Universidad de las Ciencias Informáticas

Facultad 2

Título: Estimación de la duración de proyectos de software de gestión apoyado en Razonamiento Basado en Casos.

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

Autores: Yanelis Prada Gómez.
Yudelis García Novales.

Tutor: Msc. Yadira Ruiz Constanten.
Co-tutor: Ing. Dasiel Cordero Morales.

Ciudad de la Habana, Junio de 2013.

"Si quieres ser sabio, aprende a interrogar razonablemente, a escuchar con atención, a responder serenamente y a callar cuando no tengas nada que decir".

Johann Kaspar Lavater

DECLARACIÓN DE AUTORÍA.

Declaramos que somos los únicos autores de este trabajo y autorizamos al centro de Informatización para la Seguridad Ciudadana de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____	_____
Yudelis García Novales	Yanelis Prada Gómez
_____	_____
Msc. Yadira Ruíz Constanten	Ing. Dasiel Cordero Morales

RESUMEN

Ante la necesidad de realizar un buen proceso de estimación de proyectos de software en la Universidad de las Ciencias Informáticas (UCI) y aprovechar la experiencia adquirida, surge la idea de realizar las estimaciones partiendo de la recopilación de datos referentes a proyectos desarrollados con anterioridad. Para ello, el presente trabajo de diploma: “Estimación de la duración de proyectos de software de gestión apoyado en Razonamiento Basado en Casos”, persigue como objetivo principal la implementación de una aplicación capaz de estimar proyectos, estableciendo comparaciones entre las características de un proyecto nuevo con otros ya concluidos. Se emplearán tecnologías o técnicas que ofrece la Inteligencia Artificial (IA), específicamente el Razonamiento Basado en Casos. El sistema propuesto brindará facilidades para realizar estimaciones de proyectos de desarrollo de software lo más reales posibles, utilizando conocimiento y experiencia acumulada. Además permitirá recuperar los proyectos semejantes al nuevo, desde la base de casos, la que a su vez almacenará los datos históricos de los proyectos concluidos. Con dicha propuesta contribuirá a obtener mejores resultados en la estimación de la duración de los proyectos de gestión de la UCI, que conllevaría a lograr productos de mejor calidad, además permitirá a los planificadores realizar una mejor planificación de los recursos con los que se cuenta para el desarrollo del software.

Palabras claves: estimación, gestión, razonamiento basado en casos, proyecto de software.

ÍNDICE.

ÍNDICE DE TABLAS.

ÍNDICE DE FIFURAS.

INTRODUCCIÓN.

En la industria del software la cantidad de esfuerzo empleado en su desarrollo continúa de manera similar desde hace algunos años y los productos a menudo son entregados con errores significativos, por lo que se exige una buena planificación.

Un proyecto es un conjunto único de actividades necesarias para producir un resultado definido, en un rango de fechas determinado y con una asignación específica de recursos (1). Los proyectos bien ejecutados pasan por tres etapas básicas para crear una planificación software. Primero se estima el tamaño del producto, luego el esfuerzo necesario para construir un producto con este tamaño y por último la duración cronológica del proyecto (2).

La estimación es una de las actividades de la planificación. Su objetivo es conocer en etapas tempranas y de manera aproximada, el costo, la duración y los recursos necesarios para el desarrollo de los proyectos. La estimación de proyectos de software nunca será una ciencia exacta ya que existen numerosas variables humanas, técnicas, del entorno y políticas, que intervienen en su proceso y pueden afectar los resultados finales, pero una combinación de buenos datos históricos y técnicas sistemáticas puede mejorar la precisión de la estimación (3).Esta actividad no debe llevarse a cabo de forma descuidada puesto que es la base de todas las actividades de planificación de un proyecto.

La importancia que está tomando la información ha determinado que sea la duración de un proyecto uno de los aspectos más prioritarios en la estimación. Se ha hecho imposible establecer una unidad de medida para la estimación del tiempo de duración de un proyecto dependiendo de sus características.

Las nuevas tecnologías o técnicas que ofrece la Inteligencia Artificial (IA) serían una alternativa para lograr una buena estimación de la duración de proyectos de software. La IA es una rama de la Ciencia de la Computación dedicada a la creación de hardware y software que intenta producir resultados similares a los expresados por los humanos (4). Sus enfoques abordan el modelado, con base en distintas arquitecturas, de distintos procesos propios del pensamiento humano tales como: la toma de decisiones, el razonamiento o el aprendizaje.

Existen diversas técnicas de razonamiento dentro de esta ciencia que permiten utilizar el conocimiento para dar solución a disímiles problemas, una de las más empleadas son los Sistemas Basados en

Conocimiento (SBC). Los SBC son una técnica que desarrollan herramientas inteligentes que simulan el pensamiento de una persona experimentada en un dominio. Su orientación es la automatización del análisis de problemas, la búsqueda de soluciones, la toma de decisiones y el empleo de conocimiento especializado en un campo específico de aplicación (5).

La Universidad de las Ciencias Informáticas (UCI), desde sus inicios en la producción de software ha mostrado varios inconvenientes relacionados con la estimación de la duración de proyectos concebida en la planificación. Se debe señalar que las estimaciones no siempre se realizan por el personal especializado en el tema, lo que conlleva a resultados poco confiables. El reto que se impone es grande, ya que para tomar decisiones acertadas, se hace necesario que el personal que labora en el desarrollo del software mantenga una estimación y planificación constante del mismo.

En entrevistas realizadas a líderes de proyectos y a personas encargadas de realizar la planificación de los mismos, en 8 centros de desarrollo de software de un total de 16 en la universidad, dedicados al desarrollo de soluciones de gestión para informatizar los procesos de disímiles ramas tales como salud, telecomunicaciones, seguridad ciudadana, educación, informática industrial, entre otros, hubo diversidad de criterios relacionados con la estimación de los proyectos.

Estos criterios extraídos de las entrevistas coinciden en que los problemas más comunes en el desarrollo de sus soluciones son los siguientes:

Falta de comunicación entre el equipo y el cliente provocando un levantamiento de requisitos incompletos o de mala calidad. Esto causa una estimación de esfuerzo inapropiada, una pobre estimación de recursos, mala preparación de los integrantes del equipo y actividades del proyecto tanto sobrestimada como subestimada.

La variabilidad de los requisitos por parte del cliente ocasiona inestabilidad en el costo y el programa de trabajo. La falta de recursos humanos y materiales; la carga docente y extracurricular de los integrantes del equipo del proyecto y su doble vinculación docente / productiva; los eventos externos al proyecto (eventos científicos, problemas con la electricidad, rotura de algún recursos materiales); en algunos casos el conocimiento insuficiente del equipo de desarrollo sobre el tema; así como el poco dominio de la tecnología con la que se va a interactuar conllevan a un atraso en el cumplimiento de los objetivos trazados y por tanto su influencia directa afecta el cronograma. El efecto de los riesgos no identificados en

el proyecto impide conocer qué sucederá en caso que se presente una situación desfavorable previamente vista o no. Además influye en el calendario del proyecto y en la entrega final del producto provocando que las estimaciones obtenidas se alejen del resultado esperado.

Los elementos resultantes de la entrevista realizada permitieron concluir que a pesar de la existencia de métodos de estimación reconocidos internacionalmente tales como Puntos de Casos de Usos, Puntos de Función, COCOMO 81 y COCOMO II no son aplicados en muchos casos en los proyectos de desarrollo de software de la UCI. La universidad cuenta con un método propuesto por CALISOFT para la estimación de la duración de proyectos, que no es utilizado en todos los proyectos, de doce líderes encuestados, lo usan cinco, dos de los proyectos usan el método de Puntos por Casos de Usos, uno solo usa el método de COCOMO II, cuatro de ellos no realizan estimaciones debido a que se rigen por los tiempos pactados con el cliente y el otro proyecto basa sus estimaciones en las experiencias obtenidas en proyectos similares resueltos en el pasado como única vía. Si un proyecto es bastante similar en tamaño y función a un proyecto terminado es probable que el nuevo proyecto requiera la misma cantidad de esfuerzo, o que dure aproximadamente lo mismo que el proyecto anterior. Si el proyecto es totalmente distinto, entonces puede que la experiencia obtenida no represente un buen indicador.

En la UCI, en un intento de representar los elementos distintivos de la estimación del tiempo de duración de los proyectos de desarrollo de software se definió una Base de Casos (BC), con el objetivo de guardar los resultados relacionados con el tiempo real que demora un proyecto de software, para utilizarlos en estimaciones futuras. El resultado lo constituyó una BC con 113 rasgos. Haciendo un análisis crítico de la propuesta se considera que esto constituye un número alto de indicadores, por lo que se hace necesario reestructurar en gran medida estos rasgos que resultan innecesarios o dificultan en alguna medida el entendimiento del sistema por parte del usuario final y es muy costoso en cuanto al tiempo que ocuparían en llenarlos. Estos indicadores resultan imprescindibles a la hora de llevar a cabo dichas estimaciones, pero sería factible agruparlos de forma más específica, derivarlos u obtenerlos de cálculos de otros para un mejor manejo del sistema.

Se diseñó además un modelo de motor de inferencia en correspondencia con las características de la base de casos propuesta, por lo que es muy probable que un cambio en la BC repercuta de forma directa en la estructura del modelo propuesto. Este es definido con el objetivo de obtener las características similares de proyectos anteriores como base para dar respuesta al nuevo proyecto. Sin embargo este

motor de inferencia no cuenta con un mecanismo de aprendizaje que permita analizar las nuevas estimaciones y determinar si es factible su almacenamiento para ser utilizadas en un futuro y así lograr una retroalimentación del conocimiento alcanzado y completar el comportamiento de un Sistema de Razonamiento Basado en Caso (SRBC).

Teniendo en cuenta la problemática anterior surge como ***problema a resolver***: La inadecuada gestión del tiempo de duración de un proyecto de desarrollo de software debido a la omisión o poco uso de información histórica, sustentada en la experiencia y el conocimiento almacenado de estimaciones anteriores, incide en la ocurrencia de errores en el proceso de planificación.

Con vistas a su solución, se actúa sobre el **objeto de estudio**: El proceso de Planificación en los proyectos de desarrollo de Software.

En aras de resolver el problema planteado se tiene como ***objetivo general***: Construir un Sistema Inteligente para estimar la duración de un proyecto de desarrollo de software de gestión utilizando la experiencia y el conocimiento almacenados a partir de estimaciones anteriores.

Como ***campo de acción*** se define: Sistemas Basados en el Conocimiento para la estimación de la duración de un proyecto de desarrollo de software.

Preguntas de investigación:

- ¿Cuáles son las principales tendencias, limitaciones y ventajas en los sistemas de software para estimar la duración de Proyectos de Desarrollo de Software?
- ¿Cuáles son los métodos de estimación utilizados para determinar la duración de proyectos de software?
- ¿Los métodos de estimación que se utilizan actualmente son factibles para determinar la duración de un proyecto de software con mayor veracidad en el resultado?
- ¿Cuáles técnicas de IA son utilizadas para solucionar problemas de estimación en proyectos de software?

- ¿Cuáles sistemas que apoyados en la técnica de RBC son utilizados para resolver problemas de estimación en proyectos de software?
- ¿Cómo perfeccionar la BC propuesta en el curso anterior 2011-2012 por (6), para estimar la duración de un proyecto de software en la Universidad de las Ciencias Informáticas?
- ¿Cómo mejorar el motor de inferencia propuesto, como parte del SRBC, para estimar la duración de un proyecto de software en la Universidad de las Ciencias Informáticas?
- ¿Cuáles técnicas de IA permiten desarrollar mecanismos de aprendizaje automático?

Obteniéndose como ***objetivos específicos***:

- Elaborar el marco teórico de la investigación identificando las principales tendencias, limitaciones y ventajas de los sistemas de software para estimar la duración de Proyectos de Desarrollo de Software.
- Formalizar un modelo computacional donde se apliquen técnicas de razonamiento basado en casos en la consideración de la estimación de la duración de un proyecto de desarrollo de software de gestión como entidad principal.
- Desarrollar el modelo propuesto cumpliendo con estándares de codificación y calidad definidos en la UCI.
- Validar el modelo desarrollado a partir de una prueba piloto.

Para cumplir con los objetivos trazados se tiene como ***tareas de investigación***:

- Caracterización de la situación existente en el mundo, en la región y en el país en particular sobre la estimación de la duración de un proyecto de software y la técnica de razonamiento basado en casos para definir la posición de los investigadores.
- Análisis de sistemas que utilicen la técnica de RBC para resolver problemas de estimación en proyectos de software.

- Revisión y análisis crítico de la BC propuesta para estimar la duración de un proyecto de software en la Universidad de las Ciencias Informáticas.
- Revisión y análisis crítico del motor de inferencia propuesto, como parte del SRBC, para estimar la duración de un proyecto de software en la Universidad de las Ciencias Informáticas.
- Análisis de diferentes técnicas de IA que permitan desarrollar mecanismos de aprendizaje automático.
- Análisis de los estándares de codificación y calidad definidos en la UCI a ser utilizados para desarrollar la solución.

Dentro de los métodos aplicados para llevar a cabo la investigación se encuentran los siguientes:

Métodos Teóricos

Analítico–Sintético: Este método es utilizado para realizar el análisis de documentos y teorías, permitiendo la extracción de los elementos más importantes que se relacionan con la estimación de proyectos de desarrollo de software. Además permite analizar a través de una profunda búsqueda, los métodos que existen en la actualidad para estimar la duración de proyectos software, posibilitando en la investigación caracterizar los mismos así como sus principales relaciones. También se emplea este método para profundizar en las tecnologías, herramientas inteligentes y metodologías a utilizar en el desarrollo de la aplicación.

Inductivo-Deductivo: En el desarrollo de la investigación, este método fue empleado para obtenerlas diferentes características y problemas existentes en la estimación del tiempo de la duración de proyectos de software. Con estos elementos obtenidos se deduce que es necesario el desarrollo de una herramienta Inteligente, factible para todas las entidades de desarrollo de software, que permita estimar la duración de un proyecto de software utilizando la experiencia y el conocimiento almacenados a partir de estimaciones anteriores.

Histórico-Lógico: El método se emplea para comprobar la evolución de las tendencias actuales de la estimación de proyectos de desarrollo de software, las técnicas de inteligencia artificial que utilizan la experiencia acumulada y sus posibles puntos de contacto. Se utiliza además para conocer la lógica del

desarrollo de los métodos de estimación del proyecto y la relación con las herramientas inteligentes obteniendo las características esenciales y la vinculación que existen entre ellas.

Métodos Empíricos

Los métodos empíricos utilizados permiten efectuar el análisis preliminar de la información, así como verificar y comprobar las concepciones teóricas. Dentro de este grupo se utiliza:

Entrevista: Se le realizaron entrevistas a los líderes de proyecto de los diferentes centros existentes en la UCI para hacer un levantamiento del estado actual de las estimaciones y los métodos utilizados para realizarlas. Además se entrevistaron expertos para determinar los indicadores más importantes del sistema propuesto y hasta qué punto el sistema favorece al proceso de estimación de los proyectos de Gestión.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

Introducción del capítulo.

En este capítulo se realiza la fundamentación teórica del tema a desarrollar a través de un estudio del estado del arte de la estimación de software y los sistemas basados en conocimiento. Se exponen algunos conceptos relacionados con el tema para un mejor entendimiento de la propuesta de solución. Se hace un análisis de las tendencias actuales de la estimación de proyectos de software, así como de métodos existentes para realizar estas estimaciones. Se caracteriza la situación existente en el mundo, el país y la universidad, en particular sobre herramientas de estimación. Se analizan los diferentes tipos de sistemas basados en conocimiento y se hace un estudio de los sistemas basados en casos. Además se describen las principales características de las herramientas y metodologías para la realización de la propuesta de solución.

1.1 Planificación de proyectos de software.

La planificación de un proyecto de software no difiere de la planificación de cualquier proyecto de ingeniería. Se identifican una serie de tareas del proyecto y se establecen interdependencias entre ellas (7).Se estima el esfuerzo asociado con cada tarea y se hace la asignación del personal y de otros recursos para obtener una mejor organización en el proceso de desarrollo de software. Además se crea una red de tareas y se desarrolla un cronograma de trabajo.

El planificador del proyecto de software debe estimar tres factores antes de que un proyecto comience: cuánto tiempo tomará, cuánto esfuerzo requerirá y cuánto personal estará involucrado. Además, debe predecir los recursos (hardware y software) que se requerirán y el riesgo involucrado (3).

1.1.1 Estimación de proyectos de software.

La primera tarea en la planificación de proyectos es la estimación. Consiste en predecir en fases iniciales del ciclo de vida, características fundamentales del software cuyo valor real solo puede conocerse en etapas posteriores o cuando el proyecto ha finalizado. Ana Mª Moreno Sánchez Capuchino (8) la define como el proceso que proporciona un valor a un conjunto de variables para la realización de un trabajo, dentro de un rango aceptable de tolerancia.

La estimación será segura cuando más elementos se tengan disponibles. A medida que se avance en el proyecto se contará con más datos, menor será la incertidumbre y se harán estimaciones más precisas que permitirá una mejor planificación de lo que resta del proyecto; pero el objetivo de esta es predecir lo que ocurrirá, y dejará de cumplirlo cuando más adelantada se realice; es por eso que se debe estimar en etapas tempranas aun cuando el margen de error es mayor.

Es razonable conocer, antes de comenzar a desarrollar el software, cuánto se va a invertir, qué tareas se deben realizar y cuánto tiempo se necesitará, no tiene sentido iniciar un proyecto que está destinado al fracaso por no contar con estos valores. En la actualidad son muchos los proyectos que fracasan, e incumplen sus plazos de entrega por esta causa. En la figura 1 se muestra un gráfico con el éxito de proyectos de software por año según el Standish Group en el 2010.

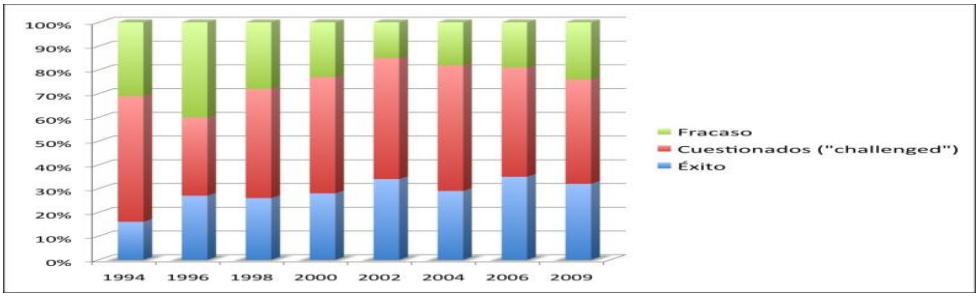

Figura 1: Éxito de proyectos de software según StandishGroup (7).

1.1.2 Tendencias actuales de la estimación en el desarrollo de software.

Antes de 1970, la estimación de esfuerzo se realizaba manualmente mediante el uso de reglas de pulgar o algoritmos, algunas de las cuales estaban basadas en prueba y error. En la década de los 70 fue un período importante para predecir los costos y cronogramas de desarrollo de software. A principios de este año, la primera herramienta de estimación de software automatizada se había construido. Para el año 1977, Frank Freiman diseñó el modelo de estimación de software PRECIO-S; esta fue la herramienta comercial usada en los Estados Unidos. En 1979, SLIM (LifeCycle Software Model), fue presentado a la US-Market por Lawrence H. Putnam basado en la curva de Norden Rayleigh. Además se comenzó a emplear la función de análisis de puntos, para estimar el tamaño y el esfuerzo de desarrollo. Este indicador se basa en cinco diferentes atributos: Entradas, Salidas, Consultas, Archivos lógicos e Interfaces.

Allan Albrecht publicó un artículo con el método de FPA (Function Point Analysis) en 1979, el cual agudizó las reglas para la calificación de la complejidad del software. Tom DeMarco en 1982, publica un libro ("Controlar los proyectos de software"), en la cual presenta una métrica funcional que hereda algunos de los rasgos de punto de función de Albrecht, pero se desarrolló de forma independiente. En 1983, Charles Symons, un investigador británico especialista en la estimación de software, presentó al mercado las métricas de puntos de función. IBM en 1984, hizo una revisión a fondo de su métrica de puntos de función, que es la base de los puntos de función de hoy en día. En 1985, Caper Jones amplió el concepto de punto de función para incluir el efecto de los algoritmos computacionalmente complejos.

En la década de los 80, Barry Boehm, en la Universidad del Sur de California comenzó a revisar y ampliar el concepto de modelo original COCOMO. Michel Genuchten y Koolen Hans, en 1991 desarrollaron una serie de métodos y herramientas que a lo largo de varios años lograron satisfacer la creciente necesidad de controlar el desarrollo de software. Betteridge, R. trabajó en el software de cálculo de costes. Se introdujo en 1993 la nueva versión del COCOMO llamado COCOMO 2.0, que surgió en 1994.

Al comienzo del siglo veintiuno, fue propuesto un nuevo enfoque basado en el razonamiento por analogía y cuantificadores lingüísticos que fueron utilizados para estimar el esfuerzo. En el 2002, M.Jorgensen, propuso la estimación de expertos que se convirtió en la estrategia de estimación más utilizada para los proyectos de software. Para el2003, Ahn Yunsik, Suh Jungseok, Kim y Kim Seungryeol Hyunsoo, propusieron SMPEEM (Mantenimiento de Proyectos de Software Estimación de Esfuerzo). En 2006, Stein Grimstad, la estimación de esfuerzo fue utilizado con frecuencia sin una aclaración suficiente sobre su significado, y que la precisión de la estimación se evalúa a menudo sin asegurarse de que el esfuerzo estimado y el real son comparables (9).

En el 2012 se propusieron en las literaturas muchos métodos para medir el tamaño y el esfuerzo de software, pero estos no fueron adoptados ampliamente en la práctica. Gran cantidad de las herramientas de estimación de costos comerciales de software quedaron puestas en libertad hasta la actualidad, permitiendo así un mayor acceso a las mismas. Desde hace algunos años, la estimación de proyectos de software se ha integrado con técnicas de IA, siendo estas últimas de gran apoyo pues permiten estimar con una mayor precisión basada en experiencias anteriores.

La International Journal of Computer Applications en su publicación Comparative Analysis of Software Effort Estimation Techniques (9), refleja una panorámica de cómo ha sido el comportamiento de la

estimación de software desde los inicios en la construcción de sistemas, así como la necesidad de la estimación.

1.1.3 Métodos de estimación de proyectos de software.

Los métodos de estimación para proyectos pueden son clasificados de manera sencilla por tres categorías básicas: Juicio del experto, Analogía y Modelos algorítmicos.

- **Juicio del experto**

El juicio de experto está basado en la experiencia profesional de los participantes en el proyecto sobre un dominio específico, se suele utilizar la opinión de varios expertos para mayor fiabilidad. En algunos casos, se apoyan en datos objetivos obtenidos de proyectos anteriores y almacenados (2). Ejemplo de métodos más específico es el Delphi y el Delphi de banda ancha.

La desventaja de este método es el alto costo en tiempo y recursos humanos necesarios para su implantación, así como la subordinación al nivel de experiencia y conocimientos en el entorno que puedan aportar los técnicos. Las ventajas que posee indican que las estimaciones parciales son neutralizadas y se presenta una estimación global. Por otro lado las estimaciones suministradas por este grupo de expertos difícilmente pueden ser obviadas gracias a la trascendencia que la organización otorga a este proceso, al proporcionar costosos recursos a esta tarea (2).

- **Analogía**

Este método es una variante formal de juicio de expertos en la que se compara el proyecto con uno o más proyectos ya terminados de los que existen datos reales y se deduce una estimación en función de las similitudes y diferencias. En la analogía pueden variar los siguientes factores como el tamaño, complejidad, usuarios. Utiliza medidas de los atributos del modelo empírico a fin de caracterizar el caso actual, para el que se realiza la estimación. Las medidas conocidas para el caso actual son usadas para buscar un conjunto de datos que identifiquen casos análogos. La predicción se hace intercalando desde uno o varios casos análogos al caso actual (2).

Para utilizar este método es necesario disponer de una base de datos histórica de proyectos finalizados con la que poder realizar la comparación. Además todos esos proyectos tendrán que haber seguido un proceso estándar. Es decir, el ciclo de vida utilizado y las actividades han de ser similares. Si no es así, es

difícil hacer comparaciones proyecto-proyecto. Otro aspecto a tener en cuenta es que los datos sobre esos proyectos han de ser fiables. Esto quiere decir que las empresas correspondientes han de tener un programa formalizado para la toma de medidas sobre sus proyectos (8).

Las ventajas que proporciona este método es un menor costo en tiempo y recursos que el método del juicio del experto. Como desventajas cabría destacar que las estimaciones de proyectos anteriores no siempre se ajustan a nuevos proyectos, ya que muchos de los factores de estas estimaciones no siempre se mantienen (2).

- **Modelos Algorítmicos**

Se basan en fórmulas y parámetros con los que realizan las estimaciones. Son también muy conocidos y empleados en la actualidad, encontrándose en este grupo el modelo COCOMO, puntos de función y puntos por casos de usos.

- **Puntos de Función**

Puntos de Función es un método creado 1979 por Allan Albrecht y refinado en 1984 que intenta medir el tamaño de un sistema software. Se basa principalmente en la identificación de los componentes del sistema informático en términos de transacciones y grupos de datos lógicos que son relevantes para el usuario en su negocio. A cada uno de estos componentes les asigna un número de puntos por función basándose en el tipo de componente y su complejidad; y la sumatoria de esto, da los puntos de función sin ajustar. El ajuste es un paso final basándose en las características generales de todo el sistema informático que se está contando.

Los objetivos de los Puntos de Función son (8):

- Medir lo que el usuario pide y lo que el usuario recibe.
- Medir independientemente de la tecnología utilizada en la implantación del sistema.
- Proporcionar una métrica de tamaño que dé soporte al análisis de la calidad y la productividad.
- Proporcionar un medio para la estimación del software.
- Proporcionar un factor de normalización para la comparación de distintos software.

El análisis de los Puntos de Función se desarrolla considerando cinco parámetros básicos externos del sistema:

1. Entrada (EI, del inglés External Input).
2. Salida (EO, del inglés External Output).
3. Consultas (EQ, del inglés ExternalQuery).
4. Grupos de datos lógicos internos (ILF, del inglés InternalLogic File).
5. Grupos de datos lógicos externos (EIF, del inglés External Interface File).

Con estos parámetros, se determinan los puntos de función sin ajustar. A este valor, se le aplica un Factor de Ajuste obtenido en base a unas valoraciones subjetivas sobre la aplicación y su entorno, es decir, las características generales del sistema. Después de obtener los puntos función hay que hacer una traducción al esfuerzo que supone, contemplándose la experiencia y estilo de programación basado en históricos.

Los puntos de función aparecen con ventajas substanciales por sobre las líneas de código, para fines de estimación temprana del tamaño del software, y por ende, del esfuerzo de desarrollo. Además es una medida ampliamente utilizada, y con éxito, en muchas organizaciones que desarrollan software en forma masiva (2).

- **COCOMO**

COCOMO fue publicado por Barry Boehm en 1981, se basa en ecuaciones no lineales obtenidas mediante técnicas de regresión sobre un histórico de proyectos (9). Tiene en cuenta el tamaño expresado en líneas de código, esfuerzo y duración de un proyecto de software. Existe el modelo COCOMO II que es una mejora del modelo original creado en la Universidad al Sur de California.

Existen tres modos de desarrollo de software según COCOMO: orgánico, semilibre y rígido, teniendo en cuentalas características de la aplicación y del entorno de desarrollo. A cada uno de estos modelos se le puede aplicar tres métodos de estimación distintos: básico, intermedio y detallado.

Modelos COCOMO

- Modelo Básico.

Se suele aplicar en los desarrollos de productos pequeños/medios, desarrollados por personal de la propia empresa en modo orgánico. Aunque también puede aplicarse al resto de los modos. Calcula el esfuerzo y costo del desarrollo de software en función del tamaño expresado en miles de líneas de código (KLOC).

Permite una estimación durante la fase de análisis previo cuando la mayoría de los factores se desconocen.

- Modelo Intermedio.

Se aplica cuando el proyecto se ha dividido en subsistemas. El modelo intermedio incorpora 15 variables de predicción que influyen en el coste del proyecto. Estas variables se agrupan en cuatro categorías: atributos del producto software, atributos de la computadora, atributos de personal y atributos del proyecto.

- Modelo Detallado.

El modelo detallado presenta una jerarquía del producto a tres niveles: nivel módulo, de subsistema y de sistema. Proporciona un conjunto de multiplicadores de esfuerzo para cada atributo en cada fase. Estos multiplicadores determinan el esfuerzo requerido para completar cada fase. Toma como base el submodelo Intermedio considerando el impacto de los atributos del proyecto en cada una de las etapas del ciclo de vida.

Consiste básicamente en la aplicación de ecuaciones matemáticas sobre los Puntos de Función sin ajustar o la cantidad de líneas de código (SLOC, SourceLines Of Code) estimados para un proyecto. Estas ecuaciones se encuentran ponderadas por ciertos factores de costo (cost drivers) que influyen en el esfuerzo requerido para el desarrollo del software.

Estos modelos presentan algunas limitantes según Ana María Moreno Capuchino (8). El modelo básico no incorpora el efecto de los factores, como la experiencia de los recursos; que influyen sobre el costo y el mantenimiento del producto. El modelo intermedio tiene dos limitaciones que pueden ser significativas en la estimación detallada de costo en grandes proyectos de software: la distribución de esfuerzo por fases puede ser inadecuada y puede ser muy engorroso utilizarlo en un producto con muchos componentes. El modelo detallado no se desarrolla debido a que su aplicación queda reservada a grandes proyectos no muy frecuentes.

- **Puntos por Casos de Usos**

La estimación mediante el análisis de Puntos de Casos de Uso es un método propuesto originalmente por Gustav Karner de Objectory AB, y posteriormente refinado por muchos otros autores. Se trata de un

método de estimación del tiempo de desarrollo de un proyecto de asignación de “pesos” a un cierto número de factores que lo afectan, para finalmente, contabilizar el tiempo total estimado para el proyecto a partir de esos factores (2).

El primer paso para la estimación consiste en el cálculo de los puntos de Casos de Uso sin ajustar (UUCP), dependiendo del factor de peso de los actores sin ajustar y el factor de peso de los casos de usos sin ajustar. Luego se estudian y calculan los factores de complejidad técnica y los factores del ambiente para crear los factores de ajustes y así finalmente obtener los puntos de casos de usos ajustados (UCP).

Entre los problemas que presenta este método está la inexistencia de un estándar para describir casos de uso, eso hace que aplicar la métrica sea más engorroso. No se distingue un criterio único aplicado para el cálculo de la complejidad de un caso de uso. Además, también existen diferentes criterios para identificar una transacción (10).

- **Método de estimación UCI**

El desarrollo del método empleado en la UCI partió de la necesidad de estimar el tamaño, costo y esfuerzo requerido para desarrollar un producto software. Los elementos que se tuvieron en cuenta para la descripción del método de estimación fueron los tiempos a emplear y el esfuerzo de cada rol, en dependencia de la complejidad de los casos de uso, así como una descripción de las principales actividades que se realizan dentro de las etapas de levantamiento de requisitos y prototipo (2). La complejidad de los casos de uso se determinó de una manera empírica e intuitiva, pues en la universidad no se contaba con una vasta experiencia, ni con una base histórica en las estimaciones de proyectos. Estos elementos fueron tenidos en cuenta por datos dispersos de algunos proyectos y la evaluación de algunos factores que, según criterio de expertos, pueden influir en las estimaciones del proyecto.

1.1.4 Estimación de duración de proyectos de software.

El resultado de una estimación de la duración de proyecto de software se representa en unidades temporales: días, semanas, meses, años.

La duración de una actividad en el proceso de desarrollo de software está determinada por cuatro factores (11):

- El volumen de trabajo a realizar.
- La cantidad de recursos necesarios.
- La disponibilidad de dichos recursos.
- La productividad en la utilización de los recursos.

En el caso de proyectos de software el recurso fundamental es la mano de obra de ingenieros software, analistas, programadores u otros miembros del equipo.

1.1.5 Herramientas para la estimación de proyectos de software.

En el mundo:

- **SEER by Galorath:** Galorath comenzó como una empresa de consultoría en 1979 en EE.UU, comprometiéndose a ayudar al gobierno y la industria para mejorar su software y hardware de desarrollo y gestión de programas. Aprovechando la tecnología de modelado sofisticado y aplicables en proyectos bases de conocimiento, las soluciones SEER son probadas para reproducir con precisión el mundo real los resultados de los proyectos más rápidamente y mucho mayor que cualquiera de las metodologías tradicionales de estimación (12).

- **SEER para software:** SEER para programación (SEER-SEM) es una poderosa herramienta de apoyo a las decisiones de desarrollo de software para la estimación de los costos y el mantenimiento, el trabajo, asignación de personal, calendario, la fiabilidad, y el riesgo en función del tamaño, tecnología, y cualquier condición del proyecto. SEER para la estimación de Software y análisis (conjunto de aplicaciones) ofrece una solución integrada que apoya todas las fases del ciclo de vida del proyecto de software (12).

- **SEER para IT:** SEER para Tecnología de Información (SEER-IT), es un software para la estimación de los proyectos de IT, ha sido diseñado para profesionales de IT, permitiendo a las organizaciones desarrollar un temprano y preciso asesoramiento de los costos del proyecto, horarios, y los riesgos, así como apoyo, ayudando a maximizar la productividad y la producción fija o con la disminución de los presupuestos (12).

- **SLIM-Estimate:** Software Lifecycle Management (SLIM)-Estimate calcula el costo, el tiempo y esfuerzo necesario para satisfacer un conjunto de requisitos del sistema y determinar la mejor estrategia para el diseño e implementación de su software o proyecto de sistemas. Además de la

estimación de costos de software, este sistema ofrece un alto grado de configurabilidad para adaptarse a los diferentes procesos de diseño utilizados por los desarrolladores de hoy: como el desarrollo ágil, implementación de paquetes, hardware, infraestructura, modelo desarrollo basado en la ingeniería y diseño de la arquitectura, la arquitectura orientada a servicios, SAP, Oracle, etc. (13).

- **Construx Estimate:** Estimación Construx es una herramienta que ayuda a mejorar sus capacidades de estimación de software. Aprovecha una mezcla de modelos de estimación (Modelo Putnam y COCOMO) de probada eficacia para predecir esfuerzo, presupuesto y cronograma para el proyecto basándose en estimaciones de tamaño. Estimación viene calibrado con datos de la industria, pero es más potente cuando se calibra con datos de la organización (14).

La principal desventaja que presentan las herramientas analizadas radica en que todas son software propietario. La solución informática que debe resultar de este trabajo está desarrollada sobre software libre y apoyado en técnicas de inteligencia artificial, con el objetivo de que el sistema realice un aprendizaje supervisado sin la presencia de un experto utilizando experiencia acumulada, elemento que no está presente en los sistemas descritos, para una correcta toma de decisiones.

En Cuba:

En las bibliografías consultadas no aparecen datos específicos sobre qué herramientas o métodos de estimación emplean las empresas nacionales desarrolladoras de software en sus proyectos. No todos optan por seguir modelos o procesos con resultados demostrados y se lanzan empíricamente a estimar y desarrollar los proyectos concentrándose preferentemente en la codificación o etapa de implementación.

La UCI tiene como una de sus misiones la producción de software y servicios informáticos a partir de la vinculación estudio-trabajo como modelo de formación. En los inicios la producción de soluciones informáticas desde el año 2002, en la Universidad se utilizó para planificar un método de estimación, ideado por un grupo de especialistas de la Infraestructura Productiva (IP) con cierto grado de experiencia en la gestión y dirección de proyectos. Este método de estimación (materializado en la herramienta privativa Microsoft Excel), se elaboró sobre una base empírica e intuitiva, pues como la UCI llevaba pocos años de creada, no contaba con registros históricos de datos de otros proyectos similares ya finalizados, de manera que las estimaciones se pudieran efectuar utilizando dichos datos como referencia (2).

Debido a la poca efectividad de este método y después de un análisis crítico del mismo se elaboró un nuevo método de estimación UCI dada la necesidad de estimar el tamaño, costo y esfuerzo requerido para desarrollar un producto de software en la institución más preciso, donde los resultados estimados se acerquen más al tiempo real de culminación de proyectos. Se construyó una herramienta del método de estimación UCI, la misma es Web y consta de 4 módulos, los módulos Proyectos y Estimaciones que son los manejados por los usuarios y los módulos Seguridad y Administración controlados por el administrador que en un tiempo estuvieron incorporados al paquete de Gestión de Proyectos GESPRO.

Esta herramienta según las entrevistas realizadas a líderes de proyectos y a las personas encargadas de realizar la planificación de los mismos, no es usada en muchos proyectos de la universidad. Algunos criterios coinciden también en que no es necesaria para el desarrollo de sus soluciones. La misma no cuenta con técnicas de inteligencia artificial que pueda facilitar y apoyar la toma de decisiones.

1.2 Inteligencia Artificial (IA).

A pesar de los esfuerzos realizados es difícil definir la IA debido a que no existe una única definición para inteligencia ni para artificial, sin embargo de manera general, la mayor parte de la bibliografía consultada coincide en que es la disciplina que busca que las máquinas realicen tareas que requerirían de inteligencia si fueran realizadas por humanos.

Según (15) la IA se divide en dos escuelas de pensamientos:

Inteligencia Artificial Convencional (simbólica-deductiva): busca traducir el problema a resolver en símbolos, los cuales son manipulados por un computador utilizando reglas. Está basada en análisis formal y estadístico del comportamiento humano ante diferentes problemas:

- Razonamiento basado en casos: Ayuda a tomar decisiones mientras se resuelven ciertos problemas concretos.
- Sistemas Expertos: Infieren una solución a través del conocimiento previo del contexto en que se aplica y de ciertas reglas o relaciones.
- Redes Bayesianas: Propone soluciones mediante inferencia estadística.
- Inteligencia Artificial basada en Comportamientos: Sistemas Complejos que tienen autonomía y pueden auto-regularse y controlarse para mejorar.

Inteligencia Artificial Computacional (simbólica-deductiva): implica desarrollo o aprendizaje interactivo (por ejemplo: modificaciones interactivas de los parámetros en sistemas conexionistas). El aprendizaje se realiza basándose en datos empíricos.

En el presente trabajo de investigación se dará una propuesta de solución basada en la técnica de Razonamiento Basado en Caso (RBC) típico de los Sistemas Basados en Conocimiento (SBC) pertenecientes a la Inteligencia Artificial Convencional.

1.2.1 Sistemas Basados en Conocimiento.

En la década del 70 se reconoció que los métodos de solución de problemas generales eran insuficientes para resolver los problemas orientados a aplicaciones. Se determinó que era necesario conocimiento específico sobre el problema, limitado a los dominios de aplicación de interés, en lugar de conocimiento general aplicable a muchos dominios. Este reconocimiento condujo al desarrollo de Sistemas Basados en el Conocimiento (SBC) (16). Los SBC son programas de computadora que mediante mecanismos de razonamiento permiten resolver problemas utilizando conocimiento especializado en un campo específico de aplicación. La solución obtenida en estos sistemas es similar a la de un experto en el dominio del problema.

Tres conceptos fundamentales relativos a los SBC lo distinguen de los programas basados en búsqueda general:

- La separación del conocimiento de cómo este es usado (distinción entre conocimiento y estrategia de control).
- El uso de conocimiento muy específico del dominio.
- Naturaleza heurística, en lugar de algorítmica, del conocimiento empleado.

Otros rasgos que lo distinguen de la programación convencional son que no requieren analizar completitud y pueden dar diversas soluciones.

- **Ventajas y Desventajas de los SBC.**

Estos sistemas según sus características y forma de aplicación presentan una serie de ventajas y desventajas que se encuentran reflejadas en (16) y (17). A continuación se detallan cada una de ellas.

Ventajas de los SBC

- Mayor disponibilidad: la experiencia está disponible para cualquier hardware de cómputo adecuado.
- Fácil modificación: estos sistemas son fáciles de modificar gracias a que presentan el conocimiento explícito y accesible.
- Gran accesibilidad: trabajan a tiempo completo en cualquier sistema de cómputo.
- Permanencia: la información se encuentra permanente en una base de conocimiento, a diferencia de lo que ocurre con la experiencia humana.
- Permiten evaluar el efecto de nuevas estrategias añadiendo o modificando conocimiento.
- Explicación de la solución: el sistema experto puede explicar clara y detalladamente el razonamiento que conduce a una conclusión.
- Preservación y amplia distribución de la experticidad escasa, estos sistemas constituyen una memoria institucional y poseen la capacidad para adquirir nuevo conocimiento y perfeccionar el que poseen.
- Experiencia múltiple, el conocimiento de varios especialistas puede estar disponible para trabajar simultánea y continuamente en un problema, además que el nivel de experiencia combinada de muchos sistemas expertos puede exceder el de un solo especialista humano.
- Respuestas rápidas y consistentes, algunas situaciones de emergencia pueden exigir respuestas más rápidas que las de un humano. Los expertos humanos pueden diferir en sus explicaciones, incluso un mismo experto puede responder de forma diferente en momentos diferentes. El sistema experto ofrece respuestas sólidas, completas y sin emociones en todo momento.

Desventajas de los SBC

- Las respuestas no siempre son correctas.
- Conocimiento limitado al dominio de experticidad.

- Ausencia de sentido común.
- No reconocen el límite de su conocimiento

Arquitectura de los SBC.

Los Sistemas Basados en el Conocimiento son un modelo computacional de más alto nivel que el paradigma de la programación convencional en el cual los sistemas están formados por tres componentes (ver figura 2): la base de conocimiento (BC), la máquina de inferencia (MI) y la interfaz (18):

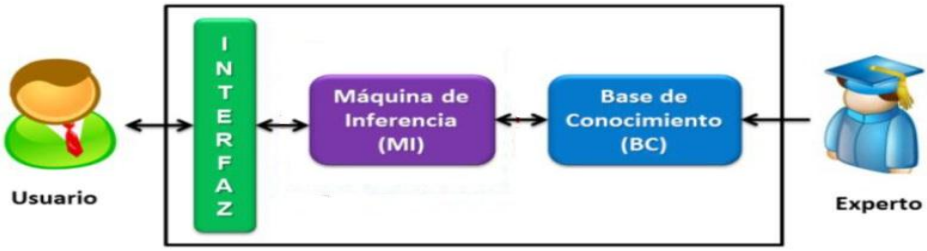

Figura 2: Arquitectura del SBC (20).

En la BC se almacena el conocimiento necesario para resolver los problemas del dominio de aplicación para el cual se desarrolla el SBC. El conocimiento que se almacena en la BC puede ser de diferente tipo: conocimiento simbólico sobre cómo resolver los problemas del dominio el cual se puede representar mediante diversas formas de representación del conocimiento (reglas de producción, frames, redes semánticas, etc.), probabilidades o frecuencias que modelan como se relacionan los valores de los diferentes rasgos que caracterizan el dominio, pesos de una red neuronal, casos o ejemplos de problemas del dominio, etc.

La máquina de inferencia es un procedimiento en el cual está implementado algún método que utiliza el conocimiento de la BC para resolver los problemas del dominio. El tipo de conocimiento determina que método de solución de problemas (MSP) es posible utilizar

Tipos de Sistemas Basados en Conocimiento.

Diferentes tipos de conocimiento dan lugar a diferentes tipos de sistemas basados en el conocimiento, entre ellos los Sistemas Basados en Reglas, los Sistemas Basados en Frames, los Sistemas Basados en Probabilidades, Sistemas Expertos Conexionistas o Redes Expertas, los Sistemas Basados en Modelos y los Sistemas Basados en Casos.

- **Sistema Basado en Reglas (SBR).**

Los Sistemas Basados en Reglas (SBR) son los más conocidos en los SBC, su forma de representación del conocimiento es mediante reglas de producción, como método de solución de problemas utilizan encadenamiento hacia delante (forward chaining o modus ponens) y encadenamiento hacia atrás (backwardchaining o modus tollens) y su fuente de conocimiento es a través de expertos, ejemplos y publicaