UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES
ARAGÓN

“DESARROLLO DE UN SISTEMA DE PROPUESTAS PARA PROFESORES DE INGENIERÍA EN COMPUTACIÓN”

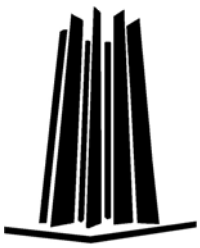
T E S I S

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN

P R E S E N T A:

EDGAR RICARDO MORALES CONTRERAS

ASESOR DE TESIS:
M. EN C. MARCELO PÉREZ MEDEL



SAN JUAN DE ARAGÓN, EDO DE MEX.
MÉXICO, 2007.

"Enseñar no es una función vital, porque no tienen el fin en sí misma; la función vital es aprender"

"La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica"

"Lo que tenemos que aprender lo aprendemos haciendo"

Aristóteles.

AGRADECIMIENTOS

A mi madre quien me enseñó, con su ejemplo, a superarme día con día sin importar la adversidad y que en todo momento recibí su apoyo.

A mis hermanos por apoyarme a lo largo de todos mis estudios.

A todos mis amigos de la carrera de Ingeniería en Computación y del Centro Tecnológico Aragón de quienes aprendí muchas cosas al convivir con ellos.

Al M. en C. Marcelo Pérez Medel por brindarme su amistad.

Y en general a todos porque la identidad de una persona es forjada por la gente que la rodea.

Edgar Ricardo Morales Contreras

ÍNDICE

ÍNDICE

INTRODUCCIÓN

INTRODUCCIÓN

En la actualidad la elaboración de tareas de forma manual consume mucho tiempo de las personas que las realizan. La tendencia en los últimos años es la de automatizar aquellos trabajos que son repetitivos para que las personas ocupen el tiempo de una manera más eficiente.

La jefatura de carrera de Ingeniería en Computación es la encargada de realizar tareas como programar las materias y los profesores que las impartirán, crear horarios, controlar y programar el uso de los salones, crear el banco de horas, la elaboración de propuestas para profesores, etc. La elaboración de propuestas es una labor repetitiva y que consume mucho tiempo, por lo cual se desarrolló un software que permitiera realizar la elaboración de propuestas para profesores de forma automatizada y así reducir el tiempo de su elaboración.

El primer capítulo trata las distintas tareas administrativas que se realizan en la jefatura de carrera de Ingeniería en Computación, como son la programación de materias que se impartirán en el semestre con los respectivos profesores asignados, los horarios y la administración de salones que se utilizarán, programar la realización de exámenes extraordinarios, creación de un banco de horas, así como la elaboración de propuestas a los profesores de la carrera. Además de lo anterior, se habla sobre la situación actual en la forma de elaborar las propuestas para profesores.

El segundo capítulo trata los distintos modelos de desarrollo de software que se están utilizando actualmente, mostrando las ventajas y desventajas que tiene cada uno de ellos. También se habla sobre las metodologías ágiles para el desarrollo de software y se pone una mayor atención sobre una metodología en especial, que es la programación extrema, la cual ha tenido mucha difusión.

El capítulo tercero habla sobre las herramientas que se utilizaron para el desarrollo de la aplicación. Las principales herramientas utilizadas son la plataforma .NET y SQL Server 2000. La plataforma .NET es el entorno de desarrollo donde se programa el sistema estudiando cada uno de sus componentes claves y las ventajas que tienen sobre otras plataformas o lenguajes de programación. El SQL Server 2000 es el sistema gestor de bases de datos que se utiliza por lo cual también se estudian sus componentes.

El último capítulo, que es el cuarto, muestra el desarrollo de la aplicación desde la captura de requerimientos por medio de casos de uso, hasta la planeación a través de la estimación de tiempos y establecer la prioridad de cada caso de uso, el diseño de la arquitectura de la aplicación, el diseño de la interfaz, de la base de datos y la lógica de la aplicación, además de las pruebas.

Al final del presente trabajo se presentan las conclusiones a las que se llegaron.

CAPÍTULO 1

ANTECEDENTES E IDENTIFICACIÓN DEL PROBLEMA

CAPÍTULO 1 – ANTECEDENTES E IDENTIFICACIÓN DEL PROBLEMA

La elaboración de propuestas para profesores de la carrera de Ingeniería en Computación en la Facultad de Estudios Superiores Aragón (FES Aragón), es una de las muchas tareas, procesos o actividades que realiza la jefatura de carrera.

Las tareas que realiza la jefatura de carrera son las siguientes:

- Programar las materias y los profesores que las impartirán en el semestre.
- Crear los horarios del semestre.
- Controlar y programar el uso de los salones.
- Programar las fechas y horarios de los exámenes extraordinarios.
- Crear el banco de horas.
- Elaborar de propuestas para profesores.

A continuación se describe cada una de ellas:

Programar las materias y los profesores que las impartirán en el semestre. Aquí la jefatura de carrera decidirá las materias que se impartirán en el semestre próximo, ya que la decisión de las materias que se impartirán depende de la demanda que exista, porque las materias que regularmente se imparten al inicio de un ciclo escolar (las materias en semestres impares), en el siguiente semestre (en semestres par) la demanda es poca y no tiene caso tener varios grupos.

Crear los horarios del semestre. En este proceso la jefatura de carrera programa los horarios de las materias que se impartirán en el semestre. Los horarios se programan en dependencia de cómo se acomoden los grupos que se abren para una materia y la disponibilidad de profesores.

Controlar y programar el uso de los salones. Con base en los horarios, se programan los salones que están asignados a la carrera para la impartición de las materias tomando en cuenta el cupo del salón, debido a que existen salones que tienen cupos para 60 y 30 alumnos.

Programar las fechas y horarios de los exámenes extraordinarios. En esta tarea se programan las fechas, horarios, salones y profesores que aplicarán los exámenes extraordinarios.

Crear el banco de horas. El banco de horas es el total de horas asignadas a la jefatura de carrera para que las distribuya de acuerdo a sus necesidades. Para llenar el banco de horas la jefatura de carrera suma todas las horas que laborarán los profesores de la carrera,

profesores de asignatura, ayudantes de profesor, técnicos académicos, funcionarios y el personal por honorarios.

Elaborar propuestas para profesores. La elaboración de propuestas es un proceso que se hace para saber las materias que imparte un profesor, así como el horario y el número de horas totales que labora.

Además la jefatura de carrera está al pendiente de las inscripciones de los alumnos de Ingeniería en Computación, que generalmente se llevan a cabo una o dos semanas antes del inicio de clases, así como el periodo de altas y bajas.

Todas las actividades que se mencionaron anteriormente se realizan cada semestre. Algunas antes del inicio de clases y otras en las primeras semanas.

1.1 - PROCESO ACTUAL DE ELABORACIÓN DE PROPUESTAS

En el periodo intersemestral la jefatura de carrera de Ingeniería en Computación tiene que realizar una serie de tareas para elaborar las propuestas de profesores.

La primera actividad a realizar por la jefatura de carrera es programar y capturar los horarios del próximo semestre.

Una vez creados los horarios, se procede a generar el banco de horas, que contiene la suma de las horas de todos los profesores que imparten las materias asignadas en los horarios.

Terminado el banco de horas se procede a realizar las propuestas a todos los profesores. Las propuestas contienen los datos acerca de las materias que imparte cada profesor, el número de horas y horario asignado.

Cada una de las propuestas se imprime y es firmada por el profesor correspondiente y el jefe de carrera para posteriormente ser enviada para revisión y autorizada por el departamento de personal de la FES Aragón, que es el encargado de realizar esa tarea.

En caso de que exista algún error en la propuesta se corrige y se vuelve a imprimir y firmar para que posteriormente vuelva a ser revisada y aprobada por el departamento de personal.

1.2 - IDENTIFICACIÓN DEL PROBLEMA

Las actividades de la jefatura de carrera de Ingeniería en Computación involucran un conjunto de procesos administrativos que consumen una gran cantidad de tiempo para llevarlos a cabo, lo cual implica que en muchas ocasiones no se llegue a entregar a tiempo la información con la precisión necesaria, además de que llevan un gran trabajo repetitivo.

Uno de los mayores problemas que enfrenta la jefatura de carrera de Ingeniería en Computación, es la sobrecarga de trabajo que se presenta al inicio y al final de cada semestre, debido a que la mayoría de las veces capturan toda la información necesaria para realizar las diferentes actividades que les corresponden, además de que se llegue a capturar la misma información varias veces, ya que existen diferentes formatos para llevar el control de los mismos datos.

El tiempo que le toma a la jefatura de carrera en realizar estos procesos depende mucho de la habilidad que tienen las personas para capturar la información.

CAPÍTULO 2

METODOLOGÍA

CAPÍTULO 2 – METODOLOGÍA

2.1 – EL CICLO DE VIDA DE DESARROLLO DE UN SISTEMA

El ciclo de vida de desarrollo de un sistema son las etapas o fases por las que llega a pasar desde su concepción hasta su funcionamiento. Aunque hay algunos autores que dividen las etapas de diferente manera. Aquí se presentan de la siguiente manera [21]:

- Identificación de problemas
- Determinación de los requisitos
- Análisis del sistema
- Diseño del sistema
- Desarrollo del sistema
- Pruebas del sistema
- Implementación del sistema
- Mantenimiento del sistema

2.1.1 - Identificación de problemas

La primera etapa del ciclo de vida de desarrollo de un sistema es la identificación de una o más necesidades o los problemas que puede tener una empresa o una institución. Dentro de las necesidades o problemas que pueden existir sería el mejorar un proceso existente que quizás sea demasiado lento y que se quiera acelerar, o en el caso de que se este utilizando un sistema que con el paso de tiempo se ha vuelto obsoleto, ya no cumple con las nuevas necesidades de la empresa y que modificarlo sería más costoso.

2.1.2 - Determinación de los requisitos

En esta etapa se deben de comprender y determinar los requisitos que va a tener el sistema que se va a desarrollar. Aquí se comprende la información que necesita el usuario para desarrollar sus actividades. Además de que se conocer y comprende el entorno donde va a trabajar el nuevo sistema, quién lo va a utilizar y dónde se va a utilizar. Para determinar los requisitos del sistema se pueden utilizar diferentes métodos como las entrevistas, la investigación de datos impresos, cuestionarios, observaciones del comportamiento y tareas del usuario, el desarrollo de prototipos, etc.

2.1.3 - Análisis del sistema

Aquí se determina lo que va a hacer el nuevo sistema. Se determinan las entradas, los procedimientos para procesar las entradas y las salidas que debe de generar. Todo esto

generalmente se logra utilizando diagramas y analizando los requisitos previamente determinados en la etapa anterior. Por medio de los diagramas se pueden saber las condiciones, las acciones y las reglas de acción para procesar la información necesaria y así, generar la salida que se espera. En esta etapa el analista puede recomendar modificar un proceso, que generalmente se hacia para mejorarlo. Al final de esta etapa se debe de entregar una propuesta con la solución o soluciones recomendadas para que la empresa decida cual va a elegir para satisfacer sus necesidades.

2.1.4 - Diseño del sistema

En la etapa de diseño se utiliza toda la información que se ha reunido en las etapas anteriores para diseñar cómo va a trabajar el sistema. Se hace un diseño lógico del sistema donde se especifica cómo se van a capturar los datos de entrada, se diseñan las interfases de usuario, la base de datos para guardar información y se diseña la salida, que puede ser mostrada sólo en pantalla, en algún reporte o en ambos. El proceso de diseño establece una arquitectura completa del sistema. El diseño del sistema identifica y describe las abstracciones fundamentales del sistema de software y sus relaciones. El diseño del software es realmente un proceso de muchos pasos que se centra en cuatro atributos distintos de un sistema: estructura de datos, arquitectura del sistema, representaciones de interfaz y detalle procedimental.

2.1.5 - Desarrollo del sistema

En esta fase, todo el diseño hecho anteriormente es codificado por los programadores a un lenguaje de programación elegido. Si se lleva a acabo el diseño de una forma detallada, la generación de código se realiza mecánicamente. Hay ocasiones en que los desarrolladores ocupan funcionalidades de algún software comercial o le hacen modificaciones para no tener que desarrollar todo partiendo de cero. La decisión que se toma de utilizar un software comercial, depende de los costos del software y del tiempo que se tenga estimado para el desarrollo del sistema completo.

2.1.6 - Pruebas del Sistema

Las pruebas se dividen en dos tipos: pruebas de unidades y pruebas de integración. La prueba de unidades verifican que cada unidad o programa cumpla su especificación. En las pruebas de integración los programas o las unidades individuales de programas se integran y se prueban como un sistema completo para asegurar que se cumplan los requerimientos del software. En esta fase puede permitirse también a un grupo limitado de usuarios que utilice el sistema, de manera que los analistas puedan captar si tratan de utilizarlo en formas no planeadas ya que es preferible detectar cualquier anomalía antes de que la empresa ponga en marcha el sistema y dependa de él.

2.1.7 - Implementación del sistema

Después de haber hecho las pruebas necesarias al sistema y no encontrar más errores (lo cual no garantiza que no los haya), se procede a instalar la nueva aplicación, verificando que se cuente con el hardware necesario o en caso contrario comprar hardware nuevo. La instalación del sistema se puede realizar por medio de una conversión directa (el sistema antiguo se deja de usar al mismo tiempo y se comienza a usar el nuevo), una conversión en paralelo (se sigue utilizando el sistema antiguo además de empezar a utilizar el sistema nuevo al mismo tiempo) o conversión en fases (el sistema nuevo se empieza a utilizar por módulos o componentes). Además en esta fase se capacitan a los usuarios del sistema para que puedan manejarlo de una manera correcta.

2.1.8 – Mantenimiento del sistema

Debido al cambio de requerimientos que tiene una empresa con el paso del tiempo, el software tiene que ser modificado para añadir nuevas funcionalidades al sistema que en un principio no eran necesarias o no se tenían contempladas. El mantenimiento también implica corregir errores no descubiertos en las etapas anteriores del ciclo de vida. Por lo general, ésta es la fase más grande del ciclo de vida.

2.2 – MODELOS DE PROCESO DE DESARROLLO

A lo largo de la historia del desarrollo de software, han surgido una variedad de procesos de desarrollo de software que permitieron llevar un orden para el desarrollo de sistemas. Entre los procesos que se han utilizado se tienen:

- Modelo de codificar y corregir
- Modelo en cascada
- Modelo de prototipos
- Modelo incremental
- Modelo en espiral

Estos modelos serán explicados en las siguientes secciones para poder observar su aplicación, ventajas y desventajas.

2.2.1 – Modelo de codificar y corregir

El modelo de codificar y corregir es una forma muy común de desarrollo de software. Cuando no se ha elegido un modelo de desarrollo este es el modelo en el que seguramente se estará trabajando.

En el modelo de codificar y corregir no existe un proceso bien definido a seguir. Cuando los desarrolladores empiezan un sistema no tienen una idea detallada de lo que van a hacer

ni como lo van a hacer, es decir, tienen una idea general de lo que se necesita construir. Se puede tener una especificación formal, o no tenerla. Entonces se utiliza cualquier combinación de diseño, código, depuración y métodos de prueba no formales que sirven hasta que se tiene el producto listo para entregarlo [26].

Generalmente este modelo se utiliza en proyectos pequeños y cuando los tiempos de entrega son en muy poco tiempo. Cuando es utilizado para proyectos medianos o grandes casi seguramente el proyecto fracasará. Por eso se recomienda principalmente en la elaboración de prototipos.

2.2.1.1 – Ventajas

- No se necesita llevar ninguna gestión (planificación, documentación, control de calidad, etc.).
- Requiere poca experiencia con lo cual cualquiera puede utilizarlo.
- Es bueno para proyectos pequeños (creación de prototipos).

2.2.1.2 – Desventajas

- No se puede hacer una evaluación de la calidad o identificación de riesgos.
- Si empieza a codificar con un mal diseño desde un principio difícilmente se podrá rediseñar sin desechar todo el trabajo previamente hecho.
- Depende mucho de la capacidad individual de los desarrolladores.
- Es difícil darle mantenimiento a un sistema construido con este modelo.

2.2.2 – Modelo en cascada

El modelo en cascada o modelo lineal secuencial es quizá el modelo más conocido que existe en el desarrollo de sistemas (de hecho también se le conoce como el ciclo de vida básico de un sistema), este modelo sugiere un enfoque sistemático y secuencial del desarrollo de un sistema y va progresando conforme pasa por las diferentes fases o actividades [31]. El modelo en cascada está manejado por los documentos, para avanzar de la fase actual a la siguiente fase, el equipo de desarrollo debe de revisar y liberar un documento con lo que se indica que el final de una fase ha concluido y va a comenzar una nueva fase [26].

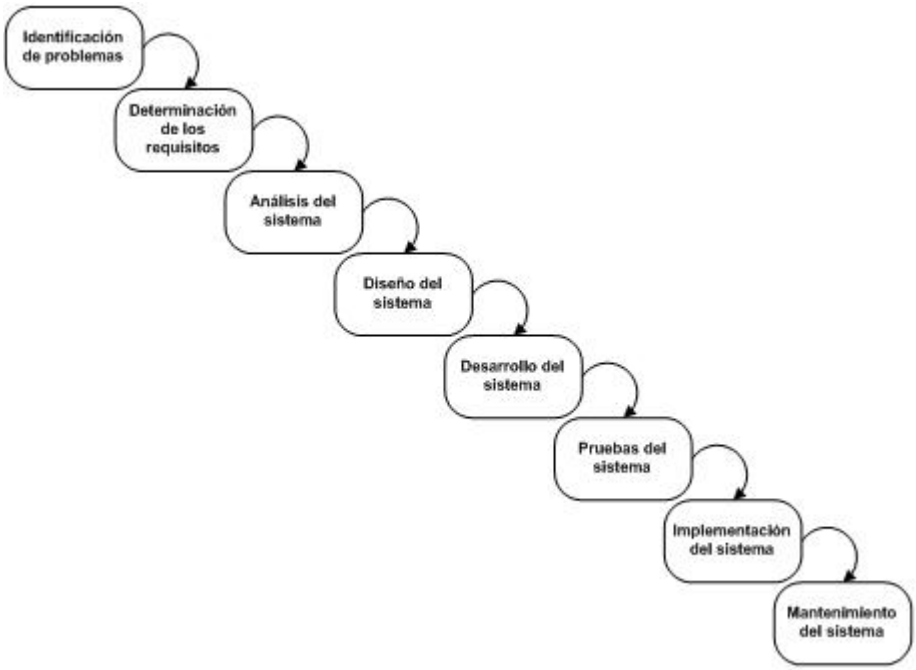


Figura 2.1 – Modelo en cascada

2.2.2.1 – Ventajas

- Este modelo trabaja muy bien cuando los requerimientos del sistema están bien definidos.
- Ayuda a localizar errores en las primeras etapas del proyecto.
- Ayuda a minimizar los gastos de la planificación ya que permite realizarla sin problemas.
- Funciona bien en proyectos donde se cuenta con personal poco cualificado o inexperto, porque presenta el proyecto con una estructura que ayuda a minimizar el esfuerzo inútil.

2.2.2.2 – Desventajas

- No proporciona resultados tangibles en forma de software hasta el final del ciclo de vida del sistema.
- Crea muchos problemas el tener que especificar los requisitos al comienzo del proyecto.
- No permite flexibilidad en los cambios de los requisitos.
- Hay una mínima retroalimentación entre las fases.

2.2.3 – Modelo de prototipos

Este modelo se utiliza principalmente cuando un cliente no tiene muy claro los requisitos del sistema que quiere que se desarrolle. El primer paso en este modelo es la recolección de requisitos, donde el desarrollador y el cliente se reúnen para definir en forma general los objetivos del software a desarrollar [31]. Dentro de estos requisitos generales existen algunos requerimientos que son conocidos y que sirven para diseñar y construir un

prototipo general del sistema que posteriormente se le muestra al cliente para que lo revise y verifique que está cumpliendo con los requisitos parciales que especificó, en caso contrario se rediseña el prototipo, se modifica y se vuelve a mostrar al cliente. Todo esto se hace hasta que el sistema satisface los requerimientos completos del cliente [29].

2.2.3.1 – Ventajas

- Es apropiado cuando los requerimientos cambian con rapidez
- El equipo de desarrollo no tiene que utilizar la arquitectura o los algoritmos más apropiados para el sistema
- Es visible inmediatamente el progreso del sistema
- Es muy útil para proyectos pequeños.

2.2.3.2 – Desventajas

- No se tiene conocimiento de cuanto tardará el proyecto en tener un sistema aceptable
- Se puede caer rápidamente en el modelo codificar y corregir
- En proyectos grandes se vuelve caótico el rediseño.

2.2.4 – Modelo incremental

El modelo incremental es un proceso evolutivo de software en donde se combinan el modelo en cascada y el modelo de prototipos [31]. En éste modelo se definen los requerimientos o funcionalidades del sistema en un principio y el cliente y el equipo de desarrollo deciden la prioridad de las funcionalidades que se deben de implementar en cada uno de los incrementos. No todos los requerimientos se deben de especificar detalladamente al principio del proyecto, se pueden dejar algunos para detallarlos posteriormente, ya avanzado el proyecto. Cada incremento se realiza utilizando un modelo de ciclo de vida lineal y al acabar cada incremento existe un sistema con una funcionalidad operando [35]. Generalmente en los primeros incrementos se realiza el núcleo del sistema. Los incrementos se deben de implementar por periodos cortos.

2.2.4.1 – Ventajas

- El sistema parcial desarrollado con cada incremento puede ser utilizado sin necesidad de terminar todo el sistema
- Existe un riesgo muy bajo de que el proyecto total falle, ya que normalmente el sistema se entrega de forma satisfactoria
- No se necesita de mucho personal en comparación con otros modelos

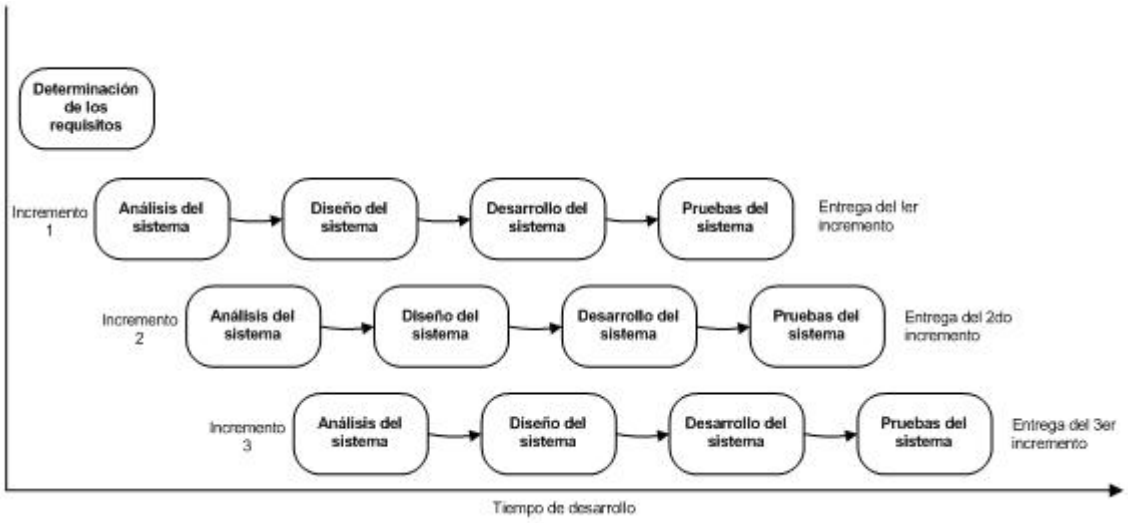


Figura 2.2 – Modelo incremental

2.2.4.2 – Desventajas

- Como los incrementos deben de ser relativamente cortos, suele ser difícil adaptarlos a los requerimientos del cliente.
- Al no detallar algunos requerimientos al principio de un proyecto es difícil identificar funcionalidades comunes

2.2.5 – Modelo en espiral

El modelo en espiral divide un proyecto en subproyectos orientados a riesgos. Cada subproyecto se centra en uno o más riesgos importantes hasta que todos estén controlados. Primero se elige al de mayor riesgo y después al siguiente con mayor riesgo de los que quedaron hasta abarcarlos a todos. Los riesgos se pueden referir a desconocimiento de una tecnología existente, posibles problemas de rendimiento, requerimientos poco comprensibles, entre otros. Después de controlar todos los riesgos más importantes en el proyecto el modelo en espiral finaliza como el modelo en cascada.

El modelo en espiral consiste de las siguientes actividades para reducir los riesgos en un proyecto:

1. Determinar metas, alternativas y restricciones: en el primer paso de la iteración, se elaboran las metas del sistema.
2. Evaluación de alternativas: en este paso se evalúan las alternativas recolectadas en el paso 1. Generalmente se implementa un prototipo para determinar la mejor solución posible o, en el peor de los casos para determinar que el proyecto no es factible.
3. Identificar y mitigar riesgos: Después de valorar las alternativas, se identifican posibles riesgos y mitigaciones. Identificando los riesgos se puede saber su dificultad.
4. Desarrollar los entregables para la iteración actual y verificar que son correctos: en este paso se implementa y se verifica el producto para la actual iteración.

5. Planear para la próxima iteración: después de haber verificado el producto de la actual iteración, se debe de planear la próxima iteración, también podríamos documentar posibles mejorar y errores conocidos para corregirlos en la actual iteración o en las posteriores.

Generalmente, el modelo en espiral es usado en combinación con otros modelos para realizar una reducción de riesgos en las etapas iniciales del desarrollo.

2.2.5.1 – Ventajas

- La principal ventaja es la división del desarrollo del sistema en pequeñas piezas, de tal manera que se afrontan los riesgos en las primeras etapas del proyecto.
- Permite trabajar con otros modelos de manera combinada
- Incorpora objetivos de calidad y gestión de riesgos
- Elimina errores y alternativas no atractivas al comienzo
- El modelo se adapta a cualquier tipo de actividad adicional

2.2.5.2 – Desventajas

- El modelo es muy complejo
- Requiere una considerable habilidad para la evaluación del riesgo
- Si un riesgo no es descubierto y gestionado, provocará problemas en el desarrollo

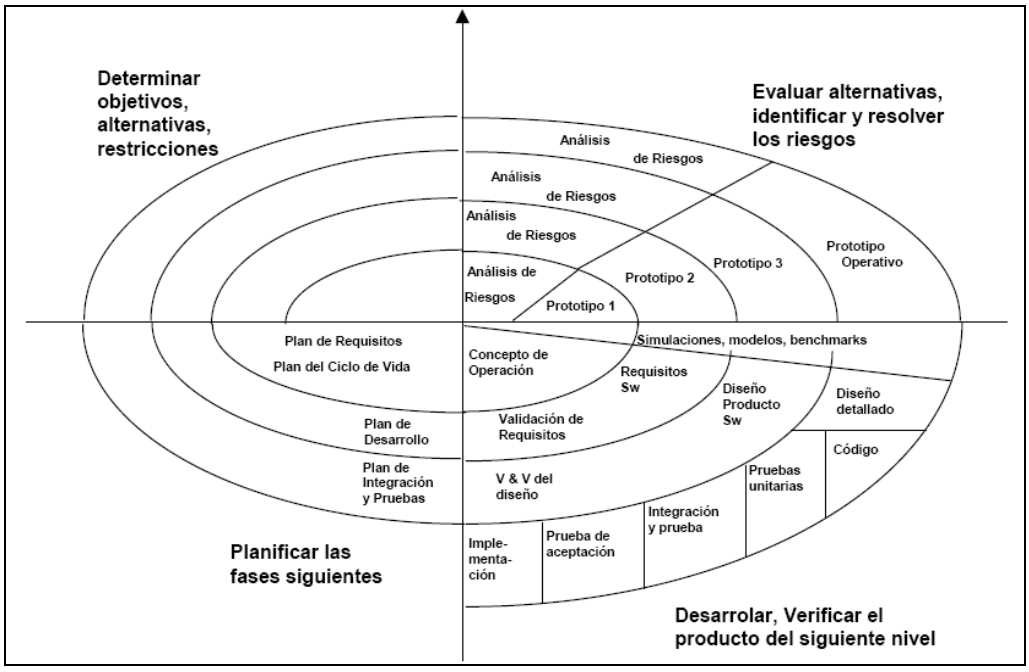


Figura 2.3 – Modelo en espiral

2.3 – METODOLOGÍAS ÁGILES

Las metodologías ágiles son un conjunto de metodologías que combinan una serie de directrices de desarrollo donde se resalta la entrega de un producto o sistema sobre el análisis y el diseño y la continua comunicación de los desarrolladores con los clientes. Poseen una filosofía donde se busca la satisfacción del cliente y la entrega temprana del software de manera incremental, con equipos pequeños de desarrollo y con alta motivación. La característica fundamental de estas metodologías es la habilidad de responder al cambio.

Las metodologías ágiles surgen en febrero del 2001 [11], en una reunión donde participaban 17 expertos de la industria del software, entre los que se incluían algunos de los creadores de las metodologías ágiles. En esta reunión se creó la Alianza Ágil [1] (*Agile Alliance*), que es una organización dedicada a promover las metodologías ágiles entre las organizaciones para que adopten los conceptos.

La compilación de los principios y valores que resaltan las metodologías ágiles fue formalizada en el manifiesto para el desarrollo de software ágil (que se verá más adelante). Este documento desarrollado por los representantes de cada una de las metodologías, que en el momento se presentaban como ágiles, logra resumir en un conjunto de ideas las prácticas que una metodología de este estilo debe llevar a cabo.

Como menciona Martin Fowler [1], las metodologías ágiles surgieron como respuesta a las metodologías tradicionales que imponen un proceso disciplinado con el fin de hacerlo más predecible y eficiente, y con un fuerte énfasis en planificar. Las metodologías ágiles son menos orientadas a la documentación, más adaptables a los cambios y orientadas a la gente.

En las metodologías tradicionales muchas veces se enfrentan con el problema de que es difícil planear bien todo el proceso de desarrollo y esto generalmente se llega a dar porque durante el proceso de desarrollo surgen cambios que se deben de hacer. Y afectan lo que ya estaba planeado.

Por eso sería importante tener diferentes metodologías que se puedan ocupar de acuerdo a los diferentes proyectos que se presenten porque en algunos podrías utilizar una metodología tradicional y en otro quizá una metodología ágil.

En el Cuadro 2.1 se muestran algunas de las diferencias que existen entre las metodologías ágiles y las metodologías tradicionales.

1 http://www.agilealliance.com
2 http://www.martinfowler.com/articles/newMethodology.html

Cuadro 2.1 – Diferencias entre las metodologías ágiles y las tradicionales

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Poca documentación	Mucha documentación
Poco análisis y diseño	Mucho análisis y diseño
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos
Pocos roles	Más roles

2.3.1 – Manifiesto para el desarrollo de software ágil

En el manifiesto para el desarrollo de software ágil están presentes los valores que caracterizan a las metodologías ágiles. El manifiesto ágil [3] es el siguiente:

Estamos descubriendo mejores formas de desarrollar software realizándolo y ayudando a otros a hacerlo. A través de este trabajo hemos llegado a valorar:

- **Individuos e interacciones** sobre procesos y herramientas
- **Software funcionando** sobre documentación exhaustiva
- **Colaboración del cliente** sobre negociación de contrato
- **Responder al cambio** sobre seguir un plan

Esto es, mientras en los productos de la derecha hay valor, valoramos más a los de la izquierda.

Las personas que realizaron el manifiesto ágil son las siguientes: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas.

Se puede comentar con más detalle lo que quieren decir cada uno de los puntos anteriores para comprender mejor porque llegaron a esas conclusiones:

3 http://www.agilemanifesto.org/

- **Individuos e interacciones sobre procesos y herramientas**. La gente es el principal factor de éxito de un proyecto software. Si se sigue un buen proceso de desarrollo, pero el equipo falla, el éxito no está asegurado; sin embargo, si el equipo funciona, es más fácil conseguir el objetivo final, aunque no se tenga un proceso bien definido. No se necesitan desarrolladores brillantes, sino desarrolladores que se adapten bien al trabajo en equipo. Así mismo, las herramientas (compiladores, depuradores, control de versiones, etc.) son importantes para mejorar el rendimiento del equipo, pero el disponer más recursos que los estrictamente necesarios también puede afectar negativamente. En resumen, es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo con base a sus necesidades.

- **Software funcionando sobre documentación exhaustiva** El software sin documentación es un desastre, por lo que deben de crear sólo los documentos necesarios. Estos documentos deben ser cortos y centrarse en lo fundamental. Si una vez iniciado el proyecto, un nuevo miembro se incorpora al equipo de desarrollo, se considera que los dos elementos que más le van a servir para ponerse al día son: el propio código y la interacción con el equipo.

- **Colaboración del cliente sobre negociación de contrato** Las características particulares del desarrollo de software hace que muchos proyectos hayan fracasado por intentar cumplir unos plazos y unos costes preestablecidos al inicio del mismo, según los requisitos que el cliente manifestaba en ese momento. Por ello, se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.

- **Responder al cambio sobre seguir un plan** La habilidad de responder a los cambios que puedan surgir a los largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta, puesto que hay muchas variables en juego debe ser flexible para adaptarse a los cambios que puedan surgir. Una buena estrategia es hacer planificaciones detalladas para unas pocas semanas y planificaciones mucho más abiertas para unos pocos meses.

2.3.2 – Principios detrás del manifiesto ágil

Detrás del manifiesto ágil, están los 12 principios [4] inspirados en él. Los principios resumen las características que definen una metodología ágil. Los principios son:

1. Nuestra mayor prioridad es satisfacer al cliente con la liberación oportuna y continua de software valioso.

4 (http://www.agilemanifesto.org/principles.html)

2. Son bienvenidos los requisitos cambiantes, aún avanzado el desarrollo. Los procesos ágiles utilizan el cambio para la ventaja competitiva del cliente.
3. Liberar software funcionando frecuentemente, de un par de semanas a un par de meses, de preferencia en la escala de tiempo más corta.
4. La gente de negocios y los desarrolladores deben trabajar juntos diariamente durante todo el proyecto.
5. Construir proyectos con individuos motivados. Darles el ambiente y soporte necesario, y confíe en que harán el trabajo.
6. La manera más efectiva de intercambiar información en el desarrollo es la conversación cara a cara.
7. El software funcionando es la medida primaria de progreso.
8. Los procesos ágiles promueven el desarrollo sustentable. Los patrocinadores, desarrolladores y usuarios deben ser capaces de mantener un paso constante indefinidamente.
9. La atención continua a la excelencia técnica y al buen diseño mejoran la agilidad.
10. La simplicidad (el arte de maximizar la cantidad de trabajo a no hacerse) es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto organizados.
12. A intervalos regulares, el equipo reflexiona sobre como ser más efectivo, entonces ajusta su desarrollo acordemente.

Todos los principios se pueden dividir en categorías que se muestran a continuación:

Los dos primeros principios son muy generales y resumen el espíritu ágil.

1. **Nuestra mayor prioridad es satisfacer al cliente con la liberación oportuna y continua de software valioso**. Un proceso es ágil si en pocas semanas de haber iniciado entrega software funcionando. El cliente debe de decidir si pone en marcha dicho software con la funcionalidad que proporciona en ese momento o simplemente lo revisa para informar de posibles cambios.

2. **Son bienvenidos los requisitos cambiantes, aún avanzado el desarrollo**. Debido a que los procesos ágiles utilizan el cambio para darle una ventaja competitiva al cliente. Este principio es una actitud que deben adoptar los miembros del equipo de desarrollo, es decir, los cambios en los requisitos deben verse como algo positivo ya que permitirá aprender más, y a la vez logran una mayor satisfacción del cliente. Este principio implica además que la estructura del software debe ser flexible para incorporar los cambios sin demasiado coste añadido. El paradigma orientado a objetos puede conseguir esta flexibilidad.

Los siguientes principios hablan sobre el proceso de desarrollo de software.

3. **Liberar frecuentemente software funcionando, de un par de semanas a un par de meses, de preferencia en la escala de tiempo más corta**. Las entregas al cliente debe ser software funcionando, y los periodos de entrega deben de estar entre dos semanas para proyectos pequeños y dos meses para proyectos más grandes, pero siempre tratando de que las entregas sean en el menor tiempo posible..

4. **La gente de negocios y los desarrolladores deben trabajar juntos diariamente durante todo el proyecto**. El proceso de desarrollo necesita ser guiado por el cliente, por lo que la interacción con el equipo es muy frecuente.

5. **Construir proyectos con individuos motivados.** Se debe proporcionar a los individuos el ambiente y soporte necesario, además de confiar en que harán el trabajo. La gente es el principal factor de éxito, todo lo demás (proceso, entorno, gestión, etc.) queda en segundo plano. Si cualquiera de ellos tiene un efecto negativo sobre los individuos debe ser cambiado.

6. **La manera más efectiva de intercambiar información en el desarrollo es la conversación cara a cara**. Los miembros de equipo deben hablar entre ellos, éste es el principal modo de comunicación. Se pueden crear documentos pero no todo estará en ellos.

7. **El software funcionando es la medida primaria de progreso**. El estado de un proyecto no viene dado por la documentación generada o la fase en la que se encuentre, sino por el código generado y en funcionamiento. Por ejemplo, un proyecto se encuentra al 50% si el 50% de los requisitos ya están en funcionamiento.

8. **Los procesos ágiles promueven el desarrollo sustentable. Los patrocinadores, desarrolladores y usuarios deben ser capaces de mantener un paso constante indefinidamente.** No se trata de desarrollar lo más rápido posible, sino de mantener el ritmo de desarrollo durante toda la duración del proyecto, asegurando en todo momento que la calidad de lo producido es máxima.

La última categoría de los principios están directamente relacionados con el equipo de desarrollo.

9. **La atención continua a la excelencia técnica y al buen diseño mejoran la agilidad.** La meta es producir código claro y robusto ya que es la clave para avanzar más rápidamente en el proyecto.

10. **La simplicidad (el arte de maximizar la cantidad de trabajo a no hacerse) es esencial.** Tomar los caminos más simples que sean consistentes con los objetivos perseguidos. Si el código producido es simple y de alta calidad será más sencillo adaptarlo a los cambios que puedan surgir.

11. **Las mejores arquitecturas, requisitos y diseños emergen de equipos auto organizados.** Todo el equipo es informado de las responsabilidades y éstas recaen sobre todos sus miembros. Es el propio equipo el que decide la mejor forma de organizarse, de acuerdo a los objetivos que se persigan.

12. **A intervalos regulares, el equipo reflexiona sobre como ser más efectivo, entonces ajusta su desarrollo acordemente.** Puesto que el entorno está cambiando continuamente, el equipo también debe ajustarse al nuevo escenario de forma continua.

Puede cambiar su organización, sus reglas, sus convenciones, sus relaciones, etc., para seguir siendo ágil.

2.4 – PROGRAMACIÓN EXTREMA (XP)

La programación extrema (*extreme programming* o XP) es una metodología reciente (con alrededor de 6 años) en el desarrollo de software. La filosofía de XP es satisfacer por completo las necesidades del cliente, por eso lo integra como una parte más del equipo de desarrollo.

La programación extrema fue inicialmente creada para el desarrollo de aplicaciones dónde el cliente no sabe muy bien lo que quiere, lo que provoca un cambio frecuente en los requisitos que debe cumplir la aplicación.

La programación extrema está diseñada para el desarrollo de aplicaciones que requieran un grupo de programadores pequeño, dónde la comunicación sea más factible que en grupos de desarrollo grandes [01]. La comunicación es un punto importante y debe realizarse entre los programadores, los jefes de proyecto y los clientes, es decir, debe de realizarse entre todos los miembros del proyecto.

2.4.1 – Valores

Los cuatro valores de la programación extrema son esenciales para que sus 12 practicas tengan sentido y puedan ser llevadas a cabo con éxito. Los valores son los siguientes:

Comunicación (*Communication*). Este valor se refiere a que el enfoque está en la comunicación oral y no en documentos, reportes o diagramas. Debe de existir una constante comunicación entre los miembros del equipo. Además los miembros del equipo deben de estar en constante comunicación con los clientes para satisfacer sus requisitos y responder rápidamente a los cambios de los mismos. Muchos problemas que surgen en los proyectos se deben a que no existe una buena comunicación.

Simplicidad (*simplicity*). Un equipo de programación extrema debe intentar mantener el software lo más sencillo posible. Esto quiere decir que no se debe de invertir ningún esfuerzo en hacer un desarrollo complicado que en un futuro pueda llegar a tener valor [19]. Se deben de hacer una codificación y diseños simples y claros. Muchos diseños son tan complicados que cuando se quieren ampliar resulta imposible hacerlo y se tienen que desechar y partir de cero.

Retroalimentación (*Feedback*). El valor de la retroalimentación es vital para el éxito de un proyecto. La retroalimentación se toma del cliente, de los miembros del equipo, es decir, de todo el entorno en el que se mueve el equipo de desarrollo. Mediante la retroalimentación se ofrece al cliente la posibilidad de conseguir un software apto a sus

necesidades ya que se le va mostrando el proyecto a tiempo para poder ser cambiado y retroceder a una fase anterior para rediseñarlo a su gusto.

Coraje (*Courage*). El coraje es un valor muy importante dentro de la programación extrema. Un miembro de un equipo de desarrollo extremo debe de tener el coraje de exponer sus dudas, miedos y experiencias. Esto es muy importante ya que un equipo de desarrollo extremo se basa en la confianza que hay entre sus miembros. Faltar a esta confianza es grave. Hay que tener valor para comunicarse con el cliente y enfatizar algunos puntos, a pesar de que esto pueda dar sensación de ignorancia por parte del programador, hay que tener coraje para mantener un diseño simple y no optar por el camino más fácil y por último hay que tener valor y confiar en que la retroalimentación sea efectiva.

2.4.2 – Principios

Dentro de la programación extrema existen cinco principios, derivados de los valores, que guían el desarrollo de software [02]:

Rápida retroalimentación (*Rapid Feedback*). La rápida retroalimentación es para que las personas o sistemas hagan una conexión entre estimulo y reacción, la retroalimentación debe de ocurrir en un intervalo razonablemente corto, es decir, en un proceso de aprendizaje, una rápida retroalimentación en periodos cortos es más significativa que una con un periodo de retroalimentación más largo. La rápida retroalimentación para el equipo de desarrollo significa que entre más cercano sea el tiempo de una acción (codificar una característica derivada de un reporte del usuario) al tiempo de la comprobación, más significativa será la retroalimentación (los resultados de la prueba). Entre más pronto un sistema se ponga en producción (en lugar de simplemente estar en desarrollo), mayor será el valor de la retroalimentación para el negocio al medir si el sistema está cumpliendo sus metas.

Adoptar la simplicidad (*Assume Simplicity*). El principio se refiere a que alrededor del 90 por ciento de los problemas se puede resolver con sencillez. Esto quiere decir que para resolver un problema sólo debe de pensar en hacer algo sencillo y no pensar en hacer algo más complejo porque posiblemente en un futuro lo va a necesitar. Se pide que el enfoque sólo sea en la iteración que se está trabajando y no en iteraciones futuras.

Cambiar incrementalmente (*Incremental Change*). Esto quiere decir, que no se debe de hacer un cambio total a un diseño, sino que se deben de hacer pequeños cambios de una manera incremental, y esto no sólo se debe de aplicar al diseño, también a la planeación y al desarrollo.

Aceptar el cambio (*Embrace Change*). Esto nos dice que debemos de tener abiertas todas nuestras opciones mientras resolvemos los problemas más urgentes. El cambio debe de estar presente en todo momento y no debe causar temor por hacerlo, al contrario la programación extrema está hecha precisamente para aceptar el cambio.

Trabajo de Calidad (*Quality Work*). Este principio se refiere a que todos los miembros del equipo deben de hacer un trabajo de calidad para que el éxito del proyecto no esté en riesgo. El punto es hacer el trabajo agradable, trabajar adecuadamente con el equipo y mantener el proyecto sano y salvo.

Los anteriores principios son los principales o más importantes que tiene la programación extrema pero existen también otros principios como: la obligación de enseñar a aprender; el estimulo de hacer una pequeña inversión para hacer un buen trabajo; jugar para ganar; usar experimentos concretos para probar el trabajo que se está haciendo; usar una comunicación abierta y honrada; aprovechar los instintos de la gente y no estar en contra de ellos; aceptar la responsabilidad de hacer algo; adaptarse localmente y hacer mediciones honestas.

2.4.3 – Prácticas

La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las siguientes prácticas.

1. **El juego de la planificación (*The Planning Game*).** Dentro del juego de la planificación debe existir una comunicación frecuente entre el cliente y los programadores para definir los requerimientos y la planeación del proyecto. Los desarrolladores hacen una estimación del esfuerzo y tiempo requerido para la implementación de las historias de usuario y los clientes deciden cuales historias tienen mayor prioridad para programar las entregas. Generalmente se toman las tareas más difíciles primero, para reducir el riesgo global asociado con el proyecto.

2. **Entregas pequeñas (*Small Releases*)**. La idea con esta práctica es producir rápidamente versiones o entregas pequeñas del sistema que sean operativas, aunque no tengan toda la funcionalidad del sistema. Cada versión ya constituye un resultado de valor para el negocio. Las versiones deberían ser entregadas cada dos o tres semanas de manera que los clientes puedan ver y tocar el producto funcionando de manera regular.

3. **Metáfora (*Metaphor*)**. Con esta práctica 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 completo. La metáfora expresa la visión evolutiva del proyecto que define el alcance y propósito del sistema.

4. **Diseño simple (*Simple Design*)**. Se debe de diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto. El diseño simple se enfoca en proporcionar un sistema que cubra las necesidades inmediatas del cliente.

5. **Pruebas (*Testing*)**. En esta práctica los programadores escriben las pruebas de unidad para validar la operación correcta de los módulos o clases antes de escribir el código

funcional para el módulo o clase en desarrollo. Las pruebas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente después de alguna modificación en el sistema.

6. **Refactorización (*Refactoring*)**. Es una práctica que se hace constantemente para reestructurar el código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y quizás hacerlo más flexible para facilitar los posteriores cambios. Con esta práctica lo que se quiere es mejorar la estructura interna del código sin alterar su comportamiento externo [15].

7. **Programación en parejas (*Pair Programming*)**. La escritura de código debe realizarse en parejas de programadores. Esto implica que tienen que trabajar juntos dos programadores en una sola computadora para desarrollar y discutir el código de cada uno de ellos. Esto tiene varias ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores, etc.)[40].

8. **Propiedad colectiva del código (*Collective Ownership*)**. Esta práctica indica que no existe un programador que tenga la propiedad de una parte del código. Esto es de gran ayuda cuando se realizan cambios, ya que no es necesario que el programador que hizo el código sea él quien lo modifique.

9. **Integración continua (*Continuous Integration*)**. Esta práctica nos dice que cada pieza de código que es modificada, es integrada en el sistema una vez que esté lista. La integración de un conjunto de cambios al mismo tiempo simplifica el proceso de integración y deja visible quién es responsable del arreglo del código cuando las pruebas de integración fallan.

10. **40 horas por semana (*40-Hour Work Week*)**. Se debe trabajar un máximo de 40 horas por semana y no se deben de trabajan horas extras dos semanas seguidas, porque el trabajo extra desmotiva al equipo y las personas sin cansancio, generalmente, están más motivados y son más productivos.

11. **Cliente en el sitio (*On-Site Customer*)**. Esto se refiere a que el cliente tiene que estar presente y disponible todo el tiempo para el equipo, porque es parte del equipo permanece con el equipo de trabajo durante todo el proyecto. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. Esto nos dice que la comunicación oral es más efectiva que la escrita.

12. **Estándares de codificación (*Coding Standards*)**. Otra forma de comunicación entre los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible. Estos estándares de codificación son un requerimiento obligatorio. Los programadores deben de seguir un conjunto de reglas comunes de manera que todo el código en el sistema se vea como si hubiera sido escrito por una sola persona.

Para conseguir un mayor beneficio de las prácticas se debe de hacer una aplicación conjunta y equilibrada, ya que cada una de las prácticas se apoya en otras. Aunque las prácticas no son una novedad el merito que se merece la programación extrema es integrarlas en una forma efectiva de trabajo complementándolas con otras ideas del manejo de negocios, los valores humanos y el trabajo en equipo.

2.4.4 – Actividades

Existen cuatro actividades que se realizan durante todo un proyecto en la programación extrema:

1. Codificar (*Coding*).
2. Probar (*Testing*).
3. Escuchar (*Listening*).
4. Diseñar (*Designing*).

Codificar. La programación extrema es programación a final de cuentas y escribir código es una habilidad que se refina a través de las prácticas de refactorización, programación en parejas y la revisión de código. Los desarrolladores escriben código intencionalmente para expresar un sentido, porque con la sola lectura del código fuente, otros miembros del equipo pueden comprender su lógica, algoritmos y flujos.

Probar. El probar no debe de ser una idea tardía antes de entregar algo al cliente, sino como un paso integral a lo largo de todo el proceso de desarrollo. Lo primero que se tiene que hacer antes de codificar, es escribir las pruebas de aceptación del código que se vaya a desarrollar. Esto implica un cambio en la manera de pensar de los programadores, pero da como resultado una calidad integrada en el código en vez de tener que encontrar defectos en el software en posteriores etapas.

Escuchar. La programación extrema está basada en la comunicación y tiene prácticas que requieren una comunicación activa para poder realizarlas. La comunicación verbal con calidad es preferible a la documentación formal escrita.

Diseñar. Una de las ideas radicales en la programación extrema es la de diseñar muy poco en comparación con otras metodologías. La idea del diseño es que puede evolucionar y crecer a lo largo del proyecto. El diseño no debe de estar solo en una etapa o parte del proceso, debe de estar presente en todo el proceso.

2.4.5 – Variables

Para terminar un proyecto con éxito es muy importante llevar una buena administración. La administración del proyecto implica tomar decisiones para balancear los recursos y las carencias que se tienen. Dentro de la programación extremase tenen cuatro variables de control de recursos que junto con las actividades de la programación extrema, deben de alcanzar un equilibrio para que el proyecto tenga éxito.

Las variables de control de recursos son:

1. Tiempo (*Time*).
2. Costo (*Cost*).
3. Calidad (*Quality*).
4. Alcance (*Scope*).

Tiempo. El tiempo es una de las variables que se deben controlar porque es necesario planificar el tiempo suficiente para terminar un proyecto. Una estimación mal hecha del tiempo puede provocar que no se termine a tiempo el proyecto. Para esto el cliente quizá puede dejar de implementar alguna funcionalidad secundaria que no le afectaría mucho a su negocio y con eso lograr que el proyecto se termine a tiempo.

Costo. También el costo juega un papel importante dentro de las variables de control de recursos. Por ejemplo, cuando un proyecto esta atrasado por problemas que se hayan tenido, quizás se decida contratar más gente para desarrollar con más rapidez el proyecto. Aunque esto no siempre es cierto puesto que el contratar más gente puede provocar que se retracen más. Algo en lo que se puede invertir para aumentar el costo y ayudar a terminar a tiempo el proyecto es mejorando el equipo tecnológico con el que se desarrolla o también en teléfonos celulares, computadoras portátiles o mejores herramientas software para el equipo de desarrollo.

Calidad. Esta es otra de las variables que se pueden controlar dentro de la programación extrema, y quizá se ponga menos esfuerzo del esperado en mantener la calidad. La calidad a la que se refiere este recurso es a la externa, es decir, el cómo percibe el cliente el sistema. Para que el sistema sea liberado a tiempo, tal vez el cliente tenga que lidiar con algunos defectos no críticos del software, como puede ser la interfaz de usuario.

Alcance. El alcance se determina escuchando al cliente y poniéndolo a redactar sus historias. Después el equipo de desarrollo examina las historias para calcular cuánto puede hacerse en un tiempo específico para satisfacer al cliente. El equipo de desarrollo no sólo determina el tiempo, sino también el costo y la calidad de cada uno de las historias. El alcance se tiene que definir junto con el cliente.

Es importante señalar que de las cuatro variables de control de recursos, tres las manejará el cliente y la cuarta el equipo de desarrollo.

2.4.6 – Roles

En un proyecto que se desarrolla con la programación extrema se deben de definir los roles que juegan cada uno de los miembros del equipo de desarrollo. Un miembro del equipo de desarrollo puede tener varios roles.

Los roles que se definen en la programación extrema son:

Programador (*Programmer*). Es el corazón del desarrollo de la programación extrema, donde sus habilidades de comunicación entran en juego tan pronto como empieza el desarrollo. También el programador necesita contar con excelentes habilidades técnicas para programar, refactorizar y realizar pruebas unitarias al código que escriba, además de una buena disposición para abordar con sencillez los problemas más difíciles, aprender de otros, compartir el código y el diseño, y tener el valor para superar cualquier temor de incompetencia o fracaso al enfrentar nuevos problemas.

Cliente (*Customer*). Debe ser alguien que será usuario del sistema y que conoce la funcionalidad del negocio. El cliente escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio. Los clientes también necesitan demostrar que tienen valor para enfrentar problemas graves referentes a la programación del proyecto y al funcionamiento del sistema.

Probador (*Tester*). Mientras que a los programadores se les pide que realicen las pruebas unitarias y de funcionamiento del nuevo código que haya escrito. El probador ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, es responsable de las herramientas de soporte para pruebas y elabora informes precisos acerca de los resultados de las pruebas.

Rastreador (*Tracker*). El rastreador proporciona retroalimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración. Los rastreadores también fungen como memoria del equipo, al dar seguimiento a los resultados de las pruebas de funcionamiento. Además, registran los defectos reportados y el nombre del programador a cargo del manejo de cada defecto y dan seguimiento a os casos de prueba agregados para resolver los defectos.

Entrenador (*Coach*). Es responsable de guiar el proceso global. Debido a que una de las características distintivas de la programación extrema es que cada persona acepta la responsabilidad por sus acciones, un entrenador podría parecer innecesario, pero los entrenadores son muy importantes. Un buen entrenador debe de resaltar las mejores cualidades de todos los miembros del equipo, además debe de proveer de guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

Consultor (*Consultant*). El consultor es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir

problemas. El consultor debe de enseñarles a los demás miembros del equipo a resolver sus problemas y no resolverlos él directamente, ya que esto les da confianza.

Gestor o gran jefe (*Big boss*). Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación y conseguir que la comunicación fluya sin convertirse en una ola de rumores. Este rol exige una total convicción en el enfoque de la programación extrema y de que si todos en el equipo se apegan a sus valores y principios básicos.

2.4.7 – *User Stories*

Las *User Stories* tienen la misma finalidad que los casos de uso pero con algunas diferencias: constan de 3 ó 4 líneas escritas por el cliente en un lenguaje no técnico sin muchos detalles; no se debe hablar de posibles algoritmos para su implementación ni de diseños de base de datos adecuados u otras cosas.

Las *User Stories* son usadas para estimar tiempos de desarrollo de la parte de la aplicación que describen. Se utilizan en la fase de planeación y en la fase de pruebas, para verificar si el programa cumple con lo que especifica la *User Story*. Cuando llega la hora de implementar una *User Story*, el cliente y los desarrolladores se reúnen para concretar y detallar lo que tiene que hacer dicha *User Story*. El tiempo de desarrollo ideal para cada una, está entre 1 y 3 semanas. Si la *User Story* puede tomar más semanas significa que se puede dividir en *User Stories* más pequeñas [20].

2.4.8 – *Release plan*

Es una planificación del proyecto, en donde los desarrolladores y clientes establecen los tiempos de implementación de cada una de las *User Stories* que describen la funcionalidad del sistema, la prioridad de cada *User Story* para ser implementada, el riesgo de cada una de ellas y las que serán implementadas en cada versión del programa.

Con el *Release plan* deben de quedar claros estos cuatro factores: los objetivos que se deben cumplir (que son principalmente las *User Stories* que se deben desarrollar en cada versión), el tiempo que tardarán en desarrollarse y publicarse las versiones del programa, el número de personas que trabajarán en el desarrollo y cómo se evaluará la calidad del trabajo realizado [30].

2.4.9 – Fases

Todo proyecto de desarrollo utilizando la programación extrema debe de pasar por las fases de planificación, diseño, codificación y pruebas. Estas fases se explicaran en detalle a continuación:

2.4.9.1 – Planificación.

Aquí se definen las *User Stories* junto con el cliente para describir la funcionalidad del sistema. Todas las *User Stories* se colocan en una carta índice. Posteriormente se hace un *Release plan* y los desarrolladores estiman el tiempo que se utilizará para el desarrollo de cada una de ellas, su prioridad y las posibles fechas en que se deben de liberar cada una de las *User Stories*.

Los clientes y los desarrolladores deciden como agrupar las *User Stories* para los incrementos posteriores. Una vez establecido el acuerdo de las *User Stories* que se incluirán, la fecha de entrega y otras cuestiones del proyecto para su lanzamiento, los desarrolladores ordenan las *User Stories* que se implementaran de una de las siguientes formas:

1. Todas serán implementadas de un modo inmediato, es decir, dentro de unas pocas semanas.
2. Las que tengan la prioridad más alta se implementaran al principio.
3. Las más riesgosas se desarrollaran al principio.

Después de que se ha entregado la primera liberación del proyecto, el equipo de desarrollo calcula la velocidad del proyecto basándose en las *User Stories* que fueron desarrolladas y las *User Stories* que no pasaron las pruebas de aceptación. Es conveniente reevaluar la velocidad del proyecto cada 3 ó 4 incrementos o iteraciones y si se aprecia que no es adecuada hay que negociar con el cliente un nuevo *Release plan*.

La velocidad del proyecto puede utilizarse para:

1. Ayudar a estimar fechas de entrega y la programación de liberaciones subsecuentes
2. Determinar si se hizo un compromiso excesivo en algunas de las historias de todo el proyecto de desarrollo.

Conforme avanza el trabajo de desarrollo, el cliente puede agregar *User Stories*, cambiar el valor de la *User Story* existente, dividirlas o eliminarlas. Así, el equipo de desarrollo considera de nuevo las liberaciones restantes y modifica los planes de acuerdo a los cambios.

Es necesario que los desarrolladores se reúnan diariamente y expongan sus problemas, soluciones e ideas de forma conjunta. Las reuniones tienen que ser fluidas y todo el mundo debe tener voz y voto.

2.4.9.2 – Diseño.

El diseño de la programación extrema sugiere de manera rigurosa hacer diseños sencillos. Procurando hacerlo todo lo menos complicado para conseguir un diseño fácil de entender y de desarrollar, que a la larga costará menos tiempo y esfuerzo desarrollar. La programación extrema desaprueba el diseño de funcionalidad extra, es decir, añadirle funcionalidad al diseño que no se va a utilizar en el momento sino que más tarde en el proyecto.

La programación extrema recomienda el uso de las tarjetas CRC (Clases – Responsabilidades - Colaboración) en el diseño para permitirle al programador centrarse y apreciar el desarrollo orientado a objetos, olvidándose de los malos hábitos de la programación procedimental clásica. Las tarjetas CRC identifican y organizan las clases orientadas al objeto que son relevantes para el incremento del software actual

Se recomienda en esta fase usar un glosario de términos y una correcta especificación de los nombres de métodos y clases que ayudará a comprender el diseño y facilitará sus posteriores ampliaciones y la reutilización del código.

Si durante el diseño se encuentra un problema difícil como parte del diseño de la historia, la programación extrema recomienda la creación de un prototipo operacional de esa parte del diseño. El prototipo del diseño se evalúa para reducir el riesgo cuando comience la verdadera implementación y validar las estimaciones que tenía la historia que contiene el problema de diseño.

Otra de las cosas que se recomiendan en la parte de diseño es la refactorización. Aunque refactorizar es una técnica de construcción, también se utiliza en el diseño. Como ya se mencionó anteriormente, en las prácticas de la programación extrema, refactorizar es mejorar y modificar la estructura y codificación de códigos ya creados sin alterar su funcionalidad. Refactorizar supone revisar de nuevo estos códigos para procurar optimizar su funcionamiento.

Una idea central de la programación extrema es que el diseño ocurre antes y después del comienzo de la codificación.

2.4.9.3 – Codificación.

La programación extrema recomienda que después de diseñar las *User Stories* y realizar el diseño preliminar, el equipo de desarrollo no debe moverse inmediatamente a la codificación, sino que debe de desarrollar una serie de pruebas de unidad que comprueben cada una de las *User Stories* que vayan a incluirse en el incremento actual.

Crear pruebas de unidad que prueben el funcionamiento de los distintos códigos implementados ayudará a implementar el código para pasar la prueba de unidad. Crear estas pruebas antes ayuda a saber qué es exactamente lo que tiene que hacer el código a implementar y saber que una vez implementado pasará dichas pruebas sin problemas ya que dicho código ha sido diseñado para ese fin. Se puede dividir la funcionalidad que debe cumplir una tarea a programar en pequeñas unidades, de esta forma se crearán primero las pruebas para cada unidad y a continuación se desarrollará dicha unidad, así poco a poco se conseguirá un desarrollo que cumpla todos los requisitos especificados.

La codificación debe hacerse apegándose a estándares de codificación ya creados, como se menciona en las prácticas de la programación extrema. Programar bajo estándares mantiene el código consistente y facilita su comprensión y escalabilidad.

Además la codificación debe hacerse con la programación en pareja ya que permite un código más eficiente y con una gran calidad. También alienta a que los desarrolladores se mantengan concentrados en el problema que se tiene al momento. En la práctica, cada persona tiene un papel sutilmente diferente. Por ejemplo, una persona puede pensar en los detalles de codificación de una porción particular en el diseño, mientras que la otra se asegura de que se sigan los estándares de codificación.

También se deben de usar repositorios de código dónde las parejas de programadores deben de publicar cada pocas horas sus códigos implementados y corregidos junto a las pruebas que deben pasar. De esta forma el resto de programadores que necesiten códigos ajenos trabajarán siempre con las últimas versiones. Para mantener un código consistente, cada pareja de programadores tiene como acción exclusiva, el publicar su código en el repositorio.

La programación extrema propone un modelo de desarrollo colectivo en el que todos los programadores están implicados en todas las tareas; cualquiera puede modificar o ampliar una clase o método de otro programador si es necesario y subirla al repositorio de código. El permitir al resto de los programadores modificar códigos que no son suyos no supone ningún riesgo ya que para que un código pueda ser publicado en el repositorio tiene que pasar las pruebas de funcionamiento definidos para el mismo.

La optimización del código siempre se debe dejar para el final. Hay que hacer que funcione y que sea correcto, más tarde se puede optimizar.

Cuando los desarrolladores completan su trabajo, el código que desarrollaron se integra con el trabajo de otros diariamente. Este trabajo lo hace el equipo de integración que se forma o los programadores. Esta estrategia ayuda a evitar problemas de compatibilidad e interfaz y proporciona de pruebas que ayuda a descubrir errores.

2.4.9.4 – Pruebas.

Una de las bases en la programación extrema es el uso de pruebas para comprobar el funcionamiento de los códigos que vayamos implementando. Las pruebas deben de implementarse en un entorno que permita automatizarlas, para facilitar su ejecución de una forma repetida.

Hay que someter a pruebas las distintas clases del sistema omitiendo los métodos más triviales. Se deben crear las pruebas que pasarán los códigos antes de implementarlos (como se menciono en la fase de codificación).

Un punto importante es crear pruebas que no tengan ninguna dependencia del código que evaluará después. Se deben de crear las pruebas abstrayéndose del futuro código, de esta forma aseguraremos la independencia de la prueba respecto al código que evalúa.

Todas las pruebas se deben subir al repositorio de códigos acompañados del código que verifican. Ningún código puede ser publicado en el repositorio sin que haya pasado su prueba de funcionamiento, de esta forma, aseguramos el uso colectivo del código

(explicado en el apartado anterior). El uso de las pruebas es adecuado para observar la refactorización, ya que permiten verificar que un cambio en la estructura de un código no tiene porqué cambiar su funcionamiento.

Otra cosa que se utiliza en la fase de pruebas son las pruebas de aceptación. Las pruebas de unidad sirven para evaluar las distintas tareas en las que ha sido dividida una historia de usuario. Para asegurar el funcionamiento final de una determinada *User Story* se deben de crear pruebas de aceptación que son creados y usados por los clientes para comprobar que las distintas *User Stories* cumplen con lo especificado.

2.4.10 – Ciclo de vida de un proyecto

En el apartado anterior se vieron las fases por las que pasa un proyecto de programación extrema. En este apartado veremos las etapas tiene un proceso de desarrollo de la programación extrema.

El proceso de desarrollo de la programación extrema consta de cinco etapas: exploración, planeación, iteraciones, producción, mantenimiento y muerte del proyecto [22].

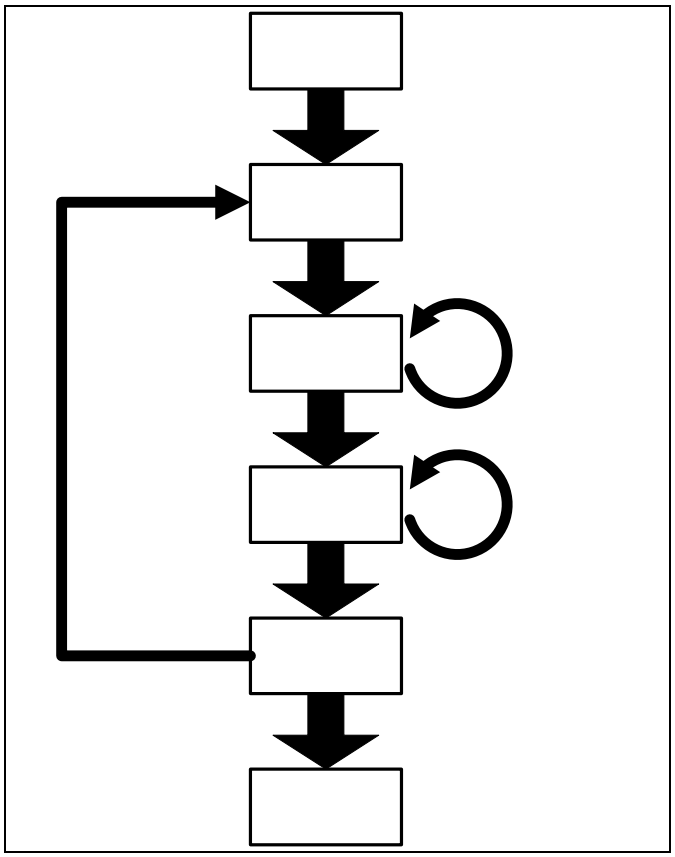


Figura 2.4 – Ciclo de vida de un proyecto en programación extrema

2.4.10.1 – Exploración

Durante la etapa de exploración se examina el entorno para saber si el proyecto se debe de realizar con programación extrema. Si se elige la programación extrema entonces hay que formar el equipo de desarrollo y valorar las habilidades de los miembros del equipo. La etapa puede durar desde unas cuantas semanas hasta meses, dependiendo si se conoce la tecnología base que se va a utilizar en el proyecto. Además de examinar tecnologías que potencialmente se ocuparán y no se conocen.

Los clientes también tienen que trabajar escribiendo las *User Stories*. El objetivo es que el cliente refine lo suficiente un relato para que se pueda calcular con eficiencia la cantidad de tiempo que se tomará en implementar la solución en el sistema que se piensa desarrollar. Lo importante en esta etapa es adoptar una actitud desenvuelta y de curiosidad con el entorno de trabajo, sus problemas, tecnologías y personas involucradas.

2.4.10.2 – Planeación

Esta es una fase donde el cliente y los desarrolladores llegan a un acuerdo sobre qué *User Stories* estarán en la primera liberación o incremento. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente.

En esta fase el cliente tiene las siguientes responsabilidades:

- Decidir el valor o prioridad que tiene cada una de las *User Stories* para su negocio.
- Debe decidir qué *User Stories* deben desarrollarse en la liberación.

Mientras que los desarrolladores tienen las siguientes:

- Estimar cuanto tiempo va a tomar desarrollar cada historia.
- Advertir al cliente de riesgos técnicos
- Medir el progreso del equipo de desarrollo

Las estimaciones de esfuerzo asociado a la implementación de las *User Stories* la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las *User Stories* generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la velocidad de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las *User Stories* que fueron terminadas en la última iteración. La planificación se puede realizar basándose en el tiempo o el alcance.

Si las actividades que se realizaron en la fase de exploración fueron suficientes, la etapa de planeación debe de ser corta.

2.4.10.3 – Iteraciones

El *Release plan* está compuesto, como se mencionó anteriormente, por iteraciones de a lo más tres semanas. En la primera iteración se puede intentar establecer una arquitectura global del sistema que llegue a ser utilizada durante el todo del proyecto.

Una meta es realizar pruebas de funcionamiento escritas por el cliente al final de cada iteración.

El ciclo que tienen las iteraciones es el siguiente:

1. Se seleccionan las *User Stories* que eligió el cliente para la primera iteración
2. Se estima y prioriza el trabajo de desarrollo de las *User Stories*
3. Se desarrollan y se prueban las historias

Al final de la última iteración el sistema estará listo para entrar en producción. Los elementos que deben tomarse en cuenta durante la elaboración del plan de la iteración son: *User Stories* no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior.

2.4.10.4 – Producción

Durante esta etapa se realizan diversas actividades. El ciclo de retroalimentación se acelera, de tal manera que en lugar de recibirla cada tres semanas, las revisiones se harán cada semana. Se deben hacer sesiones informativas para que todo el equipo sepa lo que hacen los demás.

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

Al final de esta etapa el producto o sistema es liberado. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación.

2.4.10.5 – Mantenimiento

Una vez liberado el sistema, el equipo de desarrollo debe de mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción.

Se pueden agregar nuevas características, se pueden tomar en cuenta las sugerencias más arriesgadas del cliente y se puede cambiar o incorporar nuevos miembros en el equipo de desarrollo.

2.4.10.6 – Muerte del proyecto

Se llega esta etapa cuando el cliente no tiene más *User Stories* para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.

CAPÍTULO 3

HERRAMIENTAS

CAPÍTULO 3 - HERRAMIENTAS

Entre las herramientas utilizadas para el desarrollo de la aplicación está la plataforma .NET, que surge por iniciativa de Microsoft. Se explicará qué es y sus componentes, además de sus características que lo hacen una buena opción para desarrollar aplicaciones.

SQL Server 2000 es la otra herramienta que se describe en este capítulo. Se habla de lo qué es, las tareas que realiza y una descripción de su arquitectura.

3.1 - PLATAFORMA .NET

La plataforma .NET es una iniciativa, propuesta por Microsoft, para una nueva arquitectura tecnológica que sirve para crear aplicaciones con la finalidad de ofrecer el software como un servicio. Al software que se crea con ese modelo se le conoce como Servicios Web [36].

Para lograr el modelo de programación propuesto se hace uso de estándares como los siguientes:

- **Lenguaje de marcado extensible (*extensible markup language* o XML)**. Es un formato universal para representar datos en texto plano separando la estructura de su contenido.

- **Protocolo simple de acceso a objetos (*simple object access protocol* o SOAP)**. Es un protocolo estándar, basado en XML, para el intercambio de datos en entornos descentralizados. Es decir, es el mecanismo por el cual los Servicios Web son invocados para interactuar entre sí.

- **Descubrimiento, descripción e integración universal (*universal description, discovery, and integration* o UDDI)**. Es un lenguaje que permite publicar, encontrar y usar los Servicios Web. Es un directorio para los servicios Web para poder encontrarlos.

- **Lenguaje de descripción de servicios Web (*Web services description language* o WSDL)**. Es un lenguaje por el cual un Servicio Web describe, entre otras cosas, qué hace o qué funcionalidad implementa.

La plataforma .NET es un conjunto de productos, desde sistemas operativos (*Windows XP*, *Windows 2000*, *Windows Me*, etc.), servidores de aplicaciones (*SQL Server 2000*, *Application Center 2000*, *BizTalk Server 2000*, etc.), herramientas de desarrollo (*.NET Framework*, *.NET Framework SDK* y *Visual Studio .NET*) hasta Servicios Web *(.NET Passport*. *.NET Alerts*, etc.).

3.1.1 – Componentes de la plataforma .NET

Como se mencionó anteriormente, la plataforma .NET está formada por un conjunto de productos que se agrupan en cuatro componentes principales:

- Herramientas de desarrollo para .NET
- Servicios Web XML .NET
- Servidores empresariales .NET
- Software para clientes .NET

3.1.1.1 - Herramientas de desarrollo para .NET

Las herramientas disponibles para el desarrollador son el marco de trabajo .NET (*.NET Framework*), el kit de desarrollo de software (*.NET framework SDK*) y *Visual Studio .NET*. Estas herramientas nos permiten crear aplicaciones en .NET.

.NET Framework. El marco de trabajo .NET es el elemento principal en el cual se basa la tecnología .NET y se agrupa en dos bloques principales: el tiempo de ejecución de lenguaje común (c*ommon language runtime* o CLR), y la biblioteca de clases de .NET (*.NET framework class library* o FCL).

.NET Framework SDK. Es el kit de desarrollo de software que proporciona todo lo que se necesita para escribir, generar, probar e implementar aplicaciones para el marco de trabajo .NET. Además de documentación, ejemplos, herramientas de línea de comandos y compiladores.

Visual Studio .NET. Es un entorno de desarrollo integrado (*integrated development enviroment* o IDE). Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI[1]. *Visual Studio .NET* permite crear Servicios Web, aplicaciones Windows, aplicaciones basadas en consola, y soluciones Web. *Visual Studio .NET* esta acoplado con el marco de trabajo .NET para permitirnos escribir aplicaciones en diferentes lenguajes como *Visual Basic .NET*, *Visual C++ .NET*, *Visual C#* y *Visual J#.*

[1] http://es.wikipedia.org

3.1.1.2 - Servicios Web .NET

La plataforma .NET tiene, principalmente, dos servicios Web que se complementan: pasaporte .NET (*.NET passport*) y alertas .NET (*.NET alerts*). Ambos proporcionan las bases para hacer aplicaciones y sitios donde la interfaz es consistente y sencilla. Además como son Servicios Web basados en XML pueden ser usados independientemente del dispositivo físico, sistema operativo o lenguaje de programación [37].

Pasaporte (*.NET passport*). Es un servicio de identidad desarrollado con tecnología de autenticación Windows, proporciona diversos niveles de autenticación que van desde contraseñas hasta tarjetas inteligentes y dispositivos biométricos.

Alertas (*.NET alerts*). Es un servicio de notificación y mensajería, para integrar la mensajería instantánea, correo electrónico, fax, correo de voz y otras formas de notificación y mensajería en una experiencia unificada, ofrecida a cualquier PC o dispositivo inteligente.

Otros servicios que tiene la plataforma .NET son los siguientes:

Calendario (*.NET calendar*). Mantiene un calendario personal del usuario.
Contactos (*.NET contacts*). Proporciona una lista de nombres**,** direcciones y cualquier otro tipo de información para el usuario.
Bandeja de entrada (*.NET inbox*). Permite acceder a un usuario a su coreo electrónico.
Documentación (*.NET documents*). Proporciona almacenamiento accesible por Internet para un usuario.
Cartera (*.NET wallet*). Contiene información de pagos, tal como un número de tarjeta de crédito y dirección de envío.
Perfil (*.NET profile*). Contiene información sobre un individuo como puede ser su nombre, dirección y fotografía.
Presencia (*.NET presence*). Contiene información electrónica de su presencia, tal como cuándo y dónde ese usuario es alcanzable vía mensaje instantáneo.

Además de todos estos Servicios Web se pueden crear servicios propios o utilizar servicios creados por terceros, lo cual permitiría tener un amplio conjunto de servicios, de los cuales se eligiría a los que sirvan para crear aplicaciones específicas.

3.1.1.3 - Servidores empresariales .NET

Los servidores empresariales .NET no ocupan directamente el marco de trabajo .NET pero, a pesar de eso, representan la manera más rápida y confiable de integrar y administrar aplicaciones habilitadas para el Web.

La siguiente lista proporciona una breve descripción de cada servidor empresarial .NET [37]:

Application Center 2000. Esta es una herramienta de administración de la aplicación que le permite administrar aplicaciones en grupos de servidores tan simples como las aplicaciones

de administración en una sola computadora. Proporciona administración de la aplicación, características de escala de software y disponibilidad de misión crítica.

BizTalk Server 2000. Es un producto que proporciona una tecnología de instrumentación avanzada que permite que los analistas de negocios, profesionales de informática y desarrolladores creen procesos de negocios dinámicos que amplíen los límites de las aplicaciones, plataformas y compañía. Incluye facilidades de transformación de datos, auditoria, rastreo y análisis, junto con otras características de seguridad avanzadas con base en la criptografía de clave pública, central para los requerimientos de soluciones B2B.

Commerce Server 2000. Proporciona un marco de trabajo de la aplicación para el desarrollo de soluciones enfocadas en comercio electrónico. Incluye mecanismos de retroalimentación avanzados para soportar procesamiento analítico, el cual le permite desarrollar sitios que mejoren la experiencia del cliente proporcionando contenido personalizado, alentando negocios en línea que desarrollen relaciones más firmes con los socios. Asimismo, incluye administración de productos de servicios, procesamiento de transacción y mercadotecnia destinada y comercialización para soluciones efectivas de negocio a consumidor y de negocio a negocio.

Exchange Server 2000. Además de proporcionar un sistema de mensajería de primera clase, también proporciona colaboración para los trabajadores de conocimiento, integración del Web y diseño de la aplicación de flujo de trabajo, y una sola infraestructura y modelo de usuario para trabajar con mensajes, documentos, y aplicaciones. Asimismo, proporciona una infraestructura de comunicaciones para acceder a la información en cualquier momento y en cualquier lugar a través de las tecnologías emergentes tales como comunicación inalámbrica, mensajería unificada, dispositivos manuales y teleconferencia. También ofrece herramientas de flujo de trabajo, un sistema de almacenamiento en Web como sistema de archivo instalable, mensajería unificada e instantánea, servicios de *chat*, conferencia de datos y capacidades de conferencia de audio y video.

Host Integration Server 2000. Este producto proporciona conectividad y aplicación e integración de datos con sistemas *mainframe* y *midrange*. Soporta un *host* de protocolos de red y servicios de red, una experiencia de usuario cliente/servidor para imprimir y acceder archivos. Le permite habilitar a través del Web sus aplicaciones *mainframe* existentes, proporcionando acceso a bases de datos DB2 y archivos planos en *mainframes*, AS/400 y sistemas Unix. También incluye la integración de transacción con CICS de IBM o ambientes de transacción IMS, permitiendo que el trabajo realizado en el *mainframe* se combine con el realizado en la plataforma Windows.

Internet Security y Acceleration (ISA) Server 2000. El producto proporciona una conectividad segura y rápida a Internet, integrando el *firewall* empresarial de niveles múltiples ampliable y una memoria caché Web de alto desempeño escalable. Proporciona protección para las redes de acceso no autorizados, inspecciona tráficos y alerta a los administradores sobre ataques.

Mobile Information Server 2001. Esta es una plataforma para extender el alcance de las aplicaciones empresariales .NET, datos empresariales y contenido intranet, para el usuario

móvil. Une las aplicaciones inalámbricas y dispositivos de múltiples vendedores y portadores inalámbricos. También proporciona las herramientas necesarias para ampliar sus aplicaciones existentes a dispositivos móviles.

SQL Server 2000. Esta es un sistema de administración y análisis de base de datos relacional completo para el desarrollo de comercio electrónico, línea de negocios y almacén de datos. Cuando el almacén de datos proporciona capacidades que integren, consoliden y resumen la información desde fuentes de datos heterogéneas, las cuales permiten que los tomadores de decisiones consoliden los datos necesarios para utilizar sus herramientas de análisis para evaluar los patrones y tendencias, esperen el comportamiento futuro y tomen mejores decisiones de negocios. Para soportar el análisis de datos que proporcionan las herramientas de minería de datos, servicios de procesamiento analítico en línea (OLAP), mejoras de seguridad y la posibilidad de acceder y vincular a cubos desde clientes que utilizan el explorador (a través de Internet o Intranet). Para aplicaciones de comercio electrónico y de línea de negocios, soporta los estándares de Internet tales como XML.

SharePoint Portal Server 2001. Es una solución flexible y de portal completa que le permite buscar, compartir y publicar fácilmente la información. Este servidor proporciona capacidades de administración de conocimiento, integración sin problemas con *Office* y *Windows*. Esta integración ayuda a realizar sólida administración de documentos, incluyendo búsquedas y suscripciones. También soporta la adición de discusiones en línea en sus procesos de administración de documentos y colaboración

3.1.1.4 - Software para clientes inteligentes .NET

Otro de los componentes de la plataforma .NET es el software para clientes inteligentes .NET. Este software permite conectar una gran variedad de dispositivos como pueden ser: computadoras personales, computadoras portátiles, estaciones de trabajo, teléfonos inteligentes, computadoras de bolsillo, consolas de video juegos y otros dispositivos que cuenten con el software apropiado de .NET [37].

Entre sus características están las siguientes:

- Permiten acceder a la información del usuario en el formato apropiado, en cualquier momento y lugar.
- Utilizan Servicios Web que estén disponibles en el momento para usarlos en el dispositivo.
- Optimizan de distintas maneras la forma en que la información es presentada y organizada.
- Proporcionan una interfaz sencilla y cómoda para que el usuario acceda a la información que necesite con el perfil que se quiera.
- Pueden reconocer la presencia de otros dispositivos e intercambiar información.
- Pueden adaptarse a las características de la red donde están, como podría ser la velocidad de transmisión.

- Tienen capacidad de procesamiento propio y distribuyen el procesamiento en la red haciendo uso de los servicios Web que estén a su disposición.

3.1.2 – Marco de trabajo .NET

El marco de trabajo .NET (*.NET framework*) es el elemento principal de toda la plataforma .NET el cual sirve para crear y ejecutar aplicaciones, sus principales componentes son el tiempo de ejecución de lenguaje común (CLR) y la biblioteca de clases de .NET.

El CLR es una capa que abstrae los servicios del sistema operativo comportándose como un motor de ejecución donde cada una de las acciones de la aplicación que ejecuta es verificada por el CLR [39].

La biblioteca de clases es el otro componente en donde se pueden encontrar clases que ayudan a escribir aplicaciones de una manera más rápida y fácil, ya que pueden utilizarse para crear interfaces (GUI), acceder a datos, crear multihilos, configuraciones de seguridad, etc. [38].

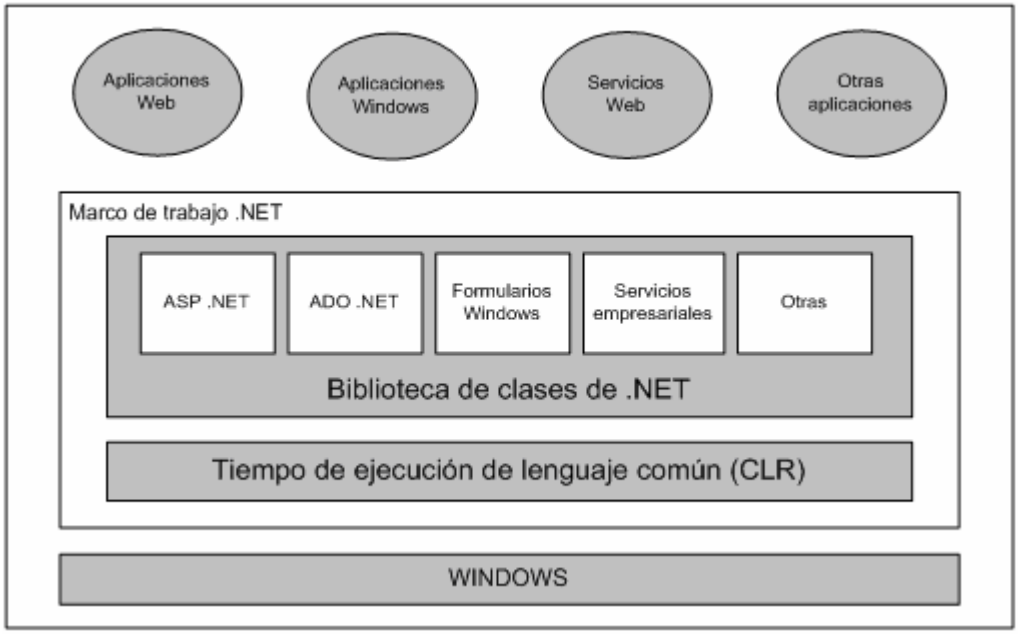


Figura 3.1 – El marco de trabajo .NET se ubica entre las aplicaciones y el sistema operativo

El marco de trabajo .NET puede usarse para crear diferentes tipos de aplicaciones. Entre los diferentes tipos de aplicaciones a elegir están: aplicaciones Web, aplicaciones para Windows, Servicios Web y otros. Debido a que los diferentes tipos de aplicaciones son creadas para ejecutarse sobre el CLR, es posible crear una combinación de ellas [10].

Detrás del marco de trabajo .NET está la norma ECMA-335[2] que define un conjunto de reglas que debe seguir un lenguaje de programación para que sea compatible con la infraestructura de lenguaje común (*common language infrastructure* o CLI) como también se le conoce al marco de trabajo .NET. La norma ECMA-335[3] no sólo define las reglas que debe de cumplir un lenguaje, también define las reglas para el entorno de ejecución común (CLR) y para la biblioteca de clases de .NET.

3.1.2.1 – Tiempo de ejecución de lenguaje común (CLR)

El tiempo de ejecución de lenguaje común (c*ommon language runtime* o CLR) es la base para todo lo demás del marco de trabajo .NET. Toda aplicación que sea creada para el marco de trabajo .NET, es ejecutada por el CLR.

3.1.2.1.1 – Características

Las características que proporciona el CLR para crear aplicaciones son las siguientes [04]:

- Proporciona un desarrollo de aplicaciones más sencillo y rápido, ya que libera al programador de muchas tareas que antes tenía que realizar y que ahora vienen implementadas para que las realice el CLR.
- Permite utilizar componentes que fueron creados en otros lenguajes de una manera más sencilla.
- Tiene un diseño completamente orientado a objetos con las ventajas que esto ofrece a los desarrolladores.
- Todo el código que es ejecutado con el CLR es administrado en tiempo de ejecución, como puede ser la carga del código, la disposición en memoria, recuperación de memoria no utilizada a través de un recolector de memoria, etc.
- Implementa características de gestión a bajo nivel (como administración de memoria).
- Proporciona un sistema común de tipos para todos los lenguajes que se ajusten a las especificaciones del marco de trabajo .NET.
- Proporciona seguridad al código que es ejecutado.

Además facilita la distribución e instalación de aplicaciones, ya que es posible instalar una aplicación simplemente copiando los archivos que la componen en uno de los directorios del equipo en el que se vaya a ejecutar (esto se le conoce como *xcopy*), eliminando los conflictos de versiones entre librerías.

[2] http://www.ecma-international.org/publications/standards/Ecma-335.htm
[3] http://www.ecma-international.org/publications/standards/Ecma-334.htm

3.1.2.1.2 – Componentes

El CLR está compuesto de diferentes componentes que se encargan de realizar diferentes tareas. Los componentes del CLR son los siguientes [07]:

- **Cargador de clases**. Es el encargado de cargar la implementación de un tipo en memoria y prepararlo para la ejecución.
- **MSIL a compiladores nativos**. Convierte el código en el lenguaje intermedio a código nativo.
- **Gestor de código**. Gestiona todo lo referente a la ejecución del código.
- **Recolector de basura**. Administra automáticamente la vida de todos los objetos creados en la ejecución de aplicaciones.
- **Motor de seguridad**. Proporciona una seguridad basada en evidencias en función del origen del código y del usuario.
- **Motor de depuración**. Este componente es el que permite depurar la aplicación y trazar la ejecución del código.
- **Verificador de tipos**. Es el que permite o no la conversión de tipos inseguros y variables no inicializadas. Es posible verificar el lenguaje intermedio para garantizar la seguridad de tipos.
- **Gestor de excepciones**. Proporciona un manejo de excepciones estructurado.
- **Soporte de hilos**. Proporciona clases e interfaces que permiten programación multihilo.
- **COM *marshaler***. Proporciona *marshaling* hacia y desde COM.
- **Soporte de las bibliotecas de clases base**. Integra el código que soporta la biblioteca de clases de .NET.

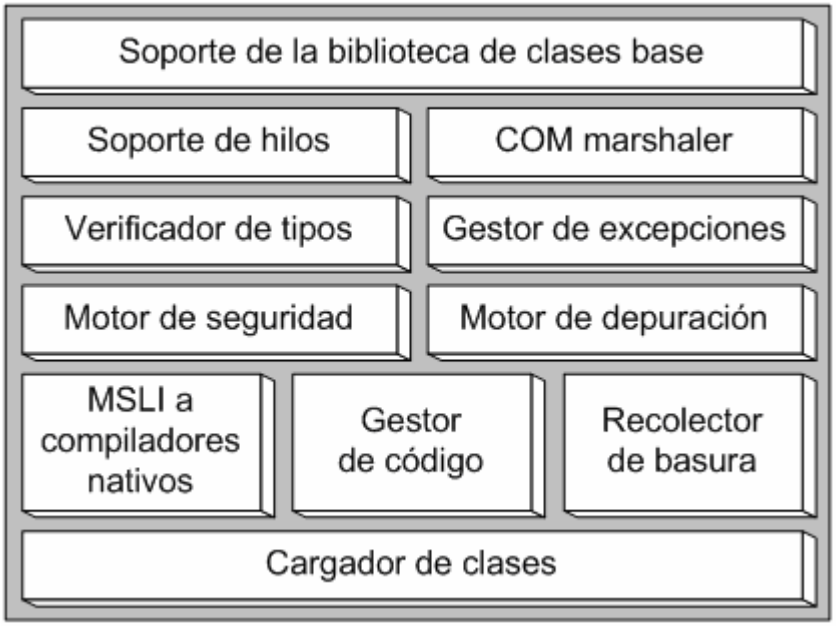


Figura 3.2 – Componentes del CLR

3.1.2.1.3 – Sistema de tipo común (CTS)

El sistema de tipo común (*common type system* o CTS), es la parte del CLR que define el modo en que los tipos serán creados, manipulados y administrados por el CLR en el marco de trabajo .NET.

El CTS permite la integración de código escrito en diferentes lenguajes, optimización del código en ejecución, un modelo de tipos orientado a objeto que soporta múltiples lenguajes, y una serie de directrices que permiten interactuar a los objetos. Además, el CTS permite definir el modo en cómo el código de la aplicación será ejecutado, sin encargarse directamente de su ejecución, es decir, el CTS es el que le indica al CLR cómo debe de ejecutar el código [25].

El CTS clasifica los tipos creados en dos grupos principales, de acuerdo a la forma en que se guardan y manejan en la memoria [08]:

- **Tipos por valor**. Este tipo almacena un dato que puede ser accedido de forma directa, es decir, sin hacer otra operación o manipulación al tipo. Los tipos por valor se organizan a su vez en subgrupos: los tipos de datos nativos o primitivos de .NET, tipos de datos creados por el programador (estructuras) y tipos enumerados.

- **Tipos por referencia**. El tipo por referencia contiene la dirección de memoria en donde esta almacena el dato. Para acceder al dato, se hace de forma indirecta utilizando la dirección de memoria. Los tipos por referencia se agrupan también en varios subgrupos: las clases propias de la biblioteca de .NET, las clases creadas por el programador, interfaces, delegados, etc.

3.1.2.1.4 – Especificación de lenguaje común (CLS)

La especificación de lenguaje común (*common language specification* o CLS) consiste en un conjunto de características comunes, que deben cumplir todos los lenguajes, para integrarse con el CLR. De hecho el CLS es un subconjunto de CTS, por lo tanto los lenguajes tienen que ajustarse también al CTS [04].

El definir un conjunto de características tiene los siguientes objetivos:

- **Independencia del lenguaje**. En la actualidad hay características que un lenguaje de programación no posee y que otros si, por lo cual hay veces que necesitamos utilizar otro lenguaje para realizar alguna tarea específica que necesitamos. Con el CLS esto ya no ocurre, ya que el CLR se encarga de proporcionar la funcionalidad de modo independiente al lenguaje, por lo que podemos escribir nuestras aplicaciones utilizando el lenguaje que generalmente utilizamos, ya que el resultado será el mismo.

- **Integración entre lenguajes**. Es posible escribir bibliotecas de componentes o de clases en un lenguaje y poder utilizarla desde otro lenguaje distinto que también se ajuste a la CLS.

Con todo lo anterior y al ser una especificación abierta es posible agregar nuevos lenguajes para que trabajen con el CLR. Se han anunciado compiladores compatibles con .NET para los siguientes lenguajes de programación:

Tabla 3.1 – Lenguajes compatibles con .NET

APL	Fortran	Pascal
C++	Haskell	Perl
C#	Java	Python
COBOL	JScript	RPG
Component Pascal	Mercury	Scheme
Curriculum	Mondrian	SmallTalk
Eiffel	Oberon	Standard ML
Forth	Oz	Visual Basic

Algunos compiladores de los lenguajes anteriores ya están hechos y los demás siguen en desarrollo.

3.1.2.2 - Biblioteca de Clases de .NET

La biblioteca de clases del marco de trabajo .NET *(.NET framework class library* o .NET FCL) es el segundo componente integrado en el marco de trabajo .NET.

El propósito principal al crear esta biblioteca fue unificar las distintas bibliotecas existentes para cada lenguaje. Cada uno de los lenguajes existentes tiene una biblioteca propia donde existen funciones o clases que permiten hacer manipulación de datos u objetos para desarrollar una aplicación, es decir, en la mayoría de los lenguajes hay una función que nos permite calcular el valor absoluto de un número, mostrar texto en la pantalla, obtener la longitud de una cadena de caracteres, acceder a bases de datos, etc. Todo lo anterior fue conjuntado en una misma biblioteca para que todos los lenguajes que hagan uso del marco de trabajo .NET puedan acceder a esa biblioteca [03].

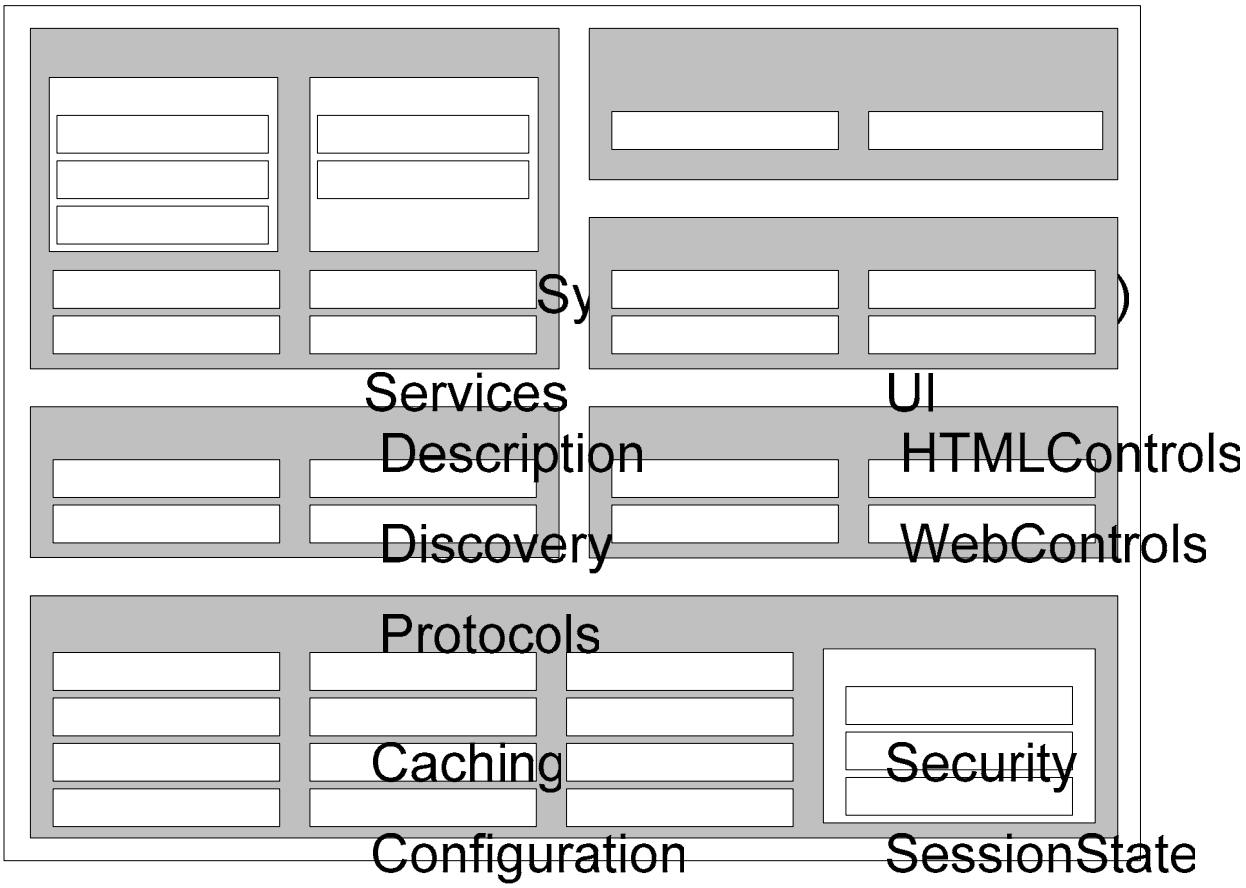


Figura 3.3 – Algunas de los espacios de nombres de la biblioteca de clases .NET

La .NET FCL proporciona el API con el cual se escribe el código administrado. Esta incluye más de 7,000 tipos (clases, estructuras, interfaces, enumeraciones y delegados), desde un tipo tan común como un entero de 32 bits y un tipo especial como una representación de una expresión regular hasta una clase para crear formularios Windows [32].

La .NET FCL esta dividida en aproximadamente en 1,000 espacios de nombres (*namespace*) de modo jerárquico. La descripción de algunos espacios de nombres de la biblioteca de clases de .NET se encuentra en la Tabla 3.2.

Tabla 3.2 – Descripción de algunos espacios de nombres de la biblioteca de clases .NET

Espacio de nombres	Descripción
System	El espacio de nombres *System* es la raíz de muchos de los tipos de bajo nivel requeridos por el marco de trabajo .NET, y es la raíz para los tipos de datos primitivos también. Este espacio de nombres es también la raíz para todos los otros espacios de nombres en la biblioteca de clases .NET.

System.Collections	Este espacio de nombres contiene clases que representan una variedad de diferentes tipos de contenedores, tal como *ArrayList*, *SortedList*, *Queue* y *Stack*. Se puede también encontrar clases abstractas como *CollectionBase*, la cual es útil para implementar nuestras propias colecciones de objetos.
System.ComponentModel	Este espacio de nombres contiene clases involucradas en la creación de componentes y controles, así como para los atributos, convertidores de tipos y vínculos a orígenes de datos.
System.Data	Este contiene clases requeridas para el acceso a bases de datos y su manipulación., así como otros espacios de nombres útiles para el acceso a datos.
System.Data.Common	Este contiene un conjunto de clases que son compartidas por el administrador de proveedores de datos.
System.Data.OleDb	Aquí podemos encontrar clases que nos permiten conectarnos con el proveedor OLE DB a un origen de datos.
System.Data.SqlClient	En este espacio de nombres podemos encontrar clases que están optimizadas para interactuar con SQL Server.
System.Drawing	Este contiene clases que nos proporcionan acceso a la funcionalidad de GDI+ para facilitarnos la creación de gráficos.
System.IO	Aquí podemos encontrar los tipos para manejar archivos.
System.Math	En este espacio de nombres están las funciones matemáticas tal como la extracción de la raíz, funciones trigonométricas y otras funciones matemáticas comunes.
System.Reflection	Este espacio de nombres proporciona soporte para obtener información y creación dinámica de tipos en tiempo de ejecución.
System.Security	Este espacio de nombres es el hogar de tipos referentes con los permisos, criptografía y la seguridad de acceso al código.
System.Threading	Este espacio de nombres contiene clases que facilitan la implementación de aplicaciones multihilos.
System.Windows.Forms	Este espacio de nombres contiene tipos involucrados en la creación de aplicaciones Windows estándar. Las clases que representan formularios y controles residen aquí también.

3.1.3 - Ensamblados

La unidad primaria de una aplicación .NET es el ensamblado (*assembly*). Un ensamblado es una colección lógica de uno o más archivos que contiene el código y los recursos de una aplicación [28].

Los ensamblados contienen código en lenguaje intermedio (i*ntermediate language* o IL) para que sea ejecutado por el CLR. También se le conoce a los ensamblados como ejecutables portables (*portable executable* o PE) cuando tiene un manifiesto asociado.

Los ensamblados crean un límite de seguridad, es decir, es la unidad en donde se solicitan y conceden los permisos, además de crear un límite o ámbito de tipos.

A un ensamblado se le puede proporcionar una clave única de identificación, denominada nombre seguro (s*trong name*). Un nombre seguro está compuesto por la identidad del ensamblado (nombre, versión e información de referencia cultural), junto a una clave pública y firma digital con lo que se asegura que el ensamblado sea único a nivel global.

Todos los ensamblados deben disponer de su correspondiente versión, que es almacenada en el manifiesto del ensamblado.

El número de versión consiste en un valor numérico representado bajo el siguiente formato:

<versión principal>.<versión secundaria>.<número de versión de compilación>.<número de revisión>

El número de versión se guarda dentro del manifiesto, junto con el nombre del ensamblado, clave pública, referencias a otros ensamblados, etc., que son utilizados por el CLR para cargar la versión correcta del ensamblado cuando se ejecuta.

Cada una de las partes que componen el número de versión, tiene un valor que define el grado de compatibilidad del ensamblado. Los niveles de compatibilidad del ensamblado son [04]:

- **Incompatible**. Este nivel viene determinado por la versión principal y secundaria, lo que quiere decir, que al realizar un cambio en cualquiera de esos números, el ensamblado se hace incompatible con otras versiones del mismo ensamblado, que difieran en estos números.

- **Posiblemente compatible**. Esta compatibilidad viene determinada por el número de versión de compilación, y significa que una nueva versión del mismo ensamblado, en la que haya un cambio sólo en este número, permite el uso de dicha versión, aunque esto no implica que puedan aparecer ciertas incompatibilidades.

- **Actualización rápida**. Este tipo de compatibilidad se indica mediante el número de revisión, y le indica al CLR que se trata de una actualización de rápida para resolver algunos problemas importantes.

Una de las características más potentes de los ensamblados es que permiten, gracias al CLR, la ejecución lado a lado (s*ide-by-side execution*), esta consiste en la capacidad tener varias versiones del mismo ensamblado ejecutándose de manera simultánea, en el mismo equipo y en el mismo proceso si fuera necesario. Un aspecto de este tipo de ejecución, consiste en que distintas versiones de un mismo ensamblado no deben mantener una

rigurosa compatibilidad.

3.1.3.1 – Tipos de ensamblados

Los ensamblados pueden clasificarse de acuerdo a la forma en que se crearon, su contenido y su ámbito. Aquí se muestran los tipos que existen [04]:

- **Ensamblado estático**. El ensamblado estático es aquél que se crea en tiempo de diseño, cuando se compila y se construye la aplicación.

- **Ensamblado dinámico**. Es un ensamblado creado en tiempo de ejecución que reside en memoria y se carga en el CLR para proporcionar servicios específicos.

- **Ensamblado de un solo archivo**. Está compuesto por un sólo archivo con extensión .EXE o .DLL, en el que se incluyen todos los recursos que sean necesarios. En este tipo de ensamblado, el manifiesto se encuentra integrado dentro del propio archivo.

- **Ensamblado de múltiples archivos**. Está compuesto por un conjunto de archivos relacionados. A diferencia de los ensamblados de un solo archivo, los ensamblados de múltiples archivos pueden dividirse en un archivo para los recursos, otro para clases personalizadas y otro con el manifiesto y el punto de inicio de la aplicación.

- **Ensamblado privado**. Un ensamblado es privado y a la vez estático si únicamente se accede a él mediante una aplicación específica. El ensamblado sólo es visible dentro del directorio donde está situado.

- **Ensamblado compartido**. Este tipo de ensamblado es utilizado por múltiples aplicaciones, por lo cual debe tener un nombre seguro para no entrar en conflicto con otros ensamblados y ser identificado por el CLR de manera univoca. Un ensamblado dinámico puede además ser compartido. Los ensamblados compartidos se colocan en la caché global de ensamblados (g*lobal assembly cache* o GAC).

3.1.3.2 – Contenido de un ensamblado

Un ensamblado está formado por los siguientes cuatro elementos:

- El manifiesto del ensamblado, que contiene información sobre los elementos que forman el ensamblado.
- Los metadatos sobre los tipos que están contenidos en el ensamblado.
- El código en lenguaje intermedio (IL).
- Un conjunto de recursos adicionales.

Figura 3.4 – Contenido de un ensamblado

3.1.3.3 - Manifiesto del ensamblado

Es un conjunto de datos que permiten saber cómo se relacionan cada uno de los elementos del ensamblado. El manifiesto del ensamblado realiza las siguientes funciones:

- Enumera cada uno de los archivos que componen el ensamblado.
- Controla cómo se asignan las referencias a los tipos y recursos del ensamblado a los archivos que contienen sus declaraciones e implementaciones.
- Almacena los nombres de otros ensamblados de los que depende este ensamblado.
- Proporciona un nivel de direccionamiento indirecto entre los consumidores del ensamblado y los detalles de implementación del ensamblado.
- Convierte al ensamblado en un elemento autodescriptivo.

El manifiesto de un ensamblado contiene la siguiente información:

- **Nombre**. Una cadena con el nombre del ensamblado.
- **Versión**. Número de versión.
- **Cultura**. Información sobre idioma y otros elementos culturales que soporta el ensamblado.
- **Nombre seguro**. En el caso de ensamblados compartidos, este nombre permite identificar al ensamblado a través de una clave.
- **Lista de ficheros**. Los nombres y un resumen de cada uno de los ficheros que forman el ensamblado.
- **Referencia de tipos**. Información que usa el entorno para localizar el código IL de cada uno de los tipos que contiene el ensamblado.
- **Ensamblados referenciados**. Lista de los ensamblados con los que el actual mantiene dependencias.

De los puntos que se acaban de describir, los cuatro primeros forman lo que se denomina la identidad del ensamblado.

Los metadatos que los ensamblados transportan para describirse a ellos mismos en el CLR comunican el código y los datos de clase, el texto, las imágenes, los recursos (como los iconos) y otro tipo de información.

3.1.4 - Seguridad en .NET

La seguridad de una aplicación es algo importante y siempre debe de estar presente en todas las aplicaciones que se desarrollan. En los últimos años se ha trabajado para tener mecanismos de seguridad que permitan crear aplicaciones seguras y la plataforma .NET proporciona varios mecanismos para proteger un sistema.

Los mecanismos que proporcionan la plataforma .NET son los siguientes [33]:

- **Seguridad en las aplicaciones ASP .NET**. Este mecanismo se implementa por medio de la autenticación del usuario, la cual puede ser de tres formas distintas: autenticación por medio de Windows., por medio de formularios y por medio del pasaporte .NET. La autorización es otra forma de implementar la seguridad ya que con ella se puede limitar a los usuarios a realizar ciertas operaciones.

- **Seguridad basada en roles**. Este mecanismo de seguridad se basa en el rol que tiene el usuario para poder tener acceso a los recursos del sistema, es decir, después de que un usuario es autorizado, se revisan los privilegios de autorización a los que tiene derecho para que no pueda realizar operaciones para las que no está autorizado.

- **Servicios criptográficos**. Los servicios criptográficos son clases que proporciona .NET para lograr la confidencialidad, integridad y autenticación de los datos. Para lograr eso se tiene que hacer uso de las primitivas criptográficas (cifrado de clave privada, cifrado de clave pública, firmas digitales y valores *hash*) para crear esquemas de seguridad.

- **Directivas de seguridad**. Las directivas de seguridad son un conjunto configurable de reglas donde se indican que permisos tiene cada ensamblado para realizar ciertos tipos de operaciones. Los permisos se pueden establecer en cuatro niveles: empresa, máquina, usuario y dominio de la aplicación.

- **Seguridad de acceso a código**. En este tipo de mecanismo de seguridad se crean niveles de confianza para el código dependiendo de su origen. El nivel de confianza lo define el administrador o el usuario del sistema.

3.1.5 - Software de código abierto para .NET

La comunidad de código abierto está desarrollando software para trabajar con la plataforma .NET. Para ello están utilizando los estándares ECMA-335 (CLI) y ECMA-334 (especificación del lenguaje C#).

Entre esos proyectos se tienen los siguientes:

- **Proyecto Mono**. Es un proyecto que trabaja en desarrollar su propio marco de trabajo .NET para ejecutarse en otras plataformas aparte de Windows.
- **Proyecto DotGNU**. Este es otro proyecto, similar a Mono, que tiene como objetivo crear una implementación de la plataforma .NET.

3.1.5.1 - Proyecto Mono

El proyecto Mono[4] fue creado para proporcionar un grupo de herramientas de código abierto, basadas en GNU/Linux y ser compatibles con .NET según lo especificado por los estándares ECMA-335 y ECMA-334.

El proyecto Mono fue iniciado por Miguel de Icaza de la empresa Ximian/Novell, para aprovechar todo el dinero y tiempo de investigación que invirtió Microsoft en .NET, además de todos los recursos y documentación[5].

Entre los componentes y herramientas que Mono tiene para desarrollar aplicaciones están:

- Una máquina virtual para interpretar código en lenguaje intermedio.
- Implementación de la biblioteca de clases .NET.
- Compiladores para C#, Visual Basic .NET y JavaScript para generar lenguaje intermedio.

Mono proporciona su máquina virtual para Linux, Solaris, Mac OS X, Windows y Unix.

Para crear aplicaciones con interfaz gráfica, Mono soporta varias tecnologías como GTK#, WinForms y CocoaSharp. Además Mono proporciona un servidor Web para ejecutar aplicaciones hechas en ASP. NET. Así como un módulo para el servidor Web Apache para poder ejecutar ese tipo de aplicaciones.

Para desarrollar aplicaciones en Mono existe el IDE MonoDevelop[6] que se utiliza en sistemas operativos como Linux y Unix. Para crear aplicaciones en Windows utilizando Mono, SharpDevelop[7] es el IDE que se puede utilizar.

[4] http://www.mono-project.com
[5] http://www.somoslibres.org/modules.php?name=News&file=article&sid=269
[6] http://www.monodevelop.org
[7] http://www.sharpdevelop.com

3.1.5.2 - Proyecto DotGNU

DotGNU[8] es un proyecto de código abierto para la plataforma .NET, la meta del proyecto es desarrollar un conjunto de herramientas para crear aplicaciones que se ejecuten en .NET. Este proyecto se basa en las especificaciones ECMA-335 (CLI) y ECMA-334 (C#), además de utilizar otras tecnologías existentes.

El proyecto DotGNU consta de las siguientes partes:

Portable.NET. Es un conjunto de herramientas para compilar y ejecutar aplicaciones para la CLI, además de buscar la compatibilidad con la implementación de Microsoft. Existen compiladores para C# y C, además de estar trabajando en compiladores para Java y Visual Basic. Las plataformas en la inicial para la que se desarrolló fue GNU/Linux, Windows, NetBSD, FreeBSD, Solaris, y Mac OS X, entre otros. Además de ejecutarse en una gran variedad de arquitecturas: x86, PPC, ARM, Sparc, s390, Alpha, ia-64, y PARISC.

PhpGroupWare. Es un conjunto de aplicaciones basadas en Web que proporcionan servicios como manejo de calendario, correo electrónico, agenda, notas, libreta de direcciones, etc. La meta es crear alternativas GNU de los servicios que ofrece Microsoft.

Portable.NET tiene muchas semejanzas con el proyecto Mono de Ximian, ya que ambos intentan proporcionar una implementación alternativa a la tecnología .NET de Microsoft utilizando los estándares ECMA publicados (ECMA-334 y ECMA-335)

[8] http://www.gnu.org/software/dotgnu/

3.2 – SQL SERVER 2000

SQL Server 2000 es un sistema manejador de bases de datos relacionales (*Relational Database Management System* o RDBMS), que almacena datos en múltiples tablas relacionadas. Un RDBMS consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos datos [34].

Un RDBMS incluye bases de datos, motor de base de datos y una aplicación que es necesaria para manejar los datos y los demás componentes del RDBMS. El RDBMS organiza los datos en registros y columnas dentro de la base de datos. Además es responsable de implementar la estructura de la base de datos.

Entre las tareas que realiza están las siguientes:

- Asegurarse que los datos son almacenados correctamente.
- Mantener las relaciones entre los datos en la base de datos.
- Asegurarse de que las reglas definidas para las relaciones entre los datos se cumplan.
- Recuperar todos los datos a un punto de consistencia conocido en caso de fallas

3.2.1 - Características

SQL Server 2000 tiene características que lo hacen una buena opción para implementar una base de datos. Algunas de las características de SQL Server son las siguientes[9]:

Facilidad de instalación, distribución y uso. SQL Server 2000 tiene un conjunto de herramientas que facilitan de gran manera su instalación, distribución y uso.

Compatibilidad con XML. SQL Server 2000 tiene un fuerte soporte a XML que permite recuperar y escribir datos XML, además de permitir la utilización de consultas XPath y URL.

Análisis habilitado para el Web. Tiene la capacidad de analizar datos desde cubos OLAP remotos por medio HTTP o HTTPS.

Múltiples instancias. Se pueden crear múltiples instancias que permiten tener diferentes aplicaciones de negocios en un mismo equipo servidor.

Seguridad. La seguridad en SQL Server 2000 es alta debido a su fuerte integración con la familia de sistemas operativos de la familia de Windows 2000, además de su seguridad basada en funciones y el cifrado de archivos y de la red.

Alta disponibilidad. SQL Server 2000 tiene una alta disponibilidad debido a su capacidad de hacer copias de seguridad en línea, mantenimiento en línea, recuperación automática y la capacidad de instalarlo como un clúster de conmutación por error.

[9] http://www.microsoft.com/latam/sql/evaluation/features/default.asp

Escalabilidad. La escalabilidad se logra gracias a la gran integración que tiene con la familia de sistemas operativos Windows NT y Windows 2000, ya que puede soportar hasta 64 GB en RAM y 32 microprocesadores instalado en Windows 2000 Datacenter.

Almacenes de datos. SQL Server 2000 puede soportar almacenes de datos y mercados de datos para posteriormente utilizar herramientas que ayudan a extraer y analizar datos.

3.2.2 - Arquitectura de SQL Server 2000

La arquitectura de SQL Server 2000 es muy diversa y compleja. Para facilitar la comprensión de todo esto se puede dividir la arquitectura en cinco categorías diferentes:

- Arquitectura de la base de datos
- Arquitectura del motor de base de datos relacional
- Arquitectura de administración
- Arquitectura del almacén de datos y el procesamiento analítico en línea
- Arquitectura del desarrollo de aplicaciones

3.2.2.1 - Arquitectura de la base de datos

La base de datos es el bloque principal de construcción de SQL Server 2000. Este consiste de componentes lógicos (como tablas, vistas y procedimientos almacenados) y también de componentes físicos, que consisten principalmente de los archivos con que está hecha la base de datos y que son almacenados en un disco.

3.2.2.1.1 – Componentes lógicos de la base de datos

Los componentes lógicos de la base de datos consisten de los siguientes objetos:

- **Tablas (*tables*)**. Si la base de datos es el bloque principal de construcción de SQL Server 2000, entonces las tablas es el bloque principal de construcción de la base de datos. Esta estructura almacena los datos en columnas (conocidas como atributos) y filas (conocidas como registros). Una base de datos relacional consiste de más de una tabla, muchas de las tablas están relacionadas de alguna manera. Típicamente, estas tablas representan modelos de entidades del entorno del negocio que necesitan almacenarse y recuperarse, para su utilización posterior [12].

- **Tipos de datos (*data types*)**. Los tipos de datos especifican la forma en que los datos pueden ser almacenados en columnas de una tabla [17]. Además de la gran variedad de tipos de datos que proporciona SQL Server 2000 se pueden crear tipos de datos definidos por el usuario.

- **Vistas (*Views*)**. Las vistas son una forma de presentar datos al usuario de una o más tablas. Generalmente, las vistas se utilizan para restringir las columnas que se desea que vean algunos usuarios por razones de seguridad [12].

- **Procedimientos almacenados (*stored procedures*)**. Los procedimientos almacenados son un conjunto de sentencias en Transact-SQL. Este conjunto de sentencias realizan tareas que las aplicaciones necesitan procesar frecuentemente. SQL Server 2000 proporciona muchos procedimientos almacenados preescritos para realizar una variedad de tareas administrativas, estos procedimientos almacenados son conocidos como procedimientos almacenados del sistema [13].

- **Funciones (*functions*)**. Las funciones ayudan a manipular datos. SQL Server 2000 proporciona funciones del sistema, además de las funciones que pueden ser definidas por el usuario a través de Transact-SQL. Los tres tipos de funciones soportadas por SQL Server son las siguientes:

    - **Funciones de conjuntos de filas**. Estas funciones devuelven un objeto que puede ser usado en lugar de una tabla en una sentencia Transact-SQL.
    - **Funciones agregadas**. Estas analizan una colección de valores y devuelven uno solo, sintetizando un valor.
    - **Funciones escalares**. Estas examinan un solo valor y retornan un solo valor.

- **Índices (*indexes*)**. Un índice ayuda a facilitar y devolver valores de una forma eficiente de las tablas en una base de datos. SQL Server 2000 soporta dos tipos de índices en las tablas [13]:

    - **Índices agrupados (*clusteres indexes*)**. Este tipo de índices ordena los datos en una tabla basándose sólo en un valor clave. Debido a que las filas son almacenadas en orden, la recuperación de datos puede ser muy eficiente. También debido a este tipo de ordenamiento especial, solo se permite un índice agrupado por tabla.
    - **Índices no agrupados (*nonclusteres indexes*)**. Estos índices no almacenan los datos en un orden en especial, en lugar de eso utilizan apuntadores para recuperar rápidamente la información específica. SQL soporta múltiples índices no agrupados por tabla.

- **Restricciones (*constraints*)**. Las restricciones ayudan a mantener la integridad de los datos almacenados en una base de datos. Se pueden asignar restricciones a las tablas para poder asegurar que sólo datos válidos sean introducidos en las columnas seleccionadas [17].

- **Reglas (*rules)*.** [16]Las reglas realizan esencialmente la misma función que las restricciones. Son proporcionadas por SQL Server 2000 solo por compatibilidad con versiones anteriores.

- **Valores por defecto (*defaults*)**. Los valores por defecto son valores que se les asigna a los campos si el usuario no específica un valor. [13]

- **Disparadores (*triggers*)**. Los disparadores son una clase especial de procedimientos almacenados. Estos procedimientos almacenados están diseñados para ejecutarse automáticamente cuando una sentencia de actualización, inserción o borrado se realiza en una tabla o vista. Los disparadores ayudan a asegurar la integridad de los datos almacenados en una base de datos y también pueden implementar las reglas de negocios requeridas por la empresa que necesita la base de datos [16].

3.2.2.1.2 – Componentes físicos de la base de datos

La arquitectura física de la base de datos en SQL Server 2000 es muy importante para la ejecución y estabilidad de las bases de datos en SQL Server 2000. Cada base de datos en SQL Server 2000 almacena los datos usando dos o más archivos del sistema operativo. Estos archivos consisten de lo siguiente [16]:

- **Archivos primarios de datos**. Cada base de datos tiene un archivo primario de datos (archivo con extensión **.mdf**). Uno de los trabajos del archivo primario de datos, además de almacenar datos, es apuntar a los además archivos del sistema operativo con los que forma la base de datos.

- **Archivos secundarios de datos**. Los archivos secundarios de datos (archivos con extensión **.ndf**) son opcionales y almacenan datos adicionales para la base de datos. Las bases de datos pueden ser hechas de cualquier número de archivos de datos secundarios. Esto depende de la estrategia del sistema de archivos que quiera usar el administrador.

- **Archivos de bitácora**. Los archivos de bitácora (archivos es **.ldf**) almacenan la bitácora de las transacciones de una base de datos. La bitácora de las transacciones permiten restaurar a una base de datos a un punto particular en el tiempo. Al menos debe de haber un archivo de bitácora para una base de datos.

Los administradores deben de agrupar estos archivos para hacer una base de datos en estructuras llamadas grupos de archivos. Los grupos de archivos proporcionan adicionalmente opciones de administración para el almacenamiento de datos en SQL Server 2000.

3.2.2.1.3 – Bases de datos del sistema

SQL Server 2000 tiene cuatro bases de datos del sistema (**master**, **tempdb**, **msdb y model**). SQL Server 2000 también proporciona bases de datos de usuario, pero estas bases de datos son creadas por los administradores y desarrolladores y proporcionan un lugar de almacenamiento para los datos en SQL Server.

Las cuatro bases de datos del sistema de SQL Server 2000 son importantes para la operación del servidor. Cada una de estas bases de datos del sistema realiza una función específica, como las siguientes [16]:

- **master**. Esta base de datos guarda toda la información a nivel del sistema para SQL Server, entre esta información incluye todas las cuentas de usuario y las opciones de configuración del sistema. Además es también responsable de rastrear la información de todos los usuarios de las bases de datos en el sistema. Es muy importante tener un respaldo de esta base de datos para tenerla disponible en cualquier momento.

- **tempdb**. Esta base de datos almacena datos que temporalmente SQL Server crea durante algunas operaciones con las bases de datos. Esto podría incluir tablas temporales o procedimientos almacenados. Para asegurarse que **tempdb** queda limpia, la información creada temporalmente cuando un usuario se conecta, es borrada cuando el usuario se desconecta de SQL Server. También la base de datos **tempdb** se vuelve a crear cada vez que se reinicia el sistema.

- **msdb**. Esta base de datos almacena información de las alertas y trabajos programados. También almacena información acerca de los operadores quienes reciben notificaciones relativas a estos trabajos. Esta base de datos del sistema es crítica para las tareas administrativas automáticas en SQL Server.

- **model**. Esta base de datos sirve como una plantilla o molde para la creación de una nueva base de datos de usuario en el sistema. Si se necesita crear sus bases de datos con ciertos parámetros podría modificar la base de datos **model** para que cada vez que necesite crear una nueva base de datos no necesite modificar los parámetros, sino ya tenerlos modificados desde esta base de datos plantilla.

3.2.2.2 - Arquitectura del motor de base de datos relacional

El motor de la base de datos relacional es de lo más poderoso en SQL Server. Examinar exactamente como la arquitectura funciona es muy importante y esto puede asegurar que un administrador verdaderamente comprenda los procesos en SQL Server.

3.2.2.2.1 - Flujo de datos tabular

Los clientes de un sistema SQL Server 2000 envían sentencias SQL al servidor usando un protocolo a nivel de aplicación llamado flujo de datos tabular (t*abular data stream* o TDS). El proceso por el cual se realiza el envío de sentencias es el siguiente: el proveedor OLE DB, el controlador ODBC o la biblioteca de enlace dinámico DB-Library crean paquetes y los pasan al cliente NET-Library de SQL Server, para que posteriormente el cliente Net-Library encapsula estos paquetes en paquetes del protocolo de red, después un servidor NET-Library recibe los paquetes del protocolo de red y extrae los paquetes TDS. El

servidor NET-Library pasa estos paquetes TDS al motor relacional para que los procese. Para devolver los resultados el proceso es el mismo pero en el orden inverso [13].

El formato de los paquetes TDS depende de la opción especificada en la sentencia Transact-SQL transmitida al motor de la base de datos. La cláusula FOR XML indica que el resultado va a ser devuelto con un formato en XML. Si no se especifica FOR XML, el motor de la base de datos envía de regreso un conjunto de resultados relacionados.

3.2.2.2.2 – Motor de la base de datos relacional

El motor de la base de datos relacional de SQL Server 2000 consiste principalmente de dos partes: el motor relacional y el motor de almacenamiento. Estos componentes están separados y usan la API OLE DB para comunicarse.

El motor relacional es el que acepta las sentencias y las compila en un plan de ejecución optimizado. Cuando el motor relacional ejecuta una serie de pasos siguiendo un plan de ejecución, utiliza al motor de almacenamiento para la recuperación de datos. Entre las responsabilidades del motor de almacenamiento están las siguientes [13]:

- Manejar los archivos donde la base de datos está almacenada y el uso del espacio en los archivos.

- Manejar el almacenamiento temporal de datos, además de todas las entradas y salidas a los archivos físicos.

- Manejar las transacciones y usar el bloqueo para controlar el acceso concurrente de usuarios a las filas de la base de datos.

- Implementar funciones de utilidad tal como las sentencias BACKUP, RESTORE y DBCC y la copia masiva de datos.

3.2.2.3 - Arquitectura de administración

SQL Server 2000 proporciona una robusta arquitectura que permite niveles de automatización y simplicidad en la administración. Los principales componentes de la arquitectura de administración de SQL Server 2000 son el marco de administración distribuida de SQL (*SQL distributed management framework* o SQL-DMF), Transact-SQL, componentes de administración automatizada, respaldo y restauración, e importación y exportación.

3.2.2.3.1 - El marco de administración distribuida de SQL

El marco de administración distribuida de SQL (*SQL distributed management framework* o SQL-DMF) proporciona un sistema integrado de servicios, componentes y objetos para la administración de SQL Server 2000. Este marco proporciona gran flexibilidad cuando se administra SQL Server 2000.

SQL-DMF tiene tres tipos de aplicaciones que los administradores usan para utilizar el poder de SQL-DMF [16]:

- **Administrador empresarial de SQL Server**. La principal herramienta de interfaz gráfica de usuario para la administración y configuración del servidor.

- **Aplicaciones del modelo de objetos componentes distribuido (d*istributed component object model* o DCOM) y las páginas de servidor activo (*active server pages* o ASP**)**. Estas permiten la creación de interfaces Web personalizadas para la administración especializada de sistemas SQL Server.

- **Aplicaciones y herramientas de vendedores de software independiente**. Permite a los vendedores de software a desarrollar aplicaciones administrativas para varias ediciones de SQL Server 2000.

Hay varias interfaces de programación de aplicaciones que están disponibles para proporcionar un fácil acceso a la administración clave del sistema operativo y servicios de SQL Server. Una de las más importantes son los objetos de administración distribuida SQL (*SQL distributed management objects* o SQL-DMO). SQL-DMO proporciona interfaces de objetos para SQL Server 2000 y sus componentes. Estos objetos SQL-DMO proporcionan propiedades y métodos y son fácilmente manipulados usando Visual Basic o Visual C++ además de componentes y lenguajes de .NET.

3.2.2.3.2 – Transact-SQL

SQL Server 2000 utiliza un lenguaje de alto nivel para realizar ciertas operaciones por medio de código. Este lenguaje es Transact-SQL, que lo podemos dividir en tres categorías [13]:

- **Lenguaje de definición de datos (*data definition l*anguage o DDL).** Podemos usar las sentencias del lenguaje de definición de datos para crear y manejar todos los objetos de una base de datos SQL SERVER. Las sentencias usualmente utilizadas son CREATE, ALTER y DROP.

- **Lenguaje de manipulación de datos (*data manipulation language* o DML)**. Las sentencias del lenguaje de manipulación de datos nos permiten seleccionar, insertar, actualizar y borrar datos de los objetos definidos usando DDL. Las sentencias que incluyen el DML son SELECT, INSERT. UPDATE y DELETE.

- **Lenguaje de control de datos (d*ata control language* o DCL)**. Las sentencias del lenguaje de control de datos proporcionan un mecanismo de seguridad para los objetos en SQL Server 2000. Dentro de esta categoría están GRANT, REVOKE y DENY.

3.2.2.3.3 – Administración automatizada

SQL Server 2000 permite una administración automática y simple en el servidor. Dentro de las tareas de administración que se pueden automatizar está la de hacer respaldos, debido a que los respaldos se hacen de manera periódica.

Las características de la arquitectura que hacen posible la automatización son [16]:

- **Agente de SQL Server (*SQL Server agent*)**. La clave para la automatización en SQL Server 2000 está en el agente de SQL Server. El agente de SQL Server maneja la ejecución de trabajos creados y alerta a los operadores del sistema. El agente se ejecuta como un servicio en Windows 2000/NT/XP o como un ejecutable en Windows98/Me.

- **Trabajos (*jobs*)**. Se pueden crear trabajos que definen las tareas administrativas que se quieran automatizar. Los trabajos consisten de múltiples pasos y pueden también ejecutar sentencias de Transact-SQL, comandos Windows, ejecutables y hasta secuencia de comandos ActiveX. Después de crear los trabajos se pueden programar para ejecutarse apropiadamente en intervalos apropiados, como podría ser diario, cada semana, cada tercer día, etc.

- **Eventos (*events*) y alertas (*alerts*)**. SQL Server 2000 reporta eventos significativos a la bitácora de eventos del sistema de Windows. Usando alertas se pueden planificar trabajos para ser ejecutados cuando ciertos eventos ocurren en el sistema.

- **Disparadores (*triggers*)**. Como se menciono anteriormente, los disparadores son una clase especial de procedimientos almacenados y con ellos se puede implementar una parte de la lógica de negocios en la base de datos. Los disparadores se incorporan bastante bien con la automatización de SQL Server 2000. Un evento específico puede disparar una alerta que a su vez ejecuta un disparador.

3.2.2.3.4 – Respaldo y restauración

SQL Server 2000 proporciona una arquitectura de respaldo y restauración. Esto es critico para las operaciones que no pueden tolerar cualquier pérdida de datos o interrupción operacional, ya que los respaldos hacen fácil la recuperación ante problemas de seguridad y eventos catastróficos.

La arquitectura de respaldo y restauración de SQL Server 2000 proporciona los siguientes componentes y características [16]:

- **Una variedad de tipos de respaldos**. Respaldos de bases de datos completas, respaldos de bitácoras de transacciones, respaldos diferenciales y respaldos de archivos o grupos de archivos que pueden ser usados para asegurar y tener la correcta estrategia de respaldo y restauración.

- **Control con las sentencias de respaldo y restauración y las herramientas gráficas**. Los administradores tienen una gran variedad de métodos para llevar a cabo respaldos y restauraciones. Las sentencias de respaldo y restauración de Transact-SQL permiten la creación de procedimientos almacenados y/o trabajos, además de que las herramientas gráficas hacen más fácil estas tareas para el mantenimiento de los respaldos y restauraciones.

- **Tablas de historia en la base de datos msdb**. La base de datos del sistema **msdb** graba automáticamente la historia de los respaldos de la base de datos, incluyendo el tipo de respaldo realizado.

- **Respaldos durante la utilización de una base de datos**. La arquitectura de respaldos y restauración de SQL Server 2000 permite al administrador respaldar una base de datos mientras está en uso.

- **Rápida tasa de transferencia**. La característica de rápida transferencia de datos hace posible proporcionar un soporte de respaldo y restauración de muy grandes bases de datos (*very large databases* o VLDB).

- **Creación de bases de datos en la restauración**. La sentencia de Transact-SQL para la restauración tiene la habilidad de crear bases de datos en la ejecución de la sentencia. Esto elimina la necesidad de crear la base de datos con sentencias separadas cuando la base de datos no existe cuando se realiza la restauración.

3.2.2.3.5 – Importación y exportación de datos

SQL Server 2000 también proporciona una arquitectura para la transferencia de datos hacia y desde el RDBMS. Los componentes de esta arquitectura son:

- **Servicios de transformación de datos (d*ata transformation services* o DTS)**. Estos servicios se proporcionan para los datos que están en diferentes lugares y formatos. Las posibilidades que permiten los DTS están las siguientes:

    - Importar y exportar datos entre diferentes orígenes (OLE DB, ODBC o archivos de texto).
    - Transformar los datos cuando son transferidos.
    - Transferencia de objetos de base de datos entre bases de datos de SQL Server.
    - Crear transformaciones personalizadas para integrarlos con otros sistemas.
    - Crear Almacenes de datos (*data warehouse*) y mercados de datos (*data marts*).

- **Replicación (*replication*).** Esto sirve para tener disponibles los datos en el momento y lugar que se necesite. Los numerosos beneficios de la replicación son:

    - Permitir múltiples copias de los mismos datos en diferentes ubicaciones.
    - Permitir gran autonomía a un sitio cuando se hacen modificaciones a los datos localmente y después se propagan los cambios a otras bases de datos configuradas en el proceso de replicación.
    - Mejora adicionalmente la realización de lecturas de datos.
    - Traer los datos más cercanos a los usuarios y de ese modo reducir los conflictos cuando múltiples usuarios están accediendo a los datos.
    - Separar los datos que están siendo explorados por los usuarios.
    - Usar la replicación como parte de su estrategia de servidor suplente.

- **Copia masiva (*Bulk copying*).** La copia masiva permite la transferencia eficiente de grandes cantidades de datos. Las copias masivas transfieren datos de una tabla hacia otra al mismo tiempo. La copia masiva permite las siguientes transferencias de copia masiva:

    - Desde una tabla o vista hasta otra tabla o vista de SQL Server 2000.
    - Desde una tabla o vista a un archivo de datos ya sea como un archivo de texto o un archivo delimitado por tabuladores.
    - El conjunto de resultados de una consulta en una tabla, vista o archivo de datos.
    - El contenido de un archivo de datos a una tabla o una vista.

3.2.2.4 - Arquitectura del almacén de datos y el procesamiento analítico en línea

SQL Server 2000 proporciona una arquitectura para la creación de almacenes de datos (*data warehouse*) y mercados de datos (*data marts*). Estas estructuras se proporcionan para intensas actividades de datos y reportes, incluyendo análisis de tendencias. A ese tipo de sistemas se le conoce como un sistema de procesamiento analítico en línea (*Online Analytical Processing* u OLAP). Esto en contraste a un sistema de procesamiento de transacciones en línea (*Online Transaction Processing* u OLTP) que está diseñado para procesar transacciones de una manera rápida.

Para transformar datos que están en sistemas OLTP a sistemas OLAP SQL Server tiene componentes que logran ese objetivo. Entre esos componentes están los siguientes [16]:

- **XML y OLE DB**. Estos proporcionan un medio de comunicación entre los distintos sistemas que almacenan datos en SQL Server 2000

- **Motor de base de datos relacional de SQL Server 2000**. Es el encargado de transformar, almacenar y administrar datos OLTP y los que están en los almacenes de datos y mercados de datos.

- **Servicios de transformación de datos (DTS)**. Los servicios de transformación de datos son los encargados de pasar los datos OLTP a los almacenes de datos, ya que los datos OLTP se almacenan en tablas de relaciones y entidades y los datos que están en un almacén de datos OLAP se almacenan en tablas de dimensiones y hechos.

- **Servicios de análisis (a*nalysis services*).** Es el encargado de generar cubos multidimensionales y permite a los programas de aplicación acceso a los cubos. Los cubos se pueden almacenan en bases de datos relacionales, como estructuras de datos multidimensionales independientes de alto rendimiento o combinaciones híbridas de ambas.

- **Minería de datos (*data mining*)** La tecnología de minería de datos ayuda a analizar datos en bases de datos relacionales o cubos multidimensionales OLAP para descubrir patrones y tendencias para hacer predicciones futuras. También admite algoritmos de minería de datos estándar.

3.2.2.5 - Arquitectura del desarrollo de aplicaciones

La arquitectura de SQL Server 2000 para el desarrollo de aplicaciones proporciona una manera fácil de crear aplicaciones que usan el RDBMS. Los desarrolladores de aplicaciones pueden usar una de las dos tecnologías de lenguajes de programación para acceder a la base de datos desde las aplicaciones: un lenguaje de base de datos (Transac-SQL) o una interfaz de programación de aplicaciones (API).

- **Transact-SQL**. Es un lenguaje que permite programar en SQL Server para ejecutar tareas como declarar y definir variables, ejecutar saltos, ciclos y comprobar errores, además de escribir rutinas reutilizables (funciones y procedimientos almacenados). Transact- SQL extiende el SQL estándar.

- **Interfaz de programación de aplicaciones (*application programming interface* o API)**. SQL Server admiten muchas API que permiten crear aplicaciones cliente. Entre esas API están las siguientes [13]:

    - **ODBC (o*pen database connectivity*)**. Es una API para acceder a bases de datos y es el estándar *de facto* de la industria. SQL Server proporciona una interfaz ODBC de gran rendimiento para los entornos de programación en Windows.

    - **ADO (*activeX data objects*)**. Es una interfaz de objetos de alto nivel que se sitúa encima de OLE DB para proporcionar la misma funcionalidad, generalmente se usa en aplicaciones hechas con Visual Basic y Visual C++.

    - **OLE DB**. Es una API que proporciona una interfaz COM con cualquier origen de datos tabular, esto es, que puede acceder a cualquier origen que pueda ser representado por medio de filas y columnas. OLE DB es una versión orientada a

objetos de ODBC pero más potente y puede acceder a muchos más orígenes de datos que ODBC.

- **DB-Library**. Es una API específica de SQL Server que proporciona todas las macros y funciones necesarias para una aplicación como abrir conexiones, formatear consultas, enviar al servidor y procesar resultados

- **ESQL (e*mbedded SQL for C*)**. Es la API que proporciona SQL Server para que los desarrolladores puedan escribir aplicaciones con código SQL incrustado en el código de una aplicación. Para esto SQL Server proporciona una precompilador de ESQL

También SQL Server 2000 permite la API ADO.NET que ya se mencionó anteriormente.

3.2.3 – Ediciones de SQL Server 2000

SQL Server 2000 está disponible en diferentes versiones para en las ediciones siguientes [13]:

- **SQL Server 2000 Enterprise**. Es la versión de producción que admite todas las características disponibles en SQL Server 2000.

- **SQL Server 2000 Standard**. Esta versión se puede utilizar como un servidor de base de datos para un pequeño grupo de trabajo o departamento en una empresa.

- **SQL Server 2000 Personal**. La utilizan los usuarios requieren un almacén de datos SQL Server local en un equipo cliente.

- **SQL Server 2000 Developer**. Esta versión es la que utilizan los desarrolladores para hacer pruebas cuando desarrollan sus aplicaciones utilizando SQL Server 2000 como su almacén de datos. SQL Server 2000 Developer tiene todas las características de SQL Server 2000 Enterprise pero sólo esta autorizado como sistema de desarrollo y pruebas

- **SQL Server 2000 CE**. Es la versión para ejecutarse en Windows CE se utiliza como almacén de datos en los dispositivos que ejecutan el sistema operativo Windows CE. Tiene la capacidad de duplicar datos con cualquier versión de SQL Server 2000 para mantener los datos de Windows CE sincronizados.

- **SQL Server 2000 Enterprise Evaluation**. Esta es la versión de descarga gratuita de SQL Server 2000 tiene todas las características de la versión Enterprise pero está limitada su utilización a 120 días desde su descarga.

- **SQL Server 2000 Desktop Engine**. En esta versión de SQL sólo se tiene el motor de la base de datos relacional de SQL Server y es la versión que se puede distribuir para

ejecutar aplicaciones. Las características del motor es parecida a las otras versiones de SQL Server, pero el tamaño de sus bases de datos no puede sobrepasar los 2 GB.

CAPÍTULO 4

DESARROLLO DE LA APLICACIÓN

CAPÍTULO 4 – DESARROLLO DE LA APLICACIÓN

4.1 – EXPLORACIÓN

En la etapa de exploración se examinó el entorno y se decidió hacerlo con programación extrema (XP) debido a que el tiempo de desarrollo era de sólo 3 meses y que el sistema era pequeño. Además de ser uno de los objetivos específicos para la realización de esta tesis.

4.1.1 – Requerimientos

Un requerimiento es una característica del sistema o una descripción de algo que el sistema es capaz de hacer con el objeto de satisfacer el propósito del sistema [29]. La obtención de requerimientos es una tarea que, en programación extrema, se debe de realizar por medio de las historias de usuario, pero se decidió hacerlo por medio de casos de uso, ya que estos tienen más tiempo utilizándose y a las personas les parecen una mejor forma de entender y especificar los requisitos de un sistema.

Una de las razones por las que las personas prefieren los casos de uso sobre las historias de usuario es que las historias de usuario las escribe el cliente y muchas veces el cliente no sabe como describir las tareas que realiza.

Los requerimientos del sistema son divididos, generalmente, en dos tipos:

1. Requerimientos funcionales
2. Requerimientos no funcionales

Estos dos tipos de requerimientos son explicados en las siguientes secciones.

4.1.1.1 – Requerimientos funcionales

Un requerimiento funcional describe una interacción entre el sistema y su ambiente [29].

Para el sistema se obtuvieron los siguientes requerimientos funcionales:

- El sistema sólo deberá permitir a la secretaria técnica utilizarlo.
- El sistema permitirá dar de alta, baja y consultar datos de los profesores, además de modificar los datos.
- El sistema permitirá administrar la información de las materias (alta, baja, modificación y consulta).
- También deberá permitir la lata, baja, modificación y consulta de periodos escolares.

- El sistema deberá de ayudarle al usuario a capturar sólo los datos necesarios.
- El sistema permitirá imprimir un reporte con la propuesta de cada profesor.
- La interfaz del sistema deberá ser fácil de usar para reducir el tiempo de capacitación para su utilización.

4.1.1.1.1 – Diagrama de casos de uso

Un caso de uso es, en esencia, una interacción típica entre un usuario y un sistema de cómputo [14]. Los casos de uso ayudan a obtener los requerimientos funcionales de un sistema.

Un actor es un usuario que interactúa con el sistema, jugando un determinado rol o papel. El nombre del actor indica el rol que tiene el usuario.

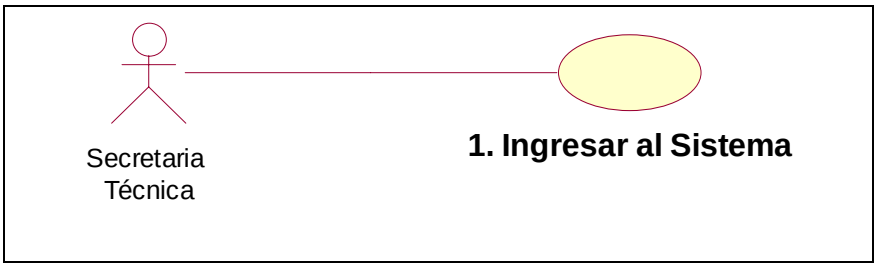


Figura 4.1 – Ejemplo de caso de uso y actor.

En la figura 4.1 se muestra el caso de uso **1. Ingresar al Sistema**, el actor Secretaria Técnica y además una línea que relaciona al actor con el caso de uso. La línea indica que el actor realiza ese caso de uso.

El nombre de un caso de uso debe comenzar por un verbo, ya que el caso de uso representa una tarea o actividad que el actor realiza en el sistema.

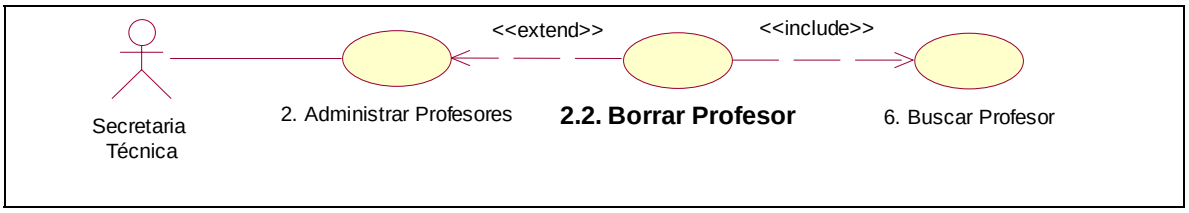


Figura 4.2 – Caso de uso con relaciones de extensión e inclusión.

La figura 4.2 muestra los casos de uso Borrar profesor y Buscar Profesor, que son una extensión (*extend*) y una inclusión (*include*) respectivamente.

Una extensión es un caso de uso que incrementa o extiende el comportamiento de un caso de uso. Las extensiones se pueden ver como casos de uso opcionales. Una inclusión es un caso de uso que tiene características o comportamiento que existe en más de un caso de uso, por lo cual, se agrupa ese comportamiento en un caso de uso aparte.

Para el sistema se obtuvo el siguiente diagrama de casos de uso, en su forma general:

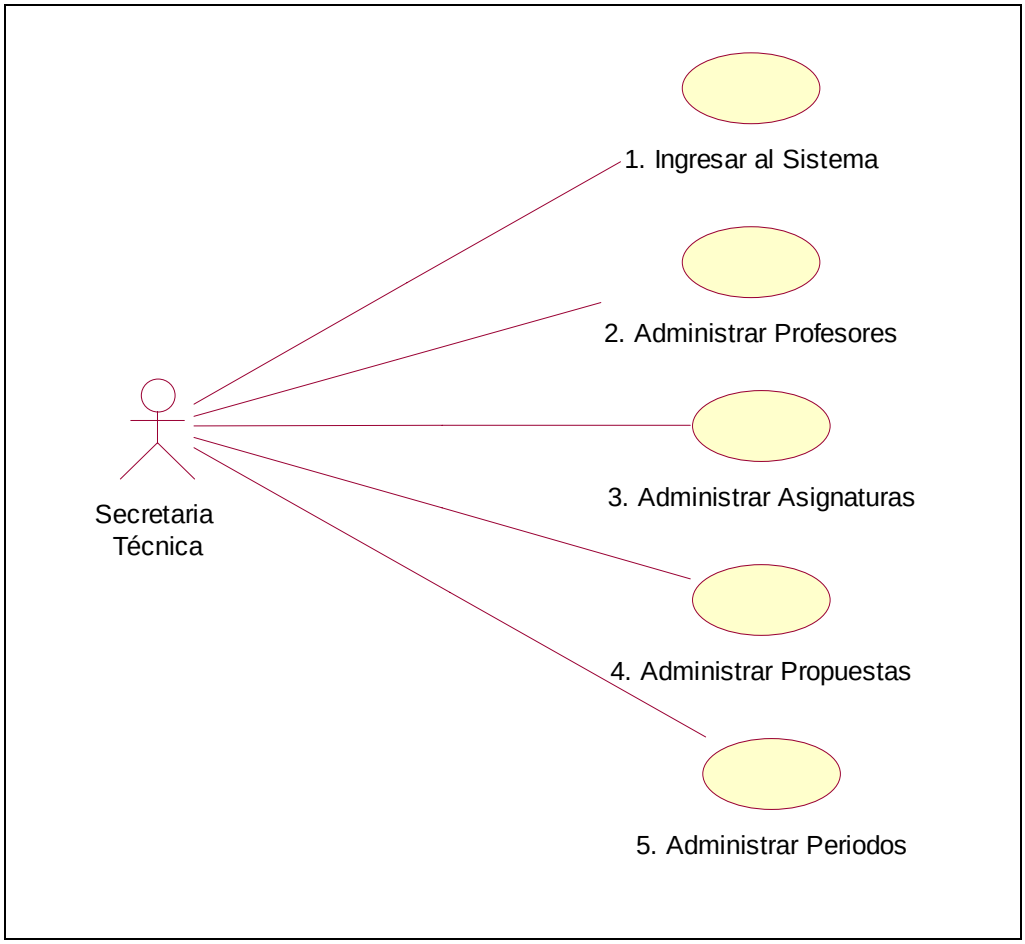


Figura 4.3 – Diagrama general de casos de uso.

El detalle de cada uno de los casos de uso se describirá en la siguiente sección.

4.1.1.1.2 – Detalle de los casos de uso

Para empezar con la explicación del diagrama de casos de uso se describirá primero el actor del sistema.

Para describir al actor del sistema se llena un formato que contiene la siguiente información:

- **Nombre o rol**: Es el nombre o rol que se le da a una o más personas en el Sistema.
- **Tipo**: Es el tipo de actor, el cual puede ser principal o secundario. Un actor principal es aquel que tiene objetivos que se satisfacen con el Sistema mientras que un actor secundario es aquel que proporciona un servicio al Sistema.
- **Descripción**: Es una descripción de lo que hace el actor en el sistema.
- **Casos de uso**: son los casos de uso en los que participa el usuario.

El formato para el actor del sistema es el siguiente:

Actor	
Nombre o rol:	Secretaria Técnica.
Tipo:	Actor principal.
Descripción:	La Secretaria Técnica es la encargada de manejar toda la información que involucra la elaboración de propuestas para profesores.
Casos de uso:	Ingresar al Sistema, Administrar Asignaturas, Administrar Profesores, Administrar Propuestas, Administrar periodos, Buscar Profesor

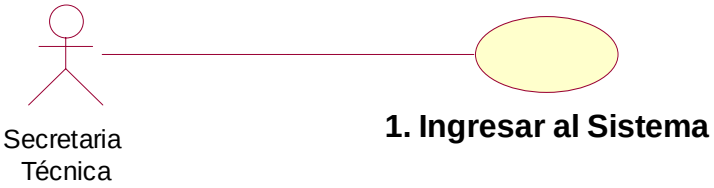
Para los casos de uso se utiliza un formato que tiene la siguiente información [23]:

- **Diagrama del caso de uso**: es el diagrama del caso de uso que se está detallando.
- **Caso de uso**: nombre del caso de uso
- **Tipo**: tipo del caso de uso entre los cuales están el básico, la inclusión y la extensión.
- **Descripción**: es una descripción corta del caso de uso.
- **Actor principal**: es el actor principal del Sistema.
- **Precondiciones**: las precondiciones establecen lo que siempre debe cumplirse antes de comenzar un caso de uso.
- **Poscondiciones**: las poscondiciones establecen qué debe cumplirse cuando el caso de uso se completa con éxito.
- **Elaborado**: es el nombre de quien elaboró el caso de uso.
- **Flujo principal**: es una descripción o una serie de pasos del camino de éxito típico del caso de uso.
- **Flujo alternativo**: son las bifurcaciones o caminos alternos que existen en el caso de uso.
- **Excepciones**: son las excepciones que se pueden presentar al realizar un determinado paso de un caso de uso.

La descripción de cada uno de los casos de uso se presenta a continuación.

4.1.1.1.3 – Caso de uso: 1. Ingresar al Sistema.

El detalle del caso de uso **1. Ingresar al Sistema** se muestra a continuación:

Diagrama del caso de uso


Definición	
Caso de uso:	1. Ingresar al Sistema.
Tipo:	Básico.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Ingresar al Sistema**, describe como la *Secretaria Técnica* se autentica para ingresar al Sistema.
Precondiciones:	Ninguna.
Poscondiciones:	La *Secretaria Técnica* accede al *Sistema*.

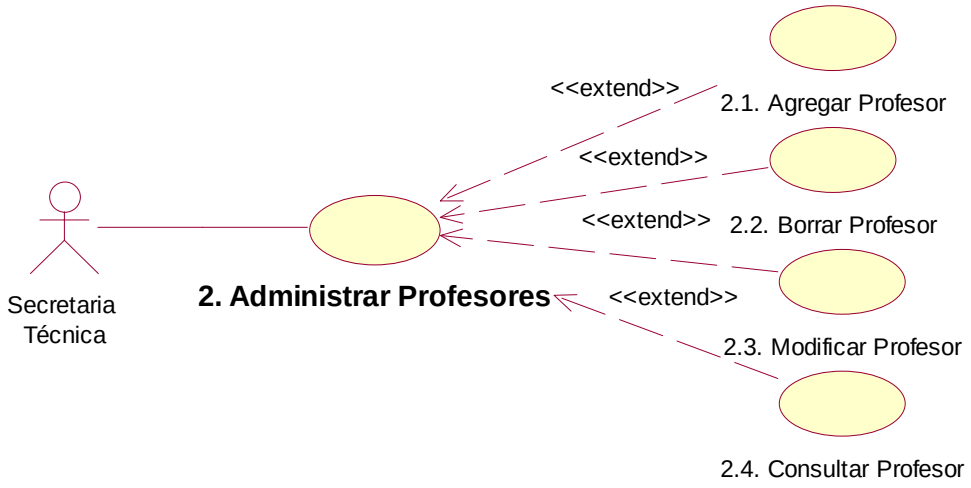
Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* inicia el Sistema.	2.	El *Sistema* pide la contraseña para acceder al Sistema.	E01
3.	La *Secretaría Técnica* introduce la contraseña y pide al Sistema que la verifique.	4.	El *Sistema* verifica que la contraseña proporcionada sea la correcta.	E02
		5.	El *Sistema* permite el acceso.	

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	Sistema no disponible	Ninguna.
E02	Contraseña inválida	El *Sistema* muestra un mensaje indicando que la contraseña es inválida y le indica al usuario que vuelva a intentarlo.

4.1.1.1.4 – Caso de uso: 2. Administrar Profesores.

El detalle del caso de uso **2. Administrar Profesores** se muestra a continuación así como sus casos de uso de extensión e inclusión.

Diagrama del caso de uso



Caso de uso:	2. Administrar Profesores.
Tipo:	Básico.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Administrar Profesores**, describe cómo la *Secretaria Técnica* administra la información de los Profesores.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema*.
Poscondiciones:	La *Secretaria Técnica* podrá elegir entre agregar, borrar, modificar o consultar la información sobre Profesores.

Flujos

Flujo principal:

Actor		Sistema		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* elige la opción Administrar Profesores del *Sistema.*	2.	El *Sistema* muestra las opciones de Agregar, Borrar, Modificar y Consultar Profesores. Además de la opción de Salir.	E01
3.	La *Secretaría Técnica* selecciona una de las opciones mostradas por el *Sistema*.	4.	El *Sistema* procesa la opción elegida**.**	E02

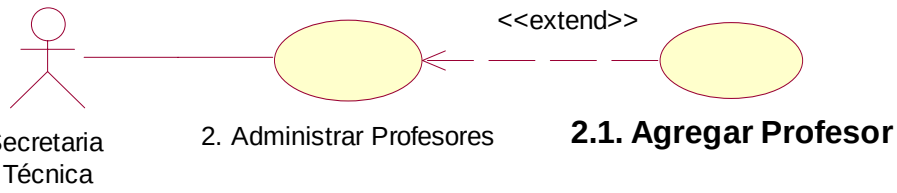
Flujos alternativos:

Actor		Sistema	
Paso	**Acciones**	**Paso**	**Acciones**
3A.1	La *Secretaria Técnica* elige la opción Agregar.	3A.2	El *Sistema* procede de acuerdo al caso de uso **Agregar Profesor**.
3B.1	La *Secretaria Técnica* elige la opción Borrar.	3B.2	El *Sistema* procede de acuerdo al caso de uso **Borrar Profesor**.

3C.1	La *Secretaria Técnica* elige la opción Modificar.	3C.2	El *Sistema* procede de acuerdo al caso de uso **Modificar Profesor**.
3D.1	La *Secretaria Técnica* elige la opción Consultar.	3D.2	El *Sistema* procede de acuerdo al caso de uso **Consultar Profesor**.
3E.1	La *Secretaria Técnica* elige la opción Salir.	3E.2	El *Sistema* sale de la opción Administrar Profesores.

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se pueden mostrar las opciones	EL *Sistema* indica que no es posible mostrar las opciones.

El caso de uso de extensión **2.1. Agregar Profesor** se muestra a continuación:

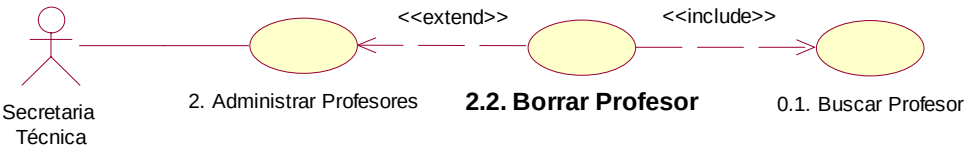
Diagrama del caso de uso


Definición	
Caso de uso:	2.1. Agregar Profesor.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Agregar Profesor**, describe como la *Secretaria Técnica* agrega profesores al *Sistema*.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema* y elegido la opción de Agregar Profesor.
Poscondiciones:	La *Secretaria Técnica* habrá agregado la información de un Profesor al *Sistema*.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* elige Agregar Profesor al *Sistema.*	2.	El *Sistema* pide los datos del Profesor (nombre, apellido paterno, apellido materno, RFC, número de trabajador, domicilio y teléfono).	
3.	La *Secretaría Técnica* introduce los datos del profesor.	4.	El *Sistema* verifica los datos ingresados del Profesor**.**	E01, E02
		5.	El *Sistema* almacena los datos del Profesor**.**	E03
		6.	El *Sistema* indica que los datos se han almacenado**.**	
Flujos alternativos:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	
5A.1	La Secretaria Técnica cancela la operación.	5A.2	El Sistema cancela la operación de agregar.	

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	Datos inválidos	EL *Sistema* indica que los datos que proporcionó el usuario son inválidos y que introduzca datos válidos.
E02	El Profesor ya está registrado	El *Sistema* muestra un mensaje indicando que el profesor ya esta registrado en el *Sistema*.
E03	No se pueden almacenar los datos	El *Sistema* muestra un mensaje indicando que no es posible almacenar los datos.

El caso de uso de extensión **2.2. Borrar Profesor** se muestra a continuación:

Diagrama del caso de uso


Definición	
Caso de uso:	2.2. Borrar Profesor.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica.*

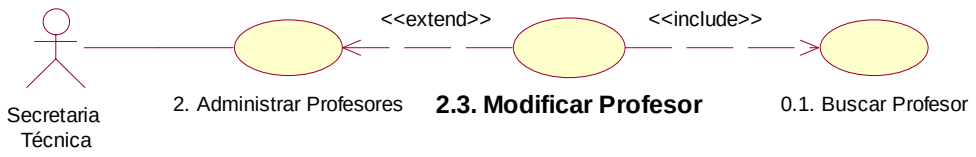
Descripción:	El caso de uso, **Borrar Profesor**, permite a la *Secretaria Técnica* borrar Profesores del *Sistema*.
Precondiciones	La *Secretaria Técnica* debe haber accedido al *Sistema*.
Poscondiciones	La *Secretaria Técnica* habrá eliminado al Profesor elegido.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* busca al Profesor que va a borrar, para buscar al Profesor realiza el caso de uso **Buscar Profesor.**	2.	El *Sistema* muestra la información del Profesor que se seleccionó.	E01
3.	La *Secretaría Técnica* elige borrar al profesor.	4.	El *Sistema* borra la información del Profesor.	E02
	.	5.	El *Sistema* muestra un mensaje indicando que la información del Profesor se ha borrado.	

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se puede mostrar la información del Profesor	EL *Sistema* indica que no se puede mostrar la información del Profesor.
E02	No se puede borrar la información del Profesor	El *Sistema* muestra un mensaje indicando que no se puede borrar la información del Profesor.

El caso de uso de extensión **2.3. Modificar Profesor** se muestra a continuación:

Diagrama del caso de uso



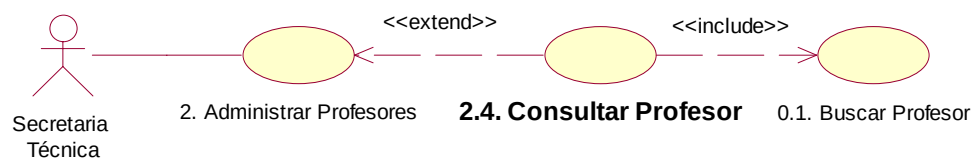
Definición	
Caso de uso:	2.3. Modificar Profesor.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Modificar Profesor**, permite a la *Secretaria Técnica* modificar la información de los Profesores del *Sistema*.

Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema*. El Profesor al que se hará una modificación en su información debe de existir.
Poscondiciones:	La *Secretaria Técnica* habrá modificado la información del Profesor elegido.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* busca al Profesor que va a modificar, para buscar al profesor realiza el caso de uso **Buscar Profesor**	2.	El *Sistema* muestra la información del Profesor que se seleccionó.	E01
3.	La *Secretaría Técnica* elige modificar la información del Profesor.	4.	El *Sistema* permite la modificación de la información del Profesor.	
5.	La Secretaría Técnica modifica la información.			
6.	La Secretaría Técnica elige guardar los cambios.	7.	El *Sistema* valida la información del Profesor.	E02
		8.	El *Sistema* guarda la información del Profesor.	E03
		9.	El *Sistema* muestra un mensaje indicando que la información del Profesor se ha guardado.	
Flujos alternativos:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	
6A.1	La *Secretaria Técnica* cancela los cambios.	6A.2	El *Sistema* cancela los cambios y recupera la información del Profesor almacenada antes de los cambios.	

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se puede mostrar la información del Profesor	EL *Sistema* indica que no se puede mostrar la información del Profesor.
E02	Datos inválidos	EL *Sistema* indica que los datos que proporcionó el usuario son inválidos y pide que introduzca datos válidos.
E03	No se puede guardar la información del Profesor	El *Sistema* muestra un mensaje indicando que no se puede guardar la información del Profesor.

El caso de uso de extensión **2.4. Consultar Profesor** se muestra a continuación:

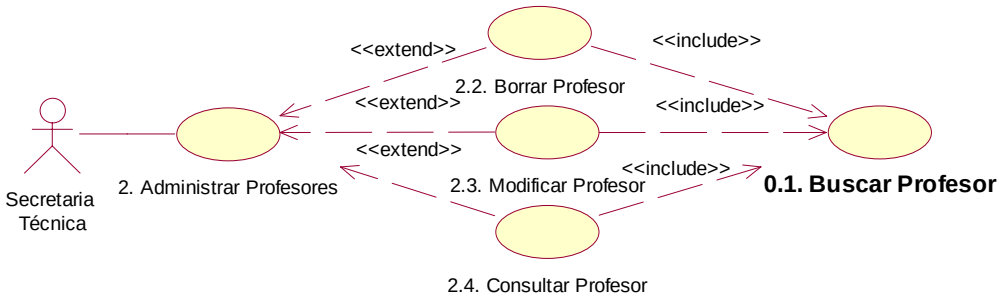
Diagrama del caso de uso


Definición	
Caso de uso:	2.4. Consultar Profesor.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Consultar Profesor**, permite a la *Secretaria Técnica* consultar información de los Profesores registrados en el *Sistema*.
Precondiciones	La *Secretaria Técnica* debe haber accedido al *Sistema*.
Poscondiciones	La *Secretaria Técnica* habrá consultado la información del Profesor elegido.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* busca al Profesor que va a consultar, para buscar al Profesor realiza el caso de uso **Buscar Profesor**	2.	El *Sistema* muestra la información del Profesor que se seleccionó.	E01

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se puede mostrar la información del Profesor	EL *Sistema* indica que no se puede mostrar la información del Profesor.

El caso de uso de inclusión **0.1. Buscar Profesor** se muestra a continuación:

Diagrama del caso de uso


Definición	
Caso de uso:	0.1. Buscar Profesor.
Tipo:	Inclusión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Buscar Profesor**, permite a la *Secretaria Técnica* buscar profesores que existan en el Sistema.
Precondiciones	La *Secretaria Técnica* debe haber accedido al Sistema. El Profesor que se busca debe existir en el Sistema.
Poscondiciones	La *Secretaria Técnica* habrá encontrado al Profesor buscado.

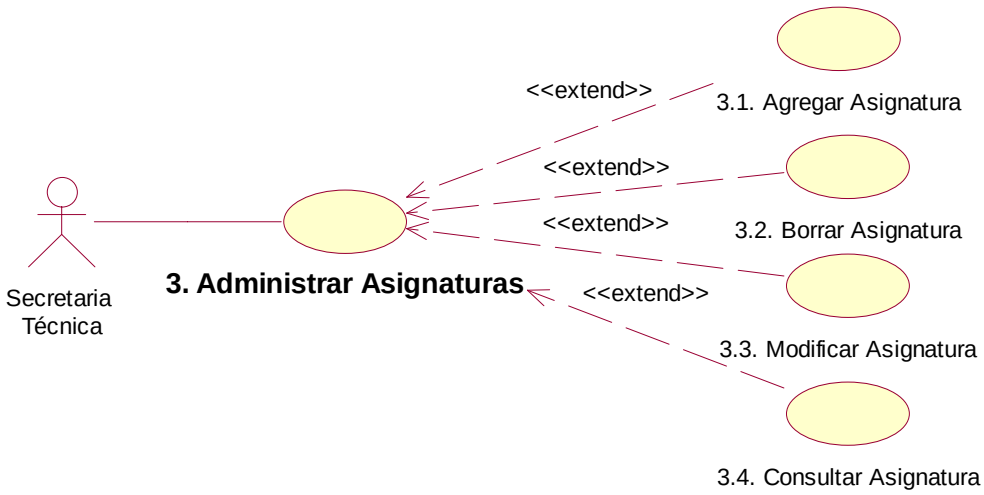
Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* selecciona si va a buscar al profesor por apellido paterno, apellido materno, nombre, RFC o por número de trabajador.	3.	EL *Sistema* muestra los profesores existentes ordenados por la elección de la *Secretaria Técnica*.	E01
3.	La *Secretaría Técnica* elige al Profesor que busca.	4.	El *Sistema* recupera todos los datos del Profesor seleccionado**.**	E02

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se pueden ordenar los datos	EL *Sistema* indica que no es posible ordenar los datos en la forma que se seleccionó.
E02	No se pueden recuperar los datos del Profesor	El *Sistema* muestra un mensaje indicando que no se pueden recuperar los datos del Profesor.

4.1.1.1.5 – Caso de uso: 3. Administrar Asignaturas.

El detalle del caso de uso **3. Administrar Asignaturas** se muestra a continuación así como sus casos de uso de extensión e inclusión.

Diagrama del caso de uso



Caso de uso:	2. Administrar Asignaturas.
Tipo:	Básico.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Administrar Asignaturas**, describe cómo la *Secretaria Técnica* administra la información de los Asignaturas.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema*.
Poscondiciones:	La *Secretaria Técnica* podrá elegir entre agregar, borrar, modificar o consultar la información sobre Asignaturas.

Flujos

Flujo principal:

Actor		Sistema		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* elige la opción Administrar Asignaturas del *Sistema.*	2.	El *Sistema* muestra las opciones de Agregar, Borrar, Modificar y Consultar Asignaturas. Además de la opción de Salir.	E01
3.	La *Secretaría Técnica* selecciona una de las opciones mostradas por el *Sistema*.	4.	El *Sistema* procesa la opción elegida**.**	E02

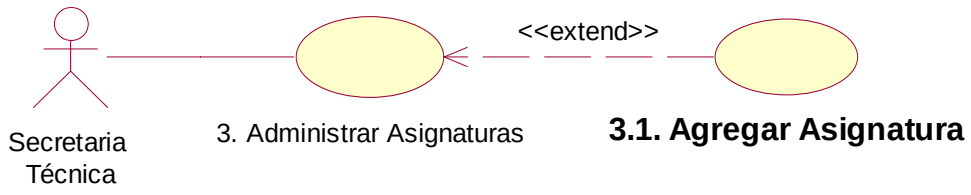
Flujos alternativos:

Actor		Sistema	
Paso	**Acciones**	**Paso**	**Acciones**
3A.1	La *Secretaria Técnica* elige la opción Agregar.	3A.2	El *Sistema* procede de acuerdo al caso de uso **Agregar Asignatura**.
3B.1	La *Secretaria Técnica* elige la opción Borrar.	3B.2	El *Sistema* procede de acuerdo al caso de uso **Borrar Asignatura**.
3C.1	La *Secretaria Técnica* elige la opción Modificar.	3C.2	El *Sistema* procede de acuerdo al caso de uso **Modificar Asignatura**.
3D.1	La *Secretaria Técnica* elige la opción Consultar.	3D.2	El *Sistema* procede de acuerdo al caso de uso **Consultar Asignatura**.
3E.1	La *Secretaria Técnica* elige la opción Salir.	3E.2	El *Sistema* sale de la opción Administrar Asignaturas.

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se pueden mostrar las opciones	EL *Sistema* indica que no es posible mostrar las opciones.

El caso de uso de extensión **2.1. Agregar Asignatura** se muestra a continuación:

Diagrama del caso de uso

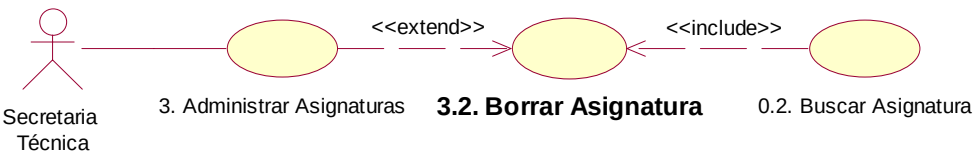


Definición	
Caso de uso:	2.1. Agregar Asignatura.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Agregar Asignatura**, describe como la *Secretaria Técnica* agrega Asignaturas al *Sistema*.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema* y elegido la opción de Agregar Asignatura.
Poscondiciones:	La *Secretaria Técnica* habrá agregado la información de una Asignatura al *Sistema*.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* elige Agregar Asignatura al *Sistema.*	2.	El *Sistema* pide los datos de la Asignatura (clave, nombre, horas de teoría y horas de práctica).	
3.	La *Secretaría Técnica* introduce los datos de la Asignatura.	4.	El *Sistema* verifica los datos ingresados de la Asignatura**.**	E01, E02
		5.	El *Sistema* almacena los datos de la Asignatura**.**	E03
		6.	El *Sistema* indica que los datos se han almacenado**.**	
Flujos alternativos:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	
5A.1	La Secretaria Técnica cancela la operación.	5A.2	El Sistema cancela la operación de agregar.	

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	Datos inválidos	EL *Sistema* indica que los datos que proporcionó el usuario son inválidos y que introduzca datos válidos.
E02	La Asignatura ya está registrado	El *Sistema* muestra un mensaje indicando que la Asignatura ya esta registrada en el *Sistema*.
E03	No se pueden almacenar los datos	El *Sistema* muestra un mensaje indicando que no es posible almacenar los datos.

El caso de uso de extensión **2.2. Borrar Asignatura** se muestra a continuación:

Diagrama del caso de uso


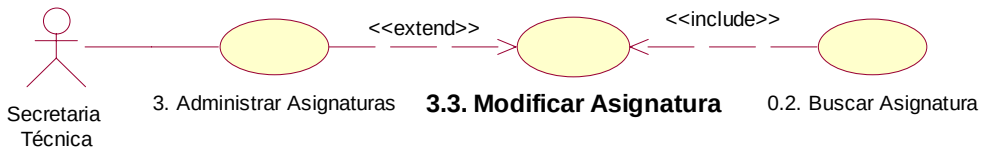
Definición	
Caso de uso:	2.2. Borrar Asignatura.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica.*
Descripción:	El caso de uso, **Borrar Asignatura**, permite a la *Secretaria*

	Técnica borrar Asignaturas del *Sistema*.
Precondiciones	La *Secretaria Técnica* debe haber accedido al *Sistema*.
Poscondiciones	La *Secretaria Técnica* habrá eliminado la Asignatura elegida.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* busca la Asignatura que va a borrar, para buscar la Asignatura realiza el caso de uso **Buscar Asignatura.**	2.	El *Sistema* muestra la información de la Asignatura que se seleccionó.	E01
3.	La *Secretaría Técnica* elige borrar la Asignatura.	4.	El *Sistema* borra la información de la Asignatura.	E02
	.	5.	El *Sistema* muestra un mensaje indicando que la información de la Asignatura se ha borrado.	

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se puede mostrar la información de la Asignatura	EL *Sistema* indica que no se puede mostrar la información de la Asignatura.
E02	No se puede borrar la información de la Asignatura	El *Sistema* muestra un mensaje indicando que no se puede borrar la información de la Asignatura

El caso de uso de extensión **2.3. Modificar Asignatura** se muestra a continuación:

Diagrama del caso de uso


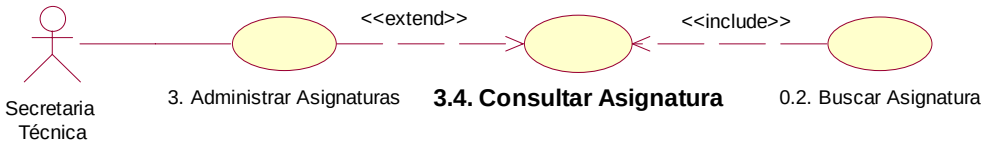
Definición	
Caso de uso:	2.3. Modificar Asignatura.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Modificar Asignatura**, permite a la *Secretaria Técnica* modificar la información de las Asignaturas del *Sistema*.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema*.

	La Asignatura a la que se hará una modificación en su información debe de existir.
Poscondiciones:	La *Secretaria Técnica* habrá modificado la información de la Asignatura elegida.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* busca la Asignatura que va a modificar, para buscar la Asignatura realiza el caso de uso **Buscar Asignatura**	2.	El *Sistema* muestra la información de la Asignatura que se seleccionó.	E01
3.	La *Secretaría Técnica* elige modificar la información de la Asignatura.	4.	El *Sistema* permite la modificación de la información de la Asignatura.	
5.	La Secretaría Técnica modifica la información.			
6.	La Secretaría Técnica elige guardar los cambios.	7.	El *Sistema* valida la información de la Asignatura.	E02
		8.	El *Sistema* guarda la información de la Asignatura.	E03
		9.	El *Sistema* muestra un mensaje indicando que la información de la Asignatura se ha guardado.	
Flujos alternativos:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	
6A.1	La *Secretaria Técnica* cancela los cambios.	6A.2	El *Sistema* cancela los cambios y recupera la información de la Asignatura almacenada antes de los cambios.	

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se puede mostrar la información de la Asignatura	EL *Sistema* indica que no se puede mostrar la información de la Asignatura.
E02	Datos inválidos	EL *Sistema* indica que los datos que proporcionó el usuario son inválidos y pide que introduzca datos válidos.
E03	No se puede guardar la información de la Asignatura	El *Sistema* muestra un mensaje indicando que no se puede guardar la información de la Asignatura.

El caso de uso de extensión **2.4. Consultar Asignatura** se muestra a continuación:

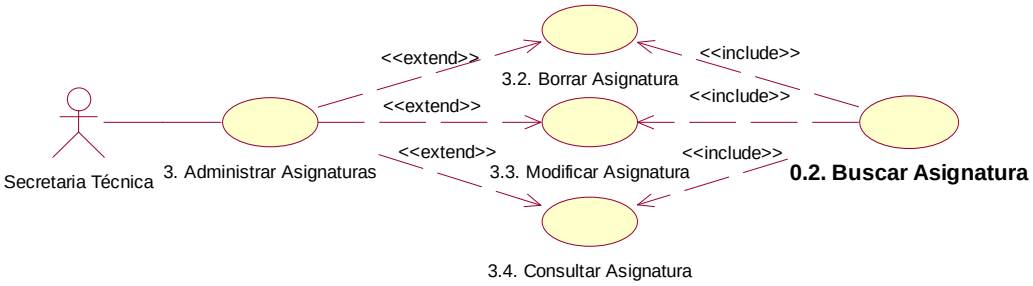
Diagrama del caso de uso


Definición	
Caso de uso:	2.4. Consultar Asignatura.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Consultar Asignatura**, permite a la *Secretaria Técnica* consultar información de las Asignaturas registradas en el *Sistema*.
Precondiciones	La *Secretaria Técnica* debe haber accedido al *Sistema*.
Poscondiciones	La *Secretaria Técnica* habrá consultado la información de la Asignatura elegida.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* busca la Asignatura que va a consultar, para buscar la Asignatura realiza el caso de uso **Buscar Asignatura**	2.	El *Sistema* muestra la información de la Asignatura que se seleccionó.	E01

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se puede mostrar la información de la Asignatura	EL *Sistema* indica que no se puede mostrar la información de la Asignatura.

El caso de uso de inclusión **0.2. Buscar Asignatura** se muestra a continuación:

Diagrama del caso de uso


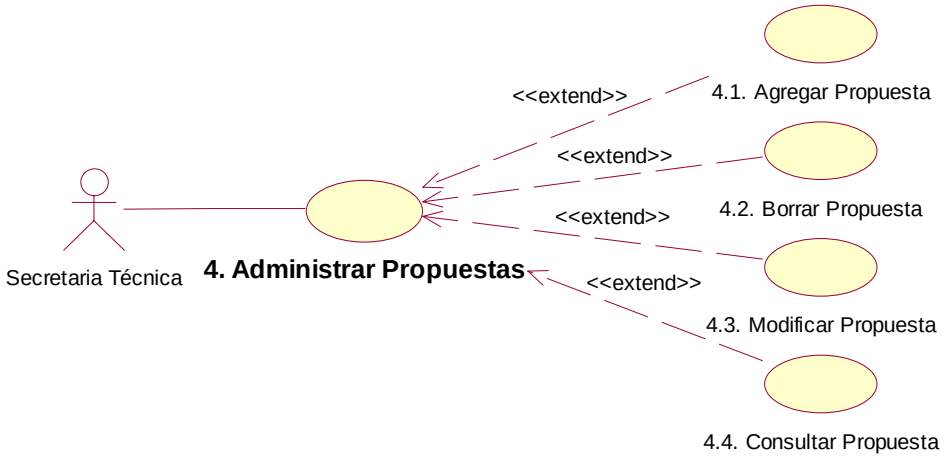
Definición	
Caso de uso:	0.2. Buscar Asignatura.
Tipo:	Inclusión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Buscar Asignatura**, permite a la *Secretaria Técnica* buscar asignaturas que existan en el Sistema.
Precondiciones	La *Secretaria Técnica* debe haber accedido al Sistema. La Asignatura que se busca debe existir en el Sistema.
Poscondiciones	La *Secretaria Técnica* habrá encontrado la Asignatura buscada.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* selecciona si va a buscar la Asignatura por nombre o por clave de la Asignatura.	3.	EL *Sistema* muestra las asignaturas existentes ordenadas por la elección de la *Secretaria Técnica*.	E01
3.	La *Secretaría Técnica* elige la Asignatura que busca.	4.	El *Sistema* recupera todos los datos de la Asignatura seleccionada**.**	E02

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se pueden ordenar los datos	EL *Sistema* indica que no es posible ordenar los datos en la forma que se seleccionó.
E02	No se pueden recuperar los datos de la Asignatura	El *Sistema* muestra un mensaje indicando que no se pueden recuperar los datos de la Asignatura.

4.1.1.1.6 – Caso de uso: 4. Administrar Propuestas.

El detalle del caso de uso **4. Administrar Propuestas** se muestra a continuación así como sus casos de uso de extensión e inclusión.

Diagrama del caso de uso


Caso de uso:	4. Administrar Propuestas.
Tipo:	Básico.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Administrar Propuestas**, describe cómo la *Secretaria Técnica* administra la información de las Propuestas.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema*.
Poscondiciones:	La *Secretaria Técnica* podrá elegir entre agregar, borrar, modificar o consultar la información sobre Propuestas.

Flujos

Flujo principal:

Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* elige la opción Administrar Propuestas del *Sistema.*	2.	El *Sistema* muestra las opciones de Agregar, Borrar, Modificar y Consultar Propuestas. Además de la opción de Salir.	E01
3.	La *Secretaría Técnica* selecciona una de las opciones mostradas por el *Sistema*.	4.	El *Sistema* procesa la opción elegida**.**	E02

Flujos alternativos:

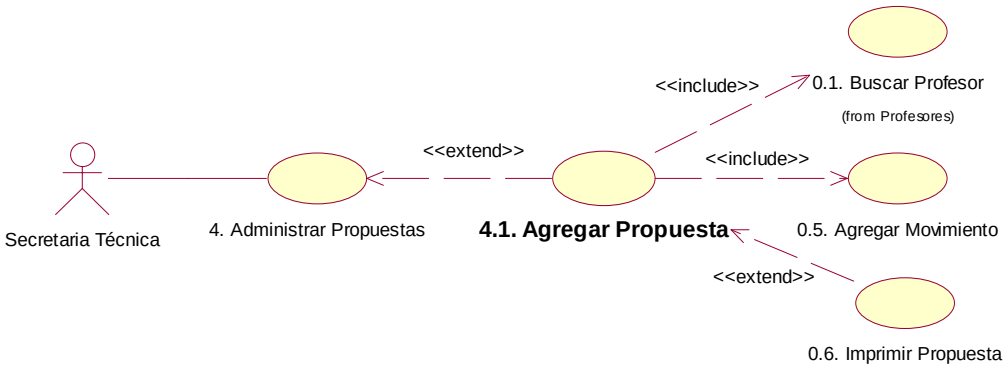
Actor		**Sistema**	
Paso	**Acciones**	**Paso**	**Acciones**
3A.1	La *Secretaria Técnica* elige la opción	3A.2	El *Sistema* procede de acuerdo al caso de uso

	Agregar.		**Agregar Propuesta**.
3B.1	La *Secretaria Técnica* elige la opción Borrar.	3B.2	El *Sistema* procede de acuerdo al caso de uso **Borrar Propuesta**.
3C.1	La *Secretaria Técnica* elige la opción Modificar.	3C.2	El *Sistema* procede de acuerdo al caso de uso **Modificar Propuesta**.
3D.1	La *Secretaria Técnica* elige la opción Consultar.	3D.2	El *Sistema* procede de acuerdo al caso de uso **Consultar Propuesta**.
3E.1	La *Secretaria Técnica* elige la opción Salir.	3E.2	El *Sistema* sale de la opción Administrar Propuestas.

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se pueden mostrar las opciones	EL *Sistema* indica que no es posible mostrar las opciones.

El caso de uso de extensión **4.1. Agregar Propuesta** se muestra a continuación:

Diagrama del caso de uso

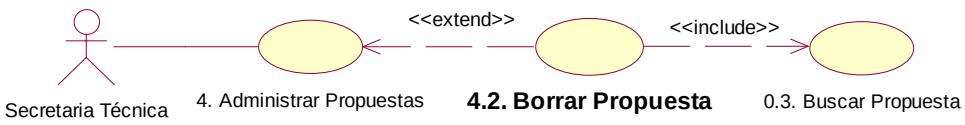


Definición	
Caso de uso:	4.1. Agregar Propuesta.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Agregar Propuesta**, describe como la *Secretaria Técnica* agrega propuestas al *Sistema*.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema* y elegido la opción de Agregar Propuesta.
Poscondiciones:	La *Secretaria Técnica* habrá agregado la información de una Propuesta al *Sistema*.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* elige Agregar Propuesta.	2.	El *Sistema* pide elegir al Profesor de la Propuesta.	
3.	La *Secretaría Técnica* elige al Profesor, realizando el caso de uso **Buscar Profesor**.	4.	El *Sistema* muestra la información del Profesor (nombre, RFC, domicilio, teléfono y número de trabajador)**.**	E01
5.	La *Secretaria Técnica* elige el periodo escolar de la Propuesta.	6.	El *Sistema* establece el periodo seleccionado por la *Secretaria Técnica***.**	
7.	La *Secretaria Técnica* agrega movimientos a la Propuesta, realizando el caso de uso **Agregar Movimiento**, cuantas veces sea necesario.	8.	El *Sistema* muestra los movimientos que se han agregado a la Propuesta**.**	E02
9.	La *Secretaria Técnica* elige cerrar la propuesta.	10.	El *Sistema* cierra la Propuesta.	
Flujos alternativos:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	
9A.1	La *Secretaria Técnica* elige imprimir la Propuesta.	9A.2	El *Sistema* procede como se indica en el caso de uso **Imprimir Propuesta**.	

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se pueden mostrar los datos del profesor	EL *Sistema* indica que no se pueden mostrar los datos del Profesor.
E02	No se pueden mostrar los movimientos	El *Sistema* muestra un mensaje indicando que no se pueden mostrar los movimientos.

El caso de uso de extensión **4.2. Borrar Propuesta** se muestra a continuación:

Diagrama del caso de uso


Definición	
Caso de uso:	4.2. Borrar Propuesta.
Tipo:	Extensión.

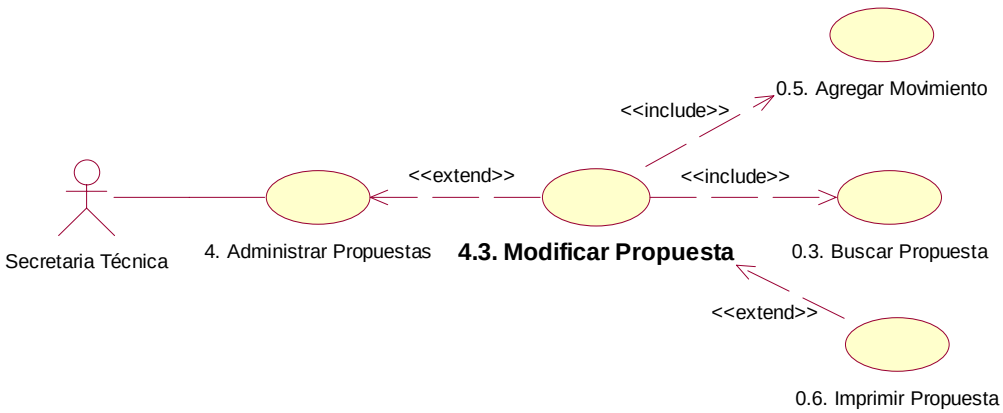
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Borrar Propuesta**, describe como la *Secretaria Técnica* borra propuestas al *Sistema*.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema* y elegido la opción de Borrar Propuesta.
Poscondiciones:	La *Secretaria Técnica* habrá borrado la información de una Propuesta del *Sistema*.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* elige Borrar Propuesta.	2.	El *Sistema* muestra las propuestas existentes y pide elegir una propuesta.	
3.	La *Secretaría Técnica* busca la propuesta realizando el caso de uso **Buscar Propuesta**, para elegir la propuesta que desea borrar.	4.	El *Sistema* borra la Propuesta elegida**.**	E01

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se pueden borrar la propuesta.	EL *Sistema* indica que no se puede borrar la propuesta elegida.

El caso de uso de extensión **4.3. Modificar Propuesta** se muestra a continuación:

Diagrama del caso de uso



Definición	
Caso de uso:	4.3. Modificar Propuesta.
Tipo:	Extensión.

Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Modificar Propuesta**, describe como la *Secretaria Técnica* modifica propuestas al *Sistema*.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema* y elegido la opción de **Modificar Propuesta**.
Poscondiciones:	La *Secretaria Técnica* habrá modificado la información de una Propuesta del *Sistema*.

Flujos

Flujo principal:

Actor		Sistema		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* elige Modificar Propuesta.	2.	El *Sistema* pide elegir la Propuesta que se va a modificar.	
3.	La *Secretaría Técnica* elige la Propuesta, realizando el caso de uso **Buscar Propuesta**.	4.	El *Sistema* muestra la información de la Propuesta**.**	E01
5.	La *Secretaria Técnica* modifica la información de la Propuesta (agregando o borrando movimientos).			
6.	La *Secretaria Técnica* elige cerrar la propuesta.	7.	El *Sistema* cierra la Propuesta.	

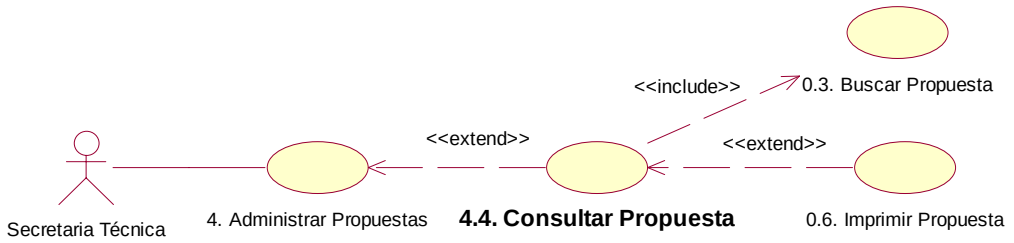
Flujos alternativos:

Actor		Sistema	
Paso	**Acciones**	**Paso**	**Acciones**
6A.1	La *Secretaria Técnica* elige imprimir la Propuesta.	9A.2	El *Sistema* procede como se indica en el caso de uso **Imprimir Propuesta**.

Excepciones

Identificador	Nombre	Respuesta del Sistema
E01	No se pueden mostrar los datos del profesor	EL *Sistema* indica que no se pueden mostrar los datos del Profesor.

El caso de uso de extensión **4.4. Consultar Propuesta** se muestra a continuación:

Diagrama del caso de uso


Definición	
Caso de uso:	4.4. Consultar Propuesta.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Consultar Propuesta**, describe como la *Secretaria Técnica* consulta una Propuesta del *Sistema*.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema* y elegido la opción de **Consultar Propuesta**.
Poscondiciones:	La *Secretaria Técnica* habrá consultado la información de una Propuesta al *Sistema*.

Flujos

Flujo principal:

Actor		Sistema		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* elige Consultar Propuesta.	2.	El *Sistema* pide elegir la Propuesta que se va a consultar.	
3.	La *Secretaría Técnica* elige la Propuesta, realizando el caso de uso **Buscar Propuesta**.	4.	El *Sistema* muestra la información de la Propuesta**.**	E01
5.	La *Secretaria Técnica* consulta la información de la Propuesta.			
6.	La *Secretaria Técnica* elige cerrar la propuesta.	7.	El *Sistema* cierra la Propuesta.	

Flujos alternativos:

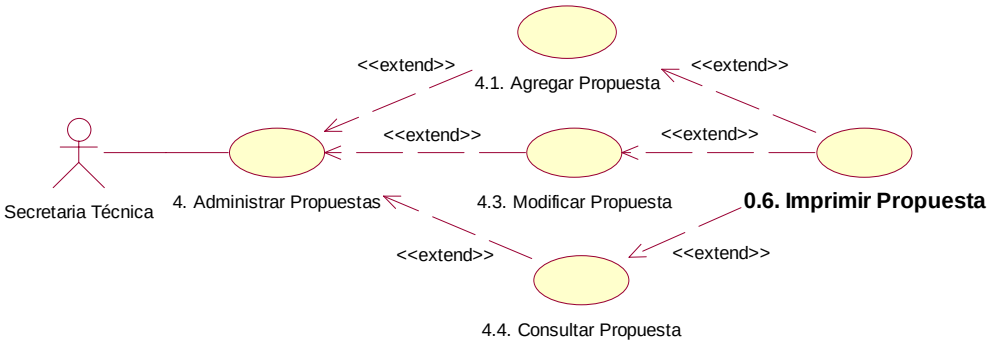
Actor		Sistema	
Paso	**Acciones**	**Paso**	**Acciones**
6A.1	La *Secretaria Técnica* elige imprimir la Propuesta.	9A.2	El *Sistema* procede como se indica en el caso de uso **Imprimir Propuesta**.

Excepciones

Identificador	Nombre	Respuesta del Sistema
E01	No se pueden mostrar	EL *Sistema* indica que no se pueden mostrar los datos del Profesor.

	los datos del profesor	

El caso de uso de extensión **0.6. Imprimir Propuesta** se muestra a continuación:

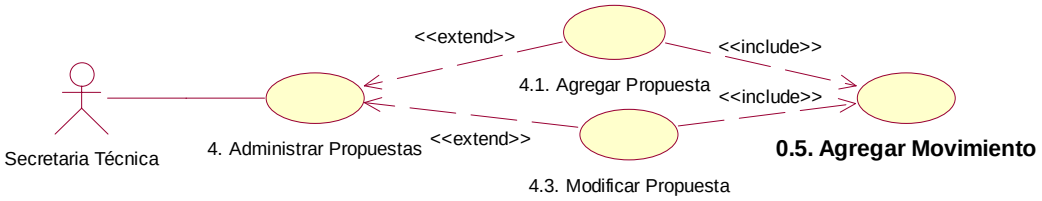
Diagrama del caso de uso


Definición	
Caso de uso:	0.6. Imprimir Propuesta.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Imprimir Propuesta**, describe como la *Secretaria Técnica* imprime una propuesta.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema* y elegido la opción de Imprimir Propuesta.
Poscondiciones:	La *Secretaria Técnica* habrá impreso una Propuesta.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* elige Imprimir Propuesta.	2.	El *Sistema* muestra la Propuesta en vista preliminar.	E01
3.	La *Secretaría Técnica* elige confirma la impresión de la Propuesta.	4.	El *Sistema* imprime la propuesta.	E02

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se puede mostrar la vista preliminar de la Propuesta.	EL *Sistema* indica que no se puede mostrar la vista preliminar de la Propuesta.
E02	No se puede imprimir la Propuesta	El *Sistema* muestra un mensaje indicando que no se pueden imprimir la Propuesta.

El caso de uso de inclusión **0.5. Agregar Movimiento** se muestra a continuación:

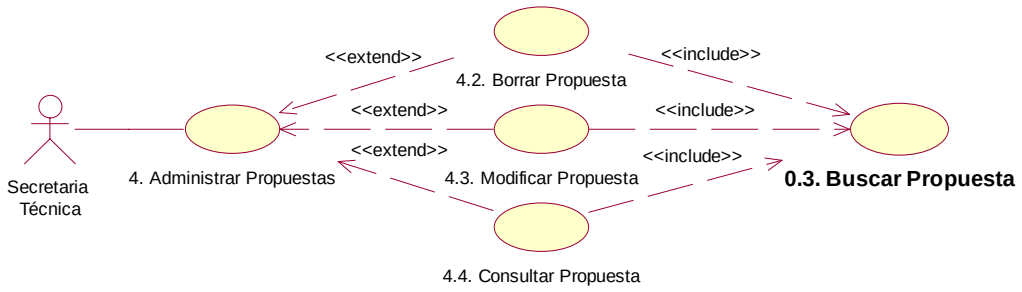
Diagrama del caso de uso


Definición	
Caso de uso:	0.5. Agregar Movimiento.
Tipo:	Inclusión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Agregar Movimiento**, describe como la *Secretaria Técnica* agrega un movimiento a una Propuesta.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema* y elegido la opción de **Agregar Movimiento**.
Poscondiciones:	La *Secretaria Técnica* habrá agregado la información de un Movimiento a una Propuesta.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* elige Agregar Movimiento al *Sistema.*	2.	El *Sistema* pide la información del movimiento (tipo, causa, clave de la asignatura, nombre de la asignatura, horario, días que se imparte la asignatura, horas de teoría, horas de práctica, fecha de inicio, fecha de termino.	
3.	La *Secretaría Técnica* introduce la información del movimiento.	4.	El *Sistema* agrega la información del movimiento a la Propuesta**.**	E01

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se puede agregar el movimiento a la Propuesta	EL *Sistema* indica que no se puede agregar la información del movimiento a la Propuesta.

El caso de uso de inclusión **0.3. Buscar Propuesta** se muestra a continuación:

Diagrama del caso de uso


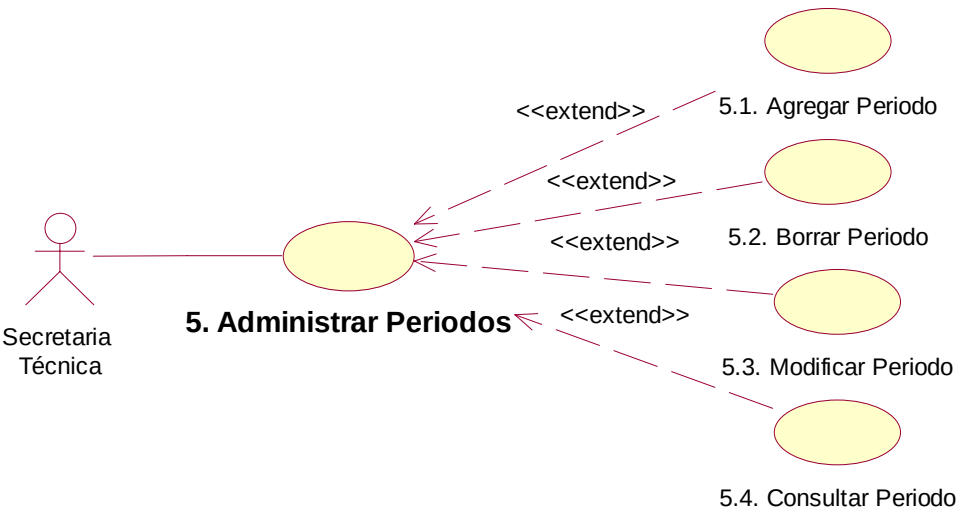
Definición	
Caso de uso:	0.3. Buscar Propuesta.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Buscar Propuesta**, describe como la *Secretaria Técnica* busca una propuesta en el *Sistema*.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema* La Propuesta que se busca debe existir en el Sistema.
Poscondiciones:	La *Secretaria Técnica* habrá encontrado la Propuesta buscada.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* selecciona si va a buscar la propuesta por nombre del profesor, RFC, por número de trabajador o por periodo escolar.	3.	EL *Sistema* muestra las propuestas existentes ordenadas por la elección de la *Secretaria Técnica*.	E01
3.	La *Secretaría Técnica* elige la propuesta que busca.	4.	El *Sistema* recupera todos los datos de la Propuesta seleccionada**.**	E02

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se pueden ordenar los datos	EL *Sistema* indica que no es posible ordenar los datos en la forma que se seleccionó.
E02	No se pueden recuperar los datos de la Propuesta	El *Sistema* muestra un mensaje indicando que no se pueden recuperar los datos de la Propuesta.

4.1.1.1.7 – Caso de uso: 5. Administrar Periodos.

El detalle del caso de uso **5. Administrar Periodos** se muestra a continuación así como sus casos de uso de extensión e inclusión.

Diagrama del caso de uso


Caso de uso:	5. Administrar Periodos.
Tipo:	Básico.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Administrar Periodos**, describe cómo la *Secretaria Técnica* administra la información de los Periodos.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema*.
Poscondiciones:	La *Secretaria Técnica* podrá elegir entre agregar, borrar, modificar o consultar la información sobre Periodos.

Flujos

Flujo principal:

Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* elige la opción Administrar Periodos del *Sistema.*	2.	El *Sistema* muestra las opciones de Agregar, Borrar, Modificar y Consultar Periodos. Además de la opción de Salir.	E01
3.	La *Secretaría Técnica* selecciona una de las opciones mostradas por el *Sistema*.	4.	El *Sistema* procesa la opción elegida**.**	E02

Flujos alternativos:

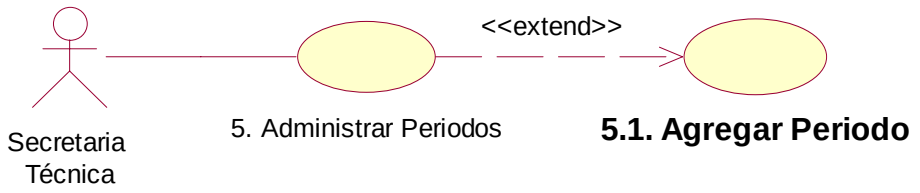
Actor	**Sistema**

Paso	Acciones	Paso	Acciones
3A.1	La *Secretaria Técnica* elige la opción Agregar.	3A.2	El *Sistema* procede de acuerdo al caso de uso **Agregar Periodo**.
3B.1	La *Secretaria Técnica* elige la opción Borrar.	3B.2	El *Sistema* procede de acuerdo al caso de uso **Borrar Periodo**.
3C.1	La *Secretaria Técnica* elige la opción Modificar.	3C.2	El *Sistema* procede de acuerdo al caso de uso **Modificar Periodo**.
3D.1	La *Secretaria Técnica* elige la opción Consultar.	3D.2	El *Sistema* procede de acuerdo al caso de uso **Consultar Periodo**.
3E.1	La *Secretaria Técnica* elige la opción Salir.	3E.2	El *Sistema* sale de la opción Administrar Periodos.

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se pueden mostrar las opciones	EL *Sistema* indica que no es posible mostrar las opciones.

El caso de uso de extensión **5.1. Agregar Periodo** se muestra a continuación:

Diagrama del caso de uso



Definición	
Caso de uso:	5.1. Agregar Periodo.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Agregar Periodo**, describe como la *Secretaria Técnica* agrega Periodos al *Sistema*.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema* y elegido la opción de Agregar Periodo.
Poscondiciones:	La *Secretaria Técnica* habrá agregado la información de un Periodo al *Sistema*.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* elige Agregar Periodo al *Sistema.*	2.	El *Sistema* pide los datos del Periodo (fecha de inicio, fecha de término, y	

			nombre del Periodo).	
3.	La *Secretaría Técnica* introduce los datos del Periodo.	4.	El *Sistema* verifica los datos ingresados del Periodo**.**	E01, E02
		5.	El *Sistema* almacena los datos del Periodo**.**	E03
		6.	El *Sistema* indica que los datos se han almacenado**.**	

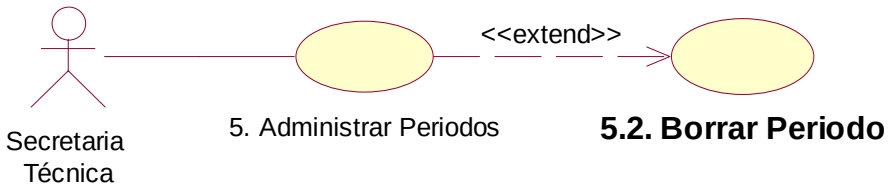
Flujos alternativos:

Actor		Sistema	
Paso	**Acciones**	**Paso**	**Acciones**
5A.1	La Secretaria Técnica cancela la operación.	5A.2	El Sistema cancela la operación de agregar.

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	Datos inválidos	EL *Sistema* indica que los datos que proporcionó el usuario son inválidos y que introduzca datos válidos.
E02	El Periodo ya está registrado	El *Sistema* muestra un mensaje indicando que el Periodo ya esta registrado en el *Sistema*.
E03	No se pueden almacenar los datos	El *Sistema* muestra un mensaje indicando que no es posible almacenar los datos.

El caso de uso de extensión **5.2. Borrar Periodo** se muestra a continuación:

Diagrama del caso de uso

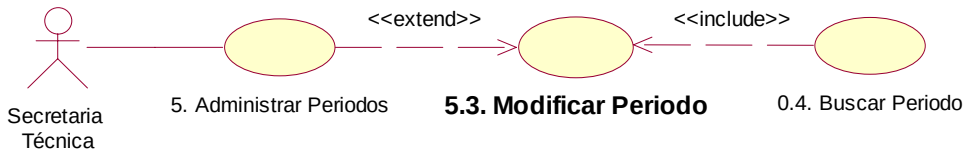


Definición	
Caso de uso:	5.2. Borrar Periodo.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Borrar Periodo**, permite a la *Secretaria Técnica* borrar Periodos del *Sistema*.
Precondiciones	La *Secretaria Técnica* debe haber accedido al *Sistema*.
Poscondiciones	La *Secretaria Técnica* habrá eliminado el Periodo elegido.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* busca el Periodo que va a borrar, para buscar el Periodo realiza el caso de uso **Buscar Periodo.**	2.	El *Sistema* muestra la información del Periodo que se seleccionó.	E01
3.	La *Secretaría Técnica* elige borrar el periodo.	4.	El *Sistema* borra la información del Periodo.	E02
	.	5.	El *Sistema* muestra un mensaje indicando que la información del Periodo se ha borrado.	

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se puede mostrar la información del Periodo	EL *Sistema* indica que no se puede mostrar la información del Periodo.
E02	No se puede borrar la información del Periodo	El *Sistema* muestra un mensaje indicando que no se puede borrar la información del Periodo.

El caso de uso de extensión **5.3. Modificar Periodo** se muestra a continuación:

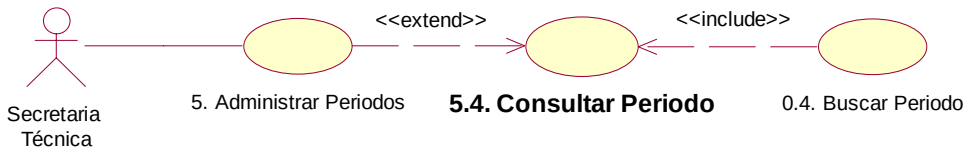
Diagrama del caso de uso


Definición	
Caso de uso:	5.3. Modificar Periodo.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Modificar Periodo**, permite a la *Secretaria Técnica* modificar la información de los Periodos del *Sistema*.
Precondiciones:	La *Secretaria Técnica* debe haber accedido al *Sistema*. El Periodo al que se hará una modificación en su información debe de existir.
Poscondiciones:	La *Secretaria Técnica* habrá modificado la información del Periodo elegido.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* busca el Periodo que va a modificar, para buscar al periodo realiza el caso de uso **Buscar Periodo**	2.	El *Sistema* muestra la información del Periodo que se seleccionó.	E01
3.	La *Secretaría Técnica* elige modificar la información del Periodo.	4.	El *Sistema* permite la modificación de la información del Periodo.	
5.	La Secretaría Técnica modifica la información.			
6.	La Secretaría Técnica elige guardar los cambios.	7.	El *Sistema* verifica la información del Periodo.	E02
		8.	El *Sistema* guarda la información del Periodo.	E03
		9.	El *Sistema* muestra un mensaje indicando que la información del Periodo se ha guardado.	
Flujos alternativos:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	
6A.1	La *Secretaria Técnica* cancela los cambios.	6A.2	El *Sistema* cancela los cambios y recupera la información del Periodo almacenada antes de los cambios.	

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se puede mostrar la información del Periodo	EL *Sistema* indica que no se puede mostrar la información del Periodo.
E02	Datos inválidos	EL *Sistema* indica que los datos que proporcionó el usuario son inválidos y pide que introduzca datos válidos.
E03	No se puede guardar la información del Periodo	El *Sistema* muestra un mensaje indicando que no se puede guardar la información del Periodo.

El caso de uso de extensión **5.4. Consultar Periodo** se muestra a continuación:

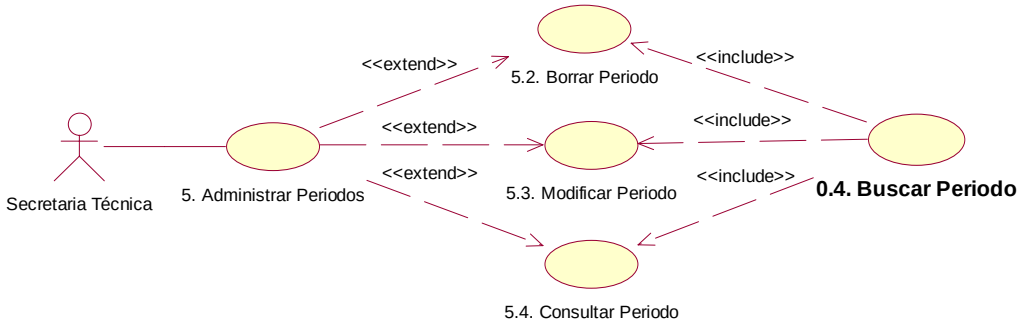
Diagrama del caso de uso


Definición	
Caso de uso:	5.4. Consultar Periodo.
Tipo:	Extensión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Consultar Periodo**, permite a la *Secretaria Técnica* consultar información de los Periodos registrados en el *Sistema*.
Precondiciones	La *Secretaria Técnica* debe haber accedido al *Sistema*.
Poscondiciones	La *Secretaria Técnica* habrá consultado la información del Periodo elegido.

Flujos				
Flujo principal:				
Actor		**Sistema**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* busca el Periodo que va a consultar, para buscar al Periodo realiza el caso de uso **Buscar Periodo.**	2.	El *Sistema* muestra la información del Periodo que se seleccionó.	E01

Excepciones		
Identificador	**Nombre**	**Respuesta del Sistema**
E01	No se puede mostrar la información del Periodo	EL *Sistema* indica que no se puede mostrar la información del Periodo.

El caso de uso de inclusión **0.4. Buscar Periodo** se muestra a continuación:

Diagrama del caso de uso


Definición	
Caso de uso:	0.4. Buscar Periodo.
Tipo:	Inclusión.
Actor principal:	*Secretaria Técnica*.
Descripción:	El caso de uso, **Buscar Periodo**, permite a la *Secretaria Técnica* buscar periodos que existan en el Sistema.
Precondiciones	La *Secretaria Técnica* debe haber accedido al Sistema. El Periodo que se busca debe existir en el Sistema.
Poscondiciones	La *Secretaria Técnica* habrá encontrado el Periodo buscado.

Flujos

Flujo principal:

Actor		Sistema		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1.	La *Secretaria Técnica* selecciona si va a buscar el periodo por nombre, fecha de inicio o fecha de término.	3.	EL *Sistema* muestra los periodos existentes ordenados por la elección de la *Secretaria Técnica*.	E01
3.	La *Secretaría Técnica* elige el Periodo que busca.	4.	El *Sistema* recupera todos los datos del Periodo seleccionado**.**	E02

Excepciones

Identificador	Nombre	Respuesta del Sistema
E01	No se pueden ordenar los datos	EL *Sistema* indica que no es posible ordenar los datos en la forma que se seleccionó.
E02	No se pueden recuperar los datos del Periodo	El *Sistema* muestra un mensaje indicando que no se puede recuperar los datos del Periodo.

4.1.1.2 – Requerimientos no funcionales

Un requerimiento no funcional es una restricción de los servicios o funciones del sistema [35].

Los requerimientos no funcionales del sistema son los siguientes:

- El sistema debe ejecutarse en los sistemas operativos Windows 2000 y Windows XP.
- El sistema tiene que ser desarrollado en la plataforma .NET (Visual Basic .NET).
- El sistema debe desarrollarse en un periodo de 3 meses.
- El sistema debe de ser fácil de usar.
- Sólo la persona autorizada debe de acceder al sistema.

4.2 – PLANEACIÓN

La planeación para el desarrollo de la aplicación se realizó con el cliente para poder realizar las siguientes actividades:

- Estimación de tiempo de desarrollo de los casos de uso.
- Prioridad de cada uno de los casos de uso.
- Establecer el plan de liberación.

Estas actividades, realizadas para el desarrollo de la aplicación, se describen en las siguientes secciones.

4.2.1 – Estimación de tiempo de desarrollo de los casos de uso

La estimación de tiempo de desarrollo para cada uno de los casos de uso se hizo basándose en la complejidad de cada caso de uso.

La estimación de tiempo la realizó el desarrollador dando como resultado la Tabla 4.1.

Tabla 4.1 – Estimación de los casos de uso

Caso de uso	Estimación (días)
1. Iniciar sesión	7
2. Administrar Profesores	10
3. Administra asignaturas	10
4. Administrar Periodos	10
5. Administrar Propuestas	18

La unidad de estimación de cada caso de uso está en días en un acuerdo entre el cliente y el desarrollador

4.2.2 – Prioridad de casos de uso

La prioridad de cada uno de los casos de uso la realizó el cliente basándose en la importancia que tiene cada caso de uso para realizar su trabajo.

Tabla 4.2 – Prioridad de los casos de uso

Caso de uso	Prioridad
1. Iniciar sesión	5
2. Administrar Profesores	1
3. Administra asignaturas	2
4. Administrar Periodos	3
5. Administrar Propuestas	4

En la Tabla 4.2 muestra la prioridad establecida para cada caso de uso.

4.2.3 – Plan de liberación

Tomando en cuenta la información obtenida al hacer una estimación de tiempo de desarrollo realizada por el desarrollador y la prioridad asignada por el cliente para cada uno de los casos de uso, se realizó un Plan de liberación para el desarrollo de la aplicación.

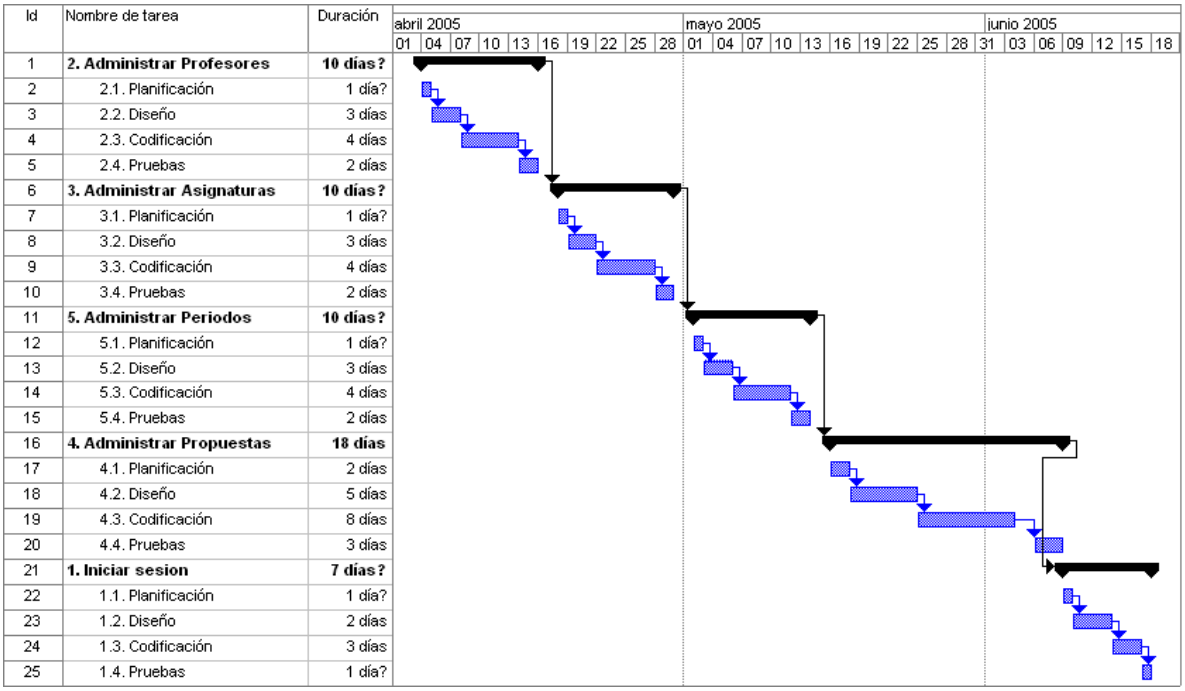


Figura 4.4 – Diagrama de Gantt del Plan de liberación

El Plan de liberación cuenta con cinco iteraciones que corresponden a cada uno de los casos de uso principales. Donde cada iteración pasa por las fases de planificación, diseño, codificación y pruebas.

4.3 – DISEÑO

El diseño de la aplicación abarca desde la definición de la arquitectura de la aplicación hasta el diseño de la interfaz de usuario, de la base de datos y las clases que realicen la lógica de la aplicación.

4.3.1 – Arquitectura de la aplicación

La arquitectura de la aplicación se muestra en la Figura 4.5.

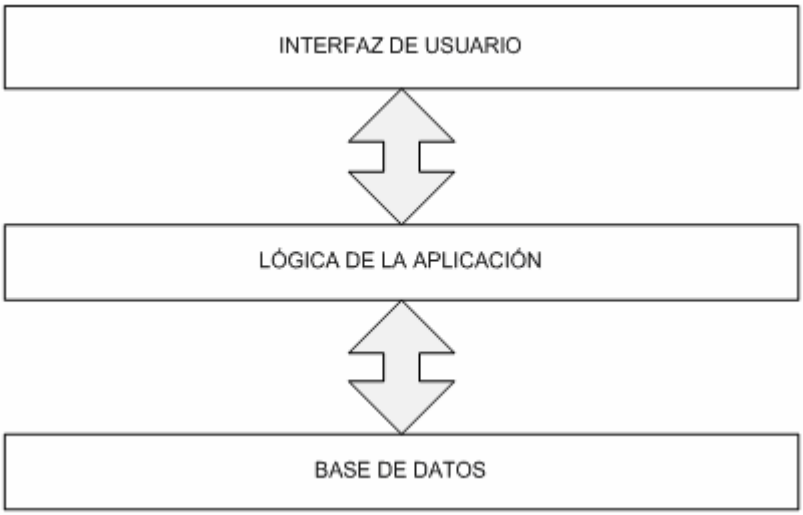


Figura 4.5 – Arquitectura de la aplicación

La arquitectura de la aplicación esta dividida en capas donde cada capa sólo se puede comunicar con la capa de abajo o de arriba, es decir, que la capa de interfaz no se puede comunicar directamente con la capa de base de datos, sino que tiene que comunicarse con la capa de la lógica de la aplicación para acceder a la base de datos.

Las capas de la arquitectura de la aplicación son las siguientes:

- **Capa de interfaz de usuario**. Esta capa contiene todas las clases que representan la interfaz de usuario con la que interactúa el usuario.
- **Capa de lógica de aplicación**. En esta capa están las clases que realizan toda la lógica de la aplicación.
- **Capa de base de datos**. Esta capa contiene los procedimientos almacenados que acceden a los datos de la base de datos.

4.3.2 – Diseño de la interfaz de usuario

En el diseño de la interfaz se tiene que desarrollar pantallas que permitan al usuario interactuar con el sistema para realizar las tareas que se especifican en los casos de uso.

En el caso del sistema, se desarrolló una pantalla inicial pidiendo al usuario autenticarse. La interfaz se muestra en la Figura 4.5.

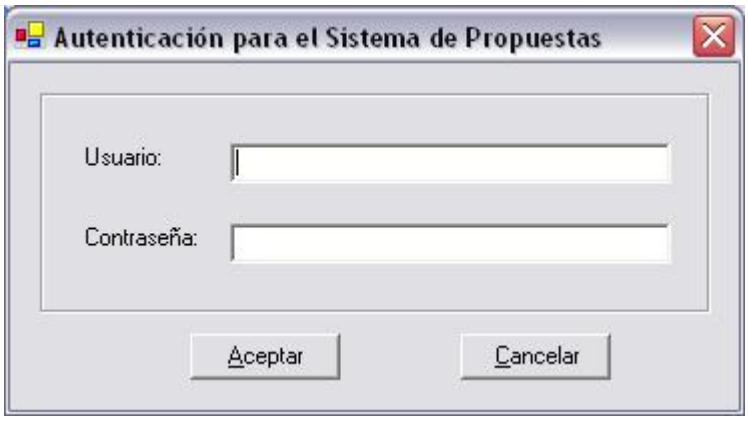


Figura 4.5 – Formulario para ingresar al sistema

La interfaz principal del sistema se muestra en la Figura 4.6.

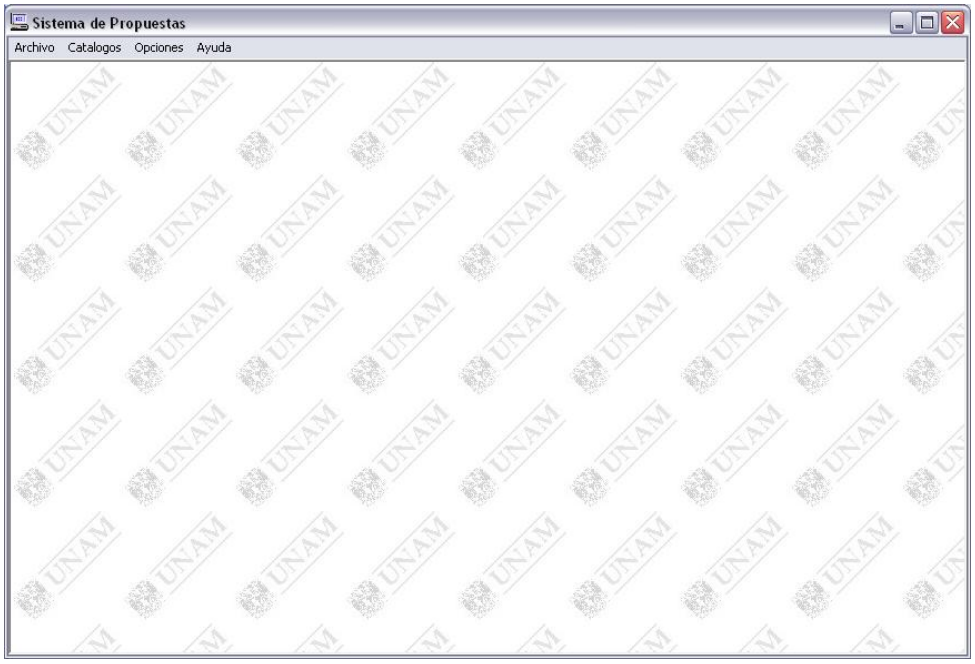


Figura 4.6 – Interfaz principal del sistema

Para el caso de uso **2. Administrar Profesores** se diseñó el formulario que se muestra en la Figura 4.7.

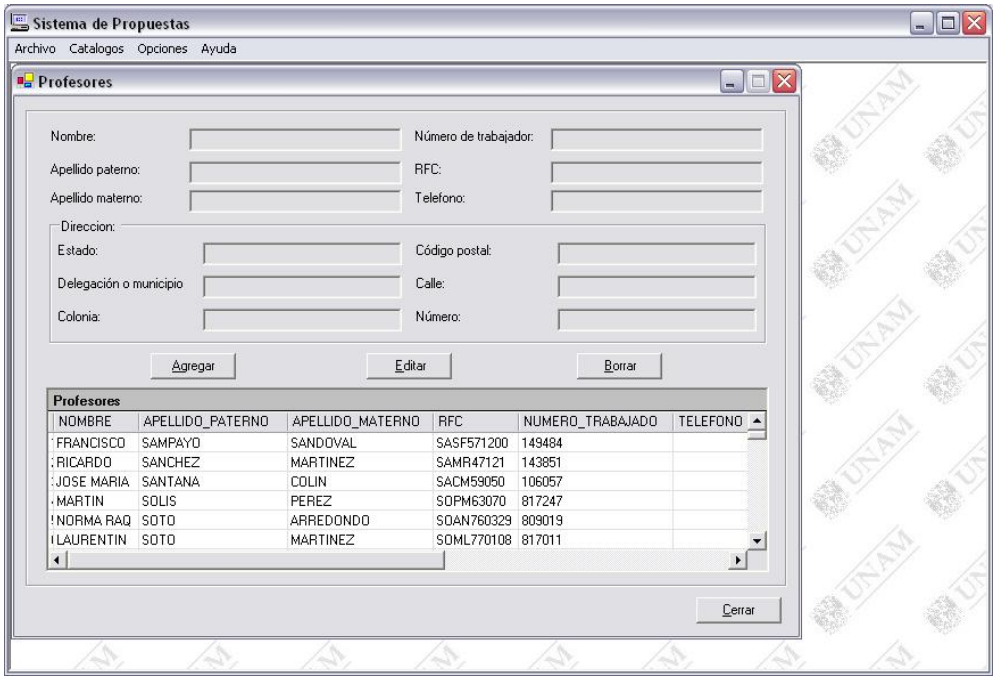


Figura 4.7 – Formulario para administrar la información de profesores

El diseño del formulario para el caso de uso **3. Administrar Asignaturas** se muestra en la Figura 4.8.

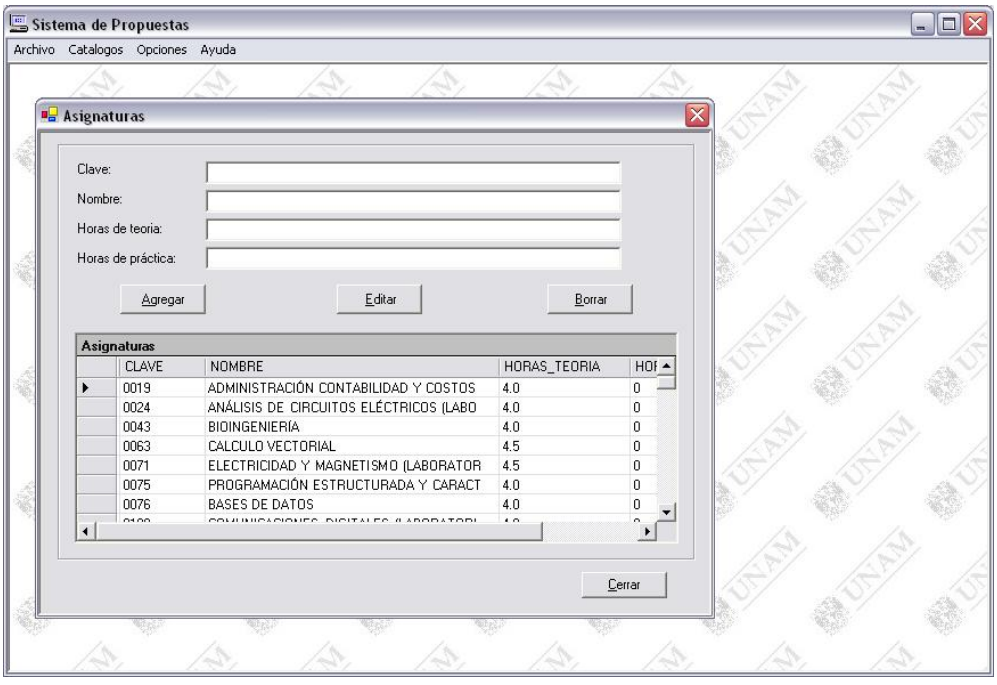


Figura 4.8 – Formulario para administrar la información de asignaturas

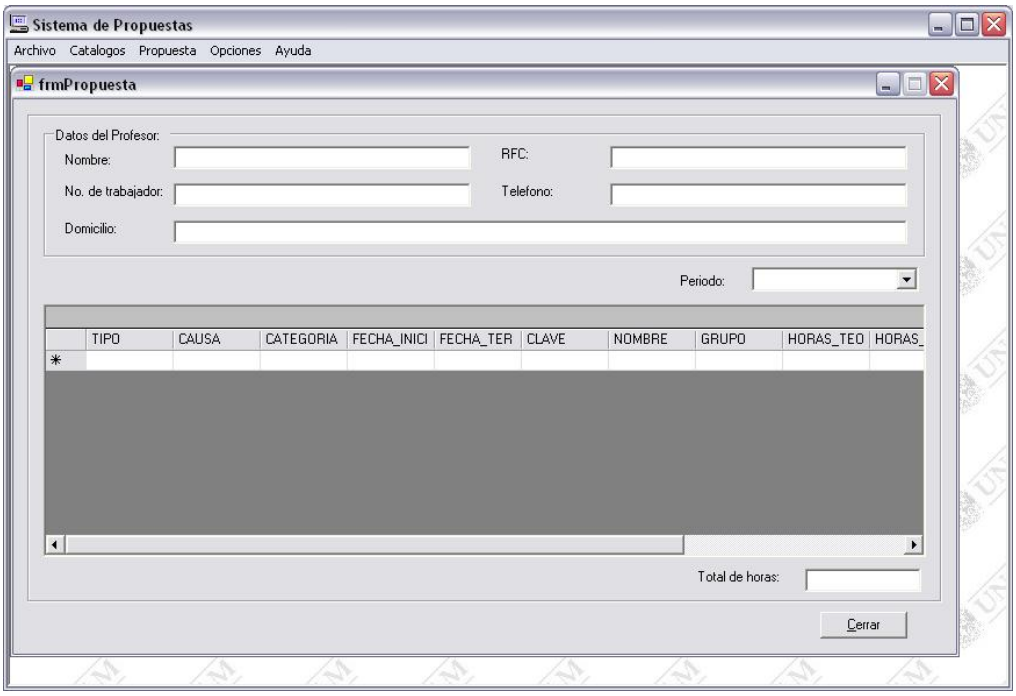


Figura 4.9 – Formulario para administrar propuestas

El formulario diseñado para el caso de uso **4. Administrar Propuestas** es el muestra la Figura 4.9.

Para el caso de uso **2. Administrar Periodos** se diseñó el formulario que se muestra en la Figura 4.10.

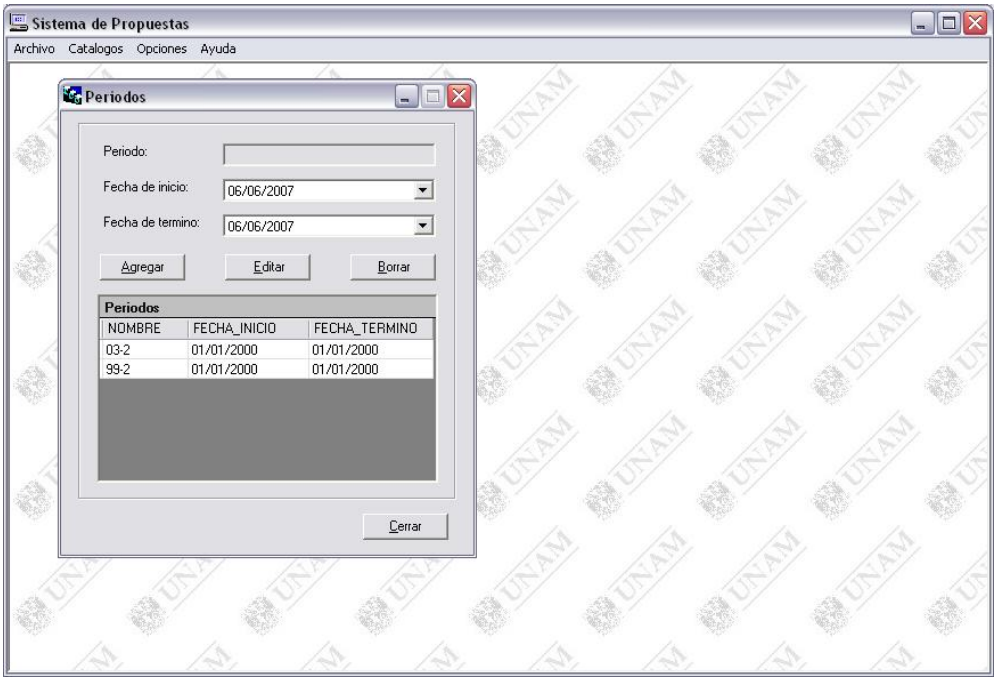


Figura 4.10 – Formulario para administrar la información de periodos

4.3.3 – Diseño de la base de datos

Después del diseño de la interfaz de usuario se realizó el diseño de la base de datos, la cual se diseño de acuerdo a las entidades identificadas en los requisitos (casos de uso).

El diagrama entidad-relación de la base de datos se muestra en la Figura 4.11.

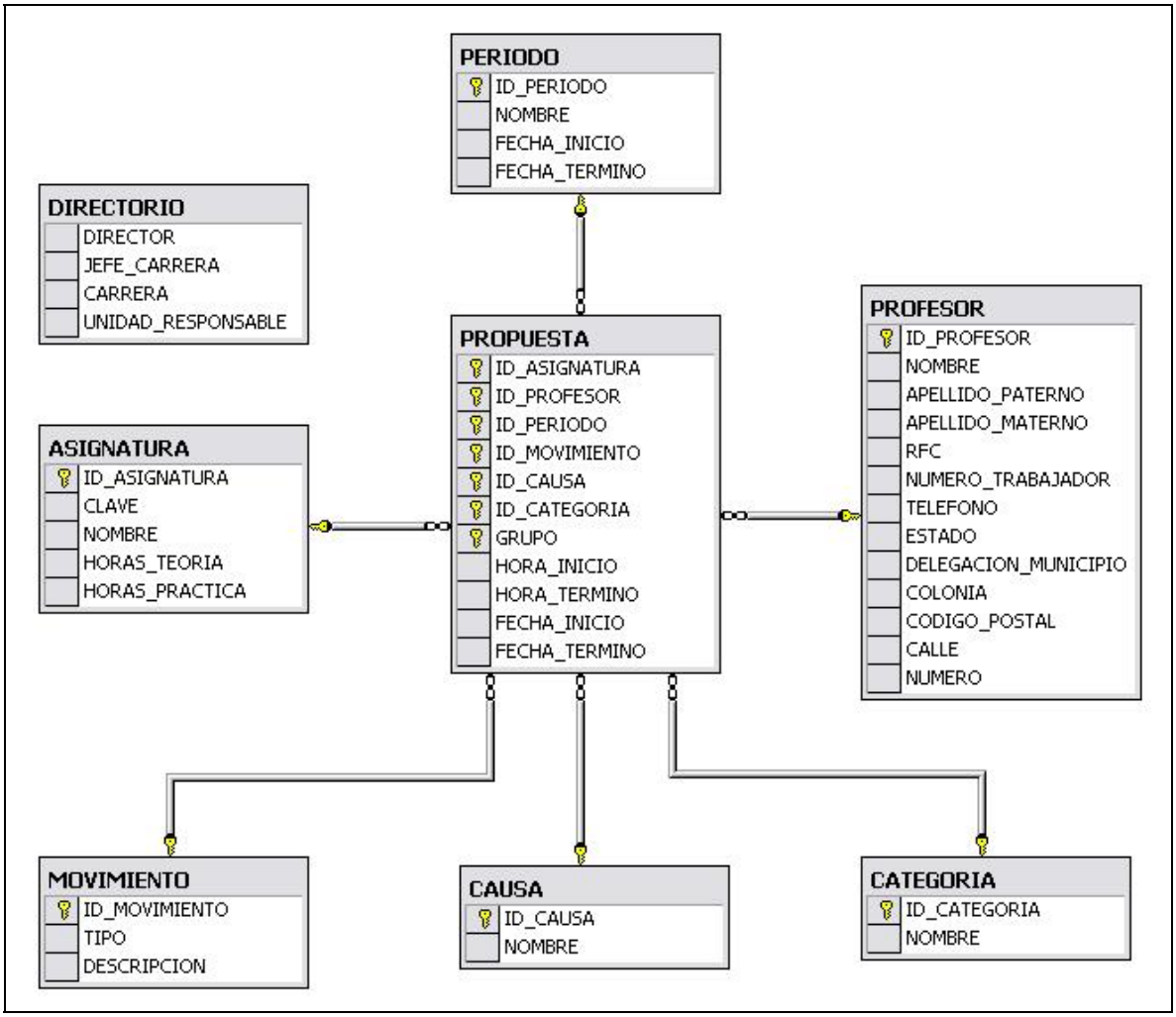


Figura 4.11 – Diagrama entidad-relación de la base de datos

Las entidades identificadas se encuentran en la Tabla 4.3.

Tabla 4.3 – Tablas de la base de datos

Tabla	Descripción
Periodo	En esta tabla se almacenan la información de los periodos.
Profesor	La información de los profesores se almacena en esta tabla.
Asignatura	Esta tabla contiene la información de las asignaturas.
Propuesta	En esta tabla se almacenan las propuestas.
Movimiento	Tabla donde se almacena los distintos tipos de movimientos que se pueden realizar.
Causa	Esta tabla contiene el catalogo de causas que origina un movimiento.
Categoría	Tabla donde se almacenan las categorías que puede tener un profesor.
Directorio	En esta tabla se almacena la información del directorio.

4.3.4 – Diagrama de clases

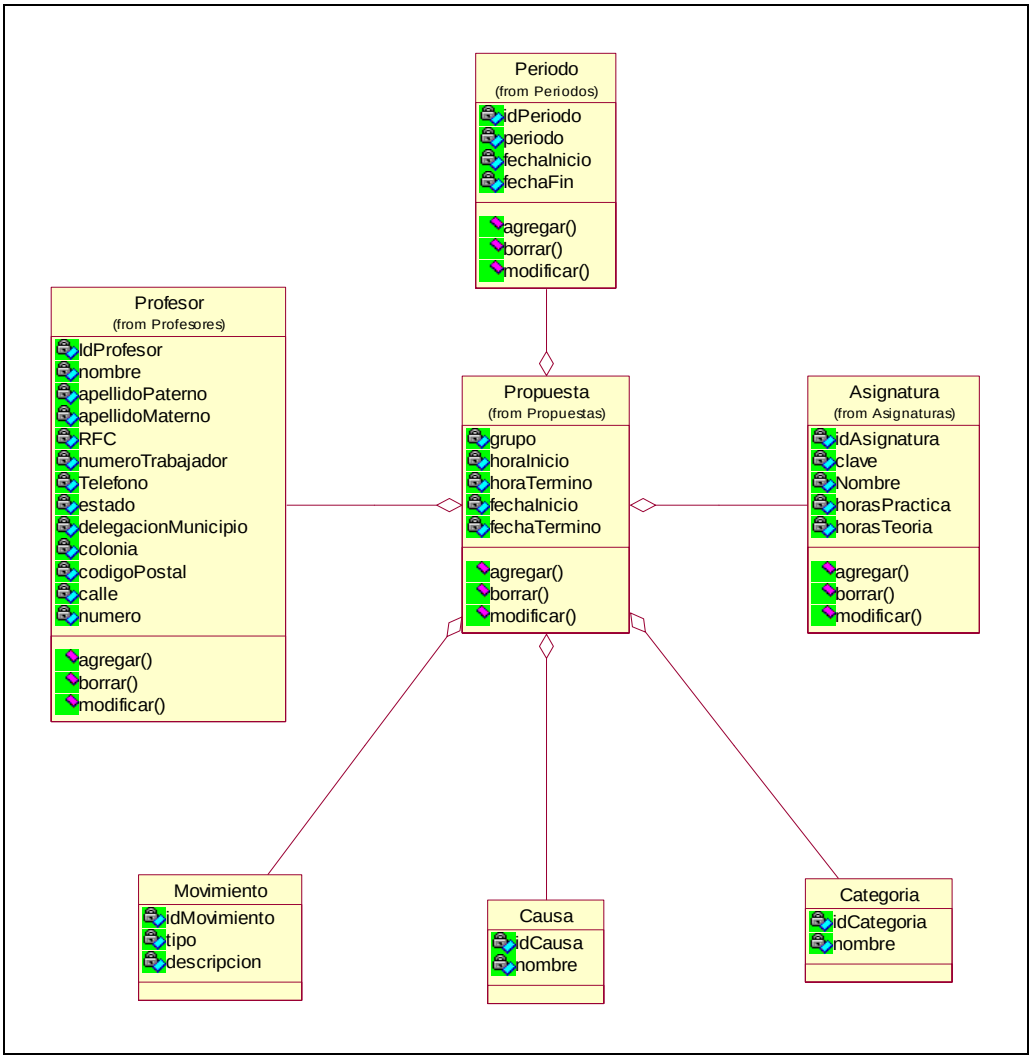


Figura 4.12 – Diagrama de clases de la aplicación

El diagrama de clases muestra las clases principales que representan toda la lógica de la aplicación.

La Figura 4.12 se muestra el diagrama de clases de la aplicación.

4.4 – PRUEBAS

Las pruebas son el proceso de encontrar diferencias entre el comportamiento esperado, especificado por los modelos del sistema, y el comportamiento observado del sistema [06].

Los tipos de pruebas realizadas en el desarrollo del sistema fueron:

- Pruebas de unidad
- Pruebas de integración
- Pruebas de sistema

Estas pruebas son explicadas en las secciones siguientes.

4.4.1 – Pruebas de unidad

Las pruebas de unidad o unitarias centran el proceso de verificación en la menor unidad del diseño del software: el componente software o módulo [31].

Existen tres motivaciones al centrarse en los componentes [06]:

1. La prueba unitaria reduce la complejidad de las actividades de prueba generales, permitiendo el enfoque en unidades pequeñas.
2. La prueba unitaria facilita encontrar y corregir defectos, ya que están involucrados pocos componentes.
3. La prueba unitaria permite el paralelismo en las actividades de prueba; esto es, cada componente puede probarse en forma independiente de los demás.

Para el desarrollo del sistema, la unidad mínima que se utilizó para las pruebas unitarias fue la clase. Cada clase dentro del sistema fue probada por medio de NUnit[1].

4.4.2 – Pruebas de integración

Las pruebas de integración detectan defectos que no se descubren durante las pruebas de unidad enfocándose en pequeños grupos de componentes. Dos o más componentes se integran y prueban, y después de que las pruebas ya no detectan ningún defecto se añaden componentes adicionales al grupo. Este procedimiento permite la prueba de partes cada vez más complejas del sistema, manteniendo relativamente pequeña la ubicación de los

[1] NUnit es una marco de trabajo de pruebas unitarias para todos los lenguajes .NET (http://nunit.com)

defectos potenciales (es decir, el componente añadido más recientemente es, por lo general, el que activa los defectos descubiertos más recientemente [31].

Existen distintas estrategias para las pruebas de integración entre las más conocidas son las pruebas de abajo hacia arriba y las de arriba hacia abajo.

- **Pruebas de integración de abajo hacia arriba**. Esta estrategia prueba primero de manera individual a cada componente de la capa inferior y luego los integra con componentes de la siguiente capa superior.
- **Pruebas de integración de arriba hacia abajo**. En esta estrategia se prueba primero en forma unitaria los componentes de la capa superior y luego integra los componentes de la siguiente capa hacia abajo.

La estrategia utilizada para el desarrollo de la aplicación fue la de abajo hacia arriba.

4.4.3 – Pruebas de sistema

Las pruebas unitarias y de integración se enfocan en encontrar defectos en componentes individuales y en las interfaces entre los componentes. Una vez que se han integrado los componentes, las pruebas de sistema aseguran que el sistema completo cumpla con los requerimientos especificados

Las pruebas de aceptación son un tipo de pruebas de sistema que realiza el cliente por medio de casos de prueba que representan condiciones típicas bajo las cuales debe operar el sistema. Las pruebas de aceptación se hacen de acuerdo a los requerimientos que se hayan especificado y acordado con el cliente [06].

Para el desarrollo del sistema se realizaron pruebas de aceptación para cada uno de los casos de uso.

CONCLUSIONES

CONCLUSIONES

La elaboración de propuestas para profesores de la carrera de Ingeniería en Computación era una labor tediosa y repetitiva que ocupaba días de trabajo. Con el desarrollo del sistema se logró disminuir el tiempo que se ocupaba para realizar ese trabajo, con lo cual las personas encargadas de las tareas administrativas de la jefatura de carrera de Ingeniería en Computación disponen ahora de mayor tiempo para realizar otras actividades. Con lo anterior se logró cumplir con uno de los objetivos de este trabajo de tesis.

Las metodologías ágiles han tenido mucha difusión y su uso ha empezado a crecer. La programación extrema tiene muchas ventajas en su utilización, entre ellas se puede mencionar su énfasis en mostrar software funcionando como un indicador del avance del proyecto, permitiendo al cliente observar parte del producto final en vez de sólo mostrarle documentación del avance del proyecto. Además, el énfasis que se le da a las pruebas también tiene sus beneficios, ya que al realizar las pruebas antes que la escritura de código permite una mayor especificación de lo que se tiene que desarrollar.

Otra cosa que se puede observar en la programación extrema, y que trae beneficios, es la programación en parejas, ya que permite tener a una persona codificando mientras que otra va revisando el código para observar posibles errores que el programador puede introducir al codificar, con lo cual se reducen los posibles errores en la aplicación.

Aunque la programación extrema da la impresión de que tienen un enfoque similar al modelo de codificar y corregir, esto es sólo cuando se revisa o comprende de manera superficial, porque el conjunto de prácticas, valores y principios llevan un control sobre lo que se está desarrollando, además de que tiene fases bien definidas sobre lo que se tiene que hacer en cada una de ellas, así como roles que tienen que asumir cada uno de los integrantes de un equipo de desarrollo.

Por otro lado, al desarrollar la aplicación con la plataforma .NET se pueden observar los beneficios que aporta para los desarrolladores. La plataforma. NET tiene características y componentes que facilitan el trabajo para crear aplicaciones. Una de las características más importante de la plataforma .NET es la posibilidad de utilizar una gran variedad de lenguajes para codificar aplicaciones y la interoperabilidad que existe entre eso lenguajes, ya que con eso se gana en reutilización de componentes. Además de la integración que tiene la plataforma .NET con SQL Server 2000, que es un muy buen manejador de bases de datos.

BIBLIOGRAFÍA

BIBLIOGRAFÍA

TEXTOS

[01] ABRAHAMSSON, P., Salo, O., Ronkainen, J., Warsta, J.; *Agile software development methods Review and analysis;* 2002, VTT Publications

[02] BECK, Kent; *Extreme Programming Explained. Embrace Change*; United States of America 1999, Pearson Education

[03] BERES, Jason; *Sams Teach Yourself Visual Studio .NET 2003 in 21 Days*; United States of America 2003, Sams Publishing

[04] BLANCO, Luis Miguel; *Framework .NET*; España 2002, Grupo Eidos

[05] BOOCH, Grady; Rumbaugh, James; JACOBSON, Ivar; *El Lenguaje Unificado de Modelado.*; España 1999, Addison Wesley

[06] BRUEGGE, Bernd; Dutoit, Allen H.; *Ingeniería de software orientado a objetos*; México 2002 Prentice Hall

[07] BURTON, Kevin; *.NET Common Language Runtime Unleashed*; United Status of America 2002, Sams Publishing

[08] CHARTE, Francisco; *Programación con Visual Basic .NET*; España 2002, Anaya Multimedia

[09] CHARTE, Francisco; *Programación de bases de datos con Visual Basic .NET*; España 2002, Anaya Multimedia

[10] CHAPPELL, David; *Understanding .NET: A Tutorial and Analysis*; United States of America 2002, Addison Wesley

[11] COCKBURN, Alistair; *Agile Software Development*; United States of America 2001, Addison-Wesley

[12] DATE, C. J.; *Introducción a los sistemas de bases de datos*; 7ª edición, México 2001, Prentice Hall

[13] DELANEY, Kalen; *A fondo Microsoft SQL Server 2000*; España 2001, McGraw – Hill

[14] FOWLER, Martin; *UML gota a agota*; México 1999, Addison Wesley

[15] FOWLER, Martin, Beck, Kent, Brant, J.; *Refactoring: Improving the Design of Existing Code;* United States of America 1999, Addison-Wesley

[16] FROHOCK García, Marci; Reding, Jaime; *Guía completa de Microsoft SQL Server 2000*; España 2001, McGraw – Hill

[17] GROFF, James R.; Weinberg, Paul N.; *SQL. Manual de referencia*; España 2003, McGraw – Hill

[18] HIGHSMITH Jim, Orr K.; *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems;* United States of America 2000, Dorset House

[19] HIGHSMITH, Jim; *Agile Software Development Ecosystems*; United States of America 2002, Addison-Wesley

[20] JEFFRIES, Ron, Anderson, A., Hendrickson, C.; *Extreme Programming Installed*; United States of America 2001, Addison-Wesley

[21] KENDALL, Kenneth E.; Kendall, Julie E.; *Análisis y diseño de sistemas*; 3ª edición, México 1997, Pearson Educación

[22] LARMAN, Craig; *Agile and Iterative Development: A Manager's Guide*; United States of America 2003, Addison Wesley

[23] LARMAN, Craig; *UML y Patrones*; 2ª edición, España 2003, Prentice Hall

[24] MEYER, Bertrand; *Construcción de software orientado a objetos*; 2ª edición, España 1999, Prentice Hall

[25] *MCAD/MCSD Self-Paced Training Kit : Developing Windows-Based Applications with Microsoft Visual Basic .NET and Microsoft Visual C# .NET / Microsoft Corporation.*; United States of America 2002, Microsoft Press

[26] McCONNELL, Steve; *Desarrollo y gestión de proyectos informáticos*; España 1997, McGraw – Hill

[27] NEWKIRK, James, Martin Robert C.; *Extreme Programming in Practice*; United States of America 2001, Addison-Wesley

[28] PLATT, David, S.; *Así es Microsoft .NET*; España 2001, McGraw – Hill

[29] PFLEEGER, Shari L.; *Ingeniería de software. Teoría y práctica*; Argentina 2002, Prentice Hall

[30] POPPENDIECK M., Poppendieck T.; *Lean Software Development: An Agile Toolkit for Software Development Managers;* United States of America 2003, Addison Wesley

[31] PRESSMAN, Roger S.; *Ingeniería del software. Un enfoque práctico*; 4ª edición, España 1998, McGraw-Hill

[32] PROSISE, Jeff; *Programming Microsoft .NET (core reference)*; United States of America 2002, Microsoft Press

[33] SHAPIRO, Jeffrey; *Visual Basic .NET. Manual de referencia*; España 2003, McGraw – Hill

[34] SILBERSCHATZ, Abraham; Korth, Henry F.; Sudarshan, S.; *Fundamentos de bases de datos*; 3ª edición, España 1998, McGraw – Hill

[35] SOMMERVILLE, Ian; *Ingeniería de software*; 6ª edición, México 2002, Addison Wesley

[36] RUBIOLO, Daniel; MEIER, J.D., JEZIERSKI, Edward, MACKMAN, Alex *Microsoft .NET Explained: Cambio de paradigma a la computación distribuida a través de Internet* 2001

[37] RUBIOLO, Daniel; MEIER, J.D., JEZIERSKI, Edward, MACKMAN, Alex *Microsoft .NET Explained: ¿Qué es Microsoft® .NET?;* 2001

[38] RUBIOLO, Daniel; MEIER, J.D., JEZIERSKI, Edward, MACKMAN, Alex *Microsoft .NET Explained: Por qué Microsoft® .NET;* 2001

[39] UTLEY, Craig; *A Programmer's Guide to Visual Basic.NET*; United States of America 2001,

Sams Publishing

[40] WAKE, William.C.; *Extreme Programming Explored*; United States of America 2002, Addison-Wesley

REFERENCIAS WEB

ECMA
http://www.ecma-international.org
Mayo del 2005

Fowler, M.
The New Methodology
http://www.martinfowler.com/articles/newMethodology.html
Junio del 2005

JUnit
http://www.xprogramming.com
Junio del 2005

La alianza ágil
http://www.agilealliance.com
Mayo del 2005

Manifiesto ágil
http://www.agilemanifesto.org/
Junio del 2006

Mono Developer
http://www.monodevelop.org
Junio del 2005

Proyecto DotGNU
http://www.gnu.org/software/dotgnu/
Junio del 2005

Proyecto Mono
http://www.mono-project.com
Junio del 2005

SharpDevelop
http://www.sharpdevelop.com
Junio del 2005

Wikipedia
http://es.wikipedia.org
Junio del 2005

OBJETIVOS

OBJETIVOS

OBJETIVO GENERAL

Desarrollar un sistema que permita la elaboración de propuestas de movimientos en materias y horarios para los profesores de la carrera de Ingeniería en Computación de una manera más eficiente en la FES Aragón.

OBJETIVOS ESPECÍFICOS

- Analizar el proceso de elaboración de propuestas de movimientos en materias y horarios identificando su problemática en la FES Aragón.
- Investigar los principios de las metodologías ágiles y la programación extrema para utilizarla en el desarrollo de una aplicación software.
- Estudiar la plataforma .NET para el desarrollo de aplicaciones software.
- Desarrollar un software que permita la automatización de la elaboración de propuestas.