas peticiones*
Nombre: Adicionar Peticiones	
Descripción: Insertar una petición introduciendo los datos correspondientes en el formulario y deje al menos un campo requerido en blanco. Se presiona el botón **Aceptar.**	
Condiciones de Ejecución: Ninguna	
Entrada / Pasos de ejecución: Se introducen los datos correspondientes en el formulario Deje al menos un campo requerido en blanco. Se presiona el botón **Aceptar.**	
Resultado Esperado: Se muestra mensaje “Debe rellenar todos los espacios”	
Evaluación de la Prueba: Satisfactoria	

Tabla 7. CPA 1 Insertar Solicitudes.

Caso de Prueba de Aceptación	
Código:CPA2.1	**Historia de Usuario:** *No.2 Planificar Solicitudes de acceso al disco*
Nombre: Planificar Solicitudes	
Descripción: Se intenta **Planificar** las solicitudes con elementos en la lista de solicitudes y con los datos del disco introducidos correctamente.	
Condiciones de Ejecución: Ninguna	
Entrada / Pasos de ejecución: Se selecciona en el algoritmo de planificación, se insertan los datos del disco correctamente, se tiene la lista de peticiones con al menos una solicitud y se presiona **Planificar**	
Resultado Esperado: Se muestran las peticiones planificadas en la **Lista de Solicitudes Planificadas**.	
Evaluación de la Prueba: Satisfactoria	
Código:CPA2.2	**Historia de Usuario:** *No.2 Planificar Solicitudes de acceso al disco*
Nombre: Planificar Solicitudes	
Descripción: Se intenta **Planificar** las solicitudes sin elementos en la lista de solicitudes y con los datos del disco introducidos correctamente.	
Condiciones de Ejecución: Ninguna	
Entrada / Pasos de ejecución: Se selecciona en el algoritmo de planificación, se insertan los datos del disco correctamente, se tiene la lista de peticiones vacía y se presiona **Planificar**	
Resultado Esperado: Se muestra mensaje “Introduzca alguna petición”	
Evaluación de la Prueba: Satisfactoria	

Código:CPA2.3	**Historia de Usuario:** *No.2 Planificar Solicitudes de acceso al disco*
Nombre: Planificar Solicitudes	
Descripción: Se intenta **Planificar** las solicitudes con elementos en la lista de solicitudes y con los datos del disco introducidos incorrectamente y/o en blanco.	
Condiciones de Ejecución: Ninguna	
Entrada / Pasos de ejecución: Se introducen los datos correspondientes en el formulario y deje al menos un campo requerido en blanco. Se presiona el botón **Planificar.**	
Resultado Esperado: Se muestran los mensajes “Introduzca el <campo requerido>”	
Evaluación de la Prueba: Satisfactoria	

Tabla 8. CPA 2 Planificar Solicitudes de acceso al disco.

Caso de Prueba de Aceptación	
Código:CPA3.1	**Historia de Usuario:** *No.3 Calcular capacidad del disco duro.*
Nombre: Calculad capacidad del disco duro.	
Descripción: Calcular la capacidad del disco insertando los datos correctamente y presionando **Calcular**	
Condiciones de Ejecución: Ninguna	
Entrada / Pasos de ejecución: Se introducen los datos correspondientes en el formulario. Se presiona el botón **Calcular.**	
Resultado Esperado: Se muestra el resultado del cálculo de3 la capacidad del disco	
Evaluación de la Prueba: Satisfactoria	
Código:CPA3.2	**Historia de Usuario:** *No.3 Calcular capacidad del disco duro.*

Nombre: Calculad capacidad del disco duro.	
Descripción: Calcular la capacidad del disco insertando los datos in correctamente y presionando **Calcular**.	
Condiciones de Ejecución: Ninguna	
Entrada / Pasos de ejecución: Se introducen los datos incorrectamente en el formulario. Se presiona el botón **Calcular.**	
Resultado Esperado: Se muestra mensaje “Valor no válido”	
Evaluación de la Prueba: Satisfactoria	
Código:CPA3.3	**Historia de Usuario:** *No.3 Calcular capacidad del disco duro.*
Nombre: Calculad capacidad del disco duro.	
Descripción: Calcular la capacidad del disco insertando los datos correctamente y deje al menos un campo requerido en blanco. Presionar **Calcular**.	
Condiciones de Ejecución: Ninguna	
Entrada / Pasos de ejecución: Se introducen los datos correspondientes en el formulario y deje al menos un campo requerido en blanco. Se presiona el botón **Calcular.**	
Resultado Esperado: Se muestra mensaje “Debe rellenar todos los espacios”	
Evaluación de la Prueba: Satisfactoria	

Tabla 9. CPA 3 Calcular capacidad del disco duro.

	Caso de Prueba de Aceptación
Código:CPA4.1	**Historia de Usuario:** *No.4 Calcular dirección lógica de un sector físico.*
Nombre: Calcular dirección lógica de un sector físico.	

Descripción: Calcular dirección lógica de un sector físico insertando los datos correctamente y presionando **Calcular**	
Condiciones de Ejecución: Ninguna	
Entrada / Pasos de ejecución: Se introducen los datos correspondientes en el formulario. Se presiona el botón **Calcular.**	
Resultado Esperado: Se muestra el cálculo de la dirección lógica del sector físico	
Evaluación de la Prueba: Satisfactoria	
Código:CPA4.2	**Historia de Usuario:** *No.4 Calcular dirección lógica de un sector físico.*
Nombre: Calcular dirección lógica de un sector físico.	
Descripción: Calcular dirección lógica de un sector físico insertando los datos incorrectamente y presionando **Calcular**.	
Condiciones de Ejecución: Ninguna	
Entrada / Pasos de ejecución: Se introducen los datos incorrectamente en el formulario. Se presiona el botón **Calcular.**	
Resultado Esperado: Se muestra mensaje “Valor no válido”	
Evaluación de la Prueba: Satisfactoria	
Código:CPA4.3	**Historia de Usuario:** *No.4 Calcular dirección lógica de un sector físico.*
Nombre: Calcular dirección lógica de un sector físico.	
Descripción: Calcular dirección lógica de un sector físico insertando los datos correctamente y deje al menos un campo requerido en blanco. Presionar **Calcular**.	
Condiciones de Ejecución: Ninguna	

Entrada / Pasos de ejecución: Se introducen los datos correspondientes en el formulario y deje al menos un campo requerido en blanco. Se presiona el botón **Calcular.**
Resultado Esperado: Se muestra mensaje “Valor Inválido”
Evaluación de la Prueba: Satisfactoria

Tabla 10. CPA 4 Calcular dirección lógica de un sector físico.

Caso de Prueba de Aceptación	
Código:CPA5.1	**Historia de Usuario:** *No.5 Calcular dirección física de un sector lógico.*
Nombre: Calcular dirección física de un sector lógico.	
Descripción: Calcular dirección física de un sector lógico insertando los datos correctamente y presionando **Calcular**	
Condiciones de Ejecución: Ninguna	
Entrada / Pasos de ejecución: Se introducen los datos correspondientes en el formulario. Se presiona el botón **Calcular.**	
Resultado Esperado: Se muestra el cálculo de la dirección física del sector lógico	
Evaluación de la Prueba: Satisfactoria	
Código:CPA5.2	**Historia de Usuario:** *No.5 Calcular dirección física de un sector lógico.*
Nombre: Calcular dirección física de un sector lógico.	
Descripción: Calcular dirección física de un sector lógico insertando los datos incorrectamente y presionando **Calcular**.	
Condiciones de Ejecución: Ninguna	

Entrada / Pasos de ejecución: Se introducen los datos incorrectamente en el formulario. Se presiona el botón **Calcular.**	
Resultado Esperado: Se muestra mensaje “Valor no válido”	
Evaluación de la Prueba: Satisfactoria	
Código:CPA5.3	**Historia de Usuario:** *No.5 Calcular dirección física de un sector lógico.*
Nombre: Calcular dirección física de un sector lógico.	
Descripción: Calcular dirección física de un sector lógico insertando los datos correctamente y deje al menos un campo requerido en blanco. Presionar **Calcular**.	
Condiciones de Ejecución: Ninguna	
Entrada / Pasos de ejecución: Se introducen los datos correspondientes en el formulario y deje al menos un campo requerido en blanco. Se presiona el botón **Calcular.**	
Resultado Esperado: Se muestra mensaje “Valor Inválido”	
Evaluación de la Prueba: Satisfactoria	

Tabla 11. CPA 5 Calcular dirección física de un sector lógico.

De esta forma se le aplicaron las pruebas de caja negra a todas las historias de usuario (funcionalidades del sistema). A continuación se resumen la cantidad de escenarios de prueba utilizados para cada funcionalidad.

- ✓ Adicionar Solicitud: 3 escenarios.
- ✓ Planificar Solicitudes: 3 escenarios.
- ✓ Calcular capacidad del disco: 3 escenarios.
- ✓ Calcular dirección lógica de un sector físico: 3 escenarios.
- ✓ Calcular dirección física de un sector lógico: 3 escenarios.

Las pruebas funcionales o de caja negra fueron realizadas por el Grupo de Calidad Centro CEGEL de la Facultad 3, no encontrando ninguna “no conformidad” en la aplicación y como anexo se muestra el acta de liberación de artefactos entregada por el Asesor y Jefe del Grupo el Ing. Raúl Velázquez Álvarez.

3.3. Conclusiones parciales

En este capítulo se realizaron pruebas estructurales o de caja blanca para validar el correcto funcionamiento del código fuente utilizando el método del camino básico ofreciendo resultados satisfactorios.

También se realizaron las pruebas funcionales o de caja negra donde se definen los casos de prueba de aceptación para cada historia de usuario, asegurando así que cada funcionalidad del sistema quede bajo prueba. Cada uno de estos casos contiene distintos escenarios donde se definen los posibles valores (válidos e inválidos) que el usuario puede insertar en el sistema, se verifica que los resultados esperados ocurran y que sean desplegados los mensajes de error apropiados cuando se insertan datos inválidos. Dichas pruebas arrojaron los resultados esperados y se le encontró libre de defecto como lo indica el acta de liberación anexada al documento.

Conclusiones

- ✓ Mediante el estudio de las funcionalidades de los diferentes laboratorios virtuales y simuladores en otros sistemas en el mundo, se contribuyó a elevar el nivel de comprensión de los desarrolladores y a determinar los objetivos y funcionalidades necesarias para el sistema a construir.
- ✓ Se investigó sobre las posibles herramientas y metodologías para el desarrollo de la aplicación seleccionándose el lenguaje de desarrollo Java con el IDE de desarrollo NetBeans 6.9, y como metodología de desarrollo se seleccionó XP, el lenguaje de modelado seleccionado fue UML 2.0 junto con la herramienta Visual Paradigm 5 para la valorización y modificación de los diagramas propuestos por la analista.
- ✓ Se realizó la implementación del producto Simulador del tema de Gestión de dispositivos de Entrada y Salida para el Laboratorio Virtual de la Asignatura Sistemas Operativos, obteniendo la documentación necesaria para garantizar el futuro mantenimiento del sistema.
- ✓ Se obtuvo un producto interactivo que cumple con los requisitos establecidos.
- ✓ Con la realización de esta aplicación, en la Universidad de las Ciencias Informáticas aumentan los materiales interactivos que sirven de apoyo para impartir los temas de Gestión de dispositivos de Entrada y Salida en la asignatura Sistemas Operativos.
- ✓ Se cumplió el objetivo general trazado para este Trabajo de Diploma: desarrollar un simulador que apoye la visualización de los contenidos del tema Gestión de dispositivos de Entrada y Salida en la asignatura Sistema Operativo.

Recomendaciones

Tomando en cuenta los resultados obtenidos:

- Se recomienda incluir funcionalidades que permitan guardar en ficheros los ejercicios que se simulan y su posterior recuperación.
- Implementar las funcionalidades restantes para la versión 2.0.
    - Algoritmos de planificación de solicitudes (SCAN, C-SCAN)
    - Que la interfaz de planificación muestre gráficamente el movimiento del brazo en cada solicitud planificada.

Bibliografía referenciada

1. **Herreros, L. Rosado y J.R.** Nuevas aportaciones didácticas de los laboratorios virtuales y remotos en la enseñanza de la Física. [En línea] 2005. [Citado el: 14 de 4 de 2011.] http://www.formatex.org/micte2005/286.pdf.

2. **Vary, James P.** Informe de la reunión de expertos. *Informe de la reunión de expertos.* [En línea] 2000. [Citado el: 1 de 4 de 2011.] http://unesdoc.unesco.org/images/0011/001191/119102s.pdf.

3. **Estrada, Julián Monge Nájera y Víctor Hugo Méndez.** 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] 2009. [Citado el: 13 de 2 de 2011.] http://www.latindex.ucr.ac.cr/edu31-1/edu-31-1-05.pdf.

4. **Herreros, L. Rosado y J.R.***Laboratorios virtuales y remotos en la enseñanza de la Física y materias afines, Didáctica de la Física y sus nuevas Tendencias.* Madrid : UNED, 2002. págs. pp. 415-603.

5. **Herreros, L. Rosado y J.R.***Internet y Multimedia en Didáctica e Investigación de la Física. Tratado teórico - práctico para profesores y doctorandos.* Madrid : UNED, 2004.

6. **Herías, Francisco Andrés Candelas.** Propuesta de Portal de la Red de Laboratorios Virtuales y Remotos de CEA. *Propuesta de Portal de la Red de Laboratorios Virtuales y Remotos de CEA.* [En línea] 27 de 11 de 2003. [Citado el: 2 de 16 de 2011.] http://www.disc.ua.es/docenweb/Docs/PropuestaDePortal.pdf.

7. **Cárdenas, Marina Elizabeth.***Software de Simulación aplicado a entornos de e-learning.* 2009.

8. **Schwaber K., Beedle M., Martin R.C.***"Agile Software Development with.* s.l. : Prentice Hall, 2001.

9. **Cockbun, A.***"Agile Software Development".* s.l. : Addison-Wesley, 2001.

10. **Letelier, Patricio; Penadés, Mª Carmen.***Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* Valencia : Universidad Politécnica de Valencia, 2003.

11. **Beck, K.***Extreme Programming Explained. Embrace Change.* s.l. : Pearson Education, 1999. Traducido al español como: "Una explicación de la programación extrema. Aceptar el cambio", Addison Wesley, 2000.

12. **Highsmith, J.***"Agile Software Development Ecosystems".* s.l. : Addison-Wesley, 2002.

13. **Barriento, Manuel Sánchez.** BPMN desventajas . BPMN desventajas. [En línea] 2 de 11 de 2008. [Citado el: 16 de 4 de 2011.] http://www.aprendergratis.com/stag/bpmn-desventajas.html.

14. **Orallo, Enrique Hernández.** El lenguaje Unificado de Modelado (UML). *El lenguaje Unificado de Modelado (UML).* [En línea] 2007. [Citado el: 3 de 2 de 2011.] www.disca.upv.es/enheror/pdf/ActaUML.

15. **Jie Zhao, Dunstan Thomas.** Comparación de Herramientas de modelado UML: Enterprise Architect y Rational Rose. Apexnet. [En línea] 15 de 6 de 2005. [Citado el: 17 de 5 de 2011.] http://www.apexnet.com.ar/index.php/news/main/38/event=view.

16. **Giraldo, L. &.***Herramientas de Desarrollo de Ingeniería de SW para Linux.* 2005.

17. **Almaguer, Anileidy Basso y Mesa, Dayaima Gómez.***Sistema automatizado para el proceso de caracterización de los estudiantes.* Habana. : s.n., 2009.

18. **Borges, Rasiel Aponcio.***Migración de la Capa de Acceso a Datos de Ciudad de la Habana.* 2009.

19. **Pelegrino, Rodolfo González.***Aplicación para el monitoreo de ventanas y notificación de tareas.* s.l. : Ciudad de la Habana, 2009.

20. **Gonzalez, Jesús Alberto Silva.** Tarea de Java. [En línea] 2009. [Citado el: 2 de 1 de 2011.]

21. **JetBrains.** www.intellij.com. *IntelliJ IDEA :: Best Java IDE to do more high-quality code in less time.* [En línea] 2010. http://www.jetbrains.com/idea/.

22. **Object Technology International, Inc.** Eclipse Platform Technical Overview. *www.Eclipse.org.* [En línea] 2003. [Citado el: 12 de 5 de 2011.] www.eclipse.org/whitepapers/eclipse-overview.pdf.

23. **Tim Boudreau, Jesse Glick, Simeon Greene, Vaughn Spurlin, Jack Woehr.**NetBeans IDE 6.9 Release Information. *http://netbeans.org.* [En línea] 2010. [Citado el: 22 de 4 de 2011.]

24. **Allison, Yanelis Valdivieso.** Análisis del Módulo de Gestión de Entrada y Salida para el Laboratorio Virtual de la asignatura Sistemas Operativos. [En línea] 2010.

25. **Jeffries, R., Anderson y A., Hendrickson, C.***"Extreme Programming Installed".* s.l. : Addison-Wesley, 2001.

26. **Mozilla Developer Center.** Concepto de JavaScript. [En línea] 5 de 2 de 2009. [Citado el: 11 de 1 de 2011.] https://developer.mozilla.org/es/Gu%C3%ADa_JavaScript_1.5/Concepto_de_JavaScript.

Bibliografía consultada

Express., Ediciones Visual Studio 2005 Standard y Visual Studio 2005. Ediciones Visual Studio 2005 Standard y Visual Studio 2005 Express. [En línea] [Citado el: 20 de 1 de 2011.] http://www.microsoft.com/spanish/msdn/vs2005/editions/stdexp/default.mspx.

Gutiérrez, José Manuel Ruiz. 2000. La Simulación como Instrumento de Aprendizaje. . *La Simulación como Instrumento de Aprendizaje.* [En línea] 2000. [Citado el: 23 de 3 de 2011.]
mami.uclm.es/jmruiz/materiales/Documentos/simulacion.PDF

Piña., Lianyi Ramos León y Yanelis Pulido.*Análisis de los módulos Planificación de Disco y Administración de Memoria de un Laboratorio Virtual de apoyo a la asignatura de Sistemas Operativos.* Habana, Cuba : s.n., 2008.

Pressman, R. S.*Ingeniería de software. Un enfoque práctico. Parte ll. .* Ciudad de la Habana : Félix Varela, 2005.

Pressman, R.*Ingeniería del Software. Un enfoque práctico.* La Habana : Félix Varela, 2005.

Rivas, Lornel A. Herramientas de Desarrollo de Software: Hacia la Construcción de una Ontología. *Herramientas de Desarrollo de Software: Hacia la Construcción de una Ontología.* [En línea] [Citado el: 2 de 3 de 2011.] http://www.lisi.usb.ve/publicaciones/05%20herramientas/herramientas_25.pdf

(ME), Visual Paradigm for UML. [En línea] 5 de 3 de 2007. [Citado el: 22 de 2 de 2011.]
http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/..

Java, Foro de.*Foro de Java.* [En línea] 13 de 6 de 2008. [Citado el: 18 de 2 de 2011.] http://www.forodejava.com/showthread

Cáceres, Janet Carreño.*ANÁLISIS DEL SISTEMA "COLISEO VIRTUAL".* Cuidad de la Habana: s.n., 2007.

Oca, E. C.*Análisis y Diseño de una herramienta interactiva de simulación de procesos: Nodo Virtual de Procesos. Segunda Iteración.* La Habana : s.n., 2009.

Ventajas y desventajas de usar laboratorios Virtuales en educación a distancia: la opinión del estudiantado en un proyecto de seis años de duración.*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] 2007. [Citado el: 16 de 2 de 2011.] http://www.latindex.ucr.ac.cr/edu31-1/edu-31-1-05.pdf.

Rodriguez, Misael Urrutia y Figueredo Jiménez, Lianni Yadira.*Simulador para la asignatura Sistemas Operativos.* Habana, Cuba : UCI, 2010.

Ramos, Lianyi León y Pulido, Yanelis Piña.*Análisis de los módulos Planificación de Disco y Administración de Memoria de un Laboratorio Virtual de apoyo a la asignatura de Sistemas Operativos.* Habana, Cuba : UCI, 2008.

Asignatura, Operativos Sistemas. Conferencia 10. Administración de Entrada y Salida. *Algoritmos de Planificación de Disco.* Habana, Cuba : UCI,2008.

Asignatura, Operativos Sistemas. Clase Práctica 11: Gestión de Entrada Salida. *Clase Práctica 11: Gestión de Entrada Salida.* Habana, Cuba : UCI,2008.

L. Gil, E. Blanco y J.M. Aulí.*Software educativo orientado a la experimentación, I Congreso Internacional de Docencia Universitaria e Innovación.* Barcelona : Universidad de Barcelona, 2000.

O. Boix, S. Fillet y J. Bergas. Nuevas posibilidades en laboratorios remotos de enseñanzas técnicas. Congreso Virtual CIVE 2002. [En línea] 2002. http://www.cibereduca.com/cive/ponencias.

Steve Holzner, PhD. *Design Patterns for Dummies.* Hoboken, NJ: Wiley Publishing, Inc. 2006. 978-0-471-79854-5.

Anexos

UCi
Universidad de las Ciencias Informáticas

Acta de Liberación de Artefactos, Grupo de Calidad Centro CEGEL de la Facultad 3 de la Universidad de las Ciencias Informáticas.

Lunes, 20 de junio de 2011.

Luego de haber efectuado las revisiones al Simulador de Gestión de Dispositivos de Entrada y Salida del Laboratorio Virtual de Sistemas Operativos se concluye que está libre de defectos, por lo que queda liberada la aplicación.

Firma del Asesor y Jefe del Grupo de Calidad Centro CEGEL

Ing. Raúl Velázquez Alvarez

Glosario

Disco Duro

Es un dispositivo de almacenamiento no volátil en donde se guarda información.

Brazo / Cabezal

Se usa para leer y/o escribir datos en los cilindros

Platos / Cilindros

Es donde se graban los datos

Solicitud / Petición

Es la acción para solicitar el movimiento del brazo hacia cierto cilindro para la entrada o salida de información.

Software

Es la suma total de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de cómputo y que “un producto de software es un producto diseñado para un usuario”.

Simulador

Es un aparato que permite la simulación de un sistema, reproduciendo su comportamiento. Los simuladores reproducen sensaciones que en realidad no están sucediendo.

Sistema Operativo

Programa o conjunto de programas que efectúan la gestión de los procesos básicos de un sistema informático, y permite la normal ejecución del resto de las operaciones.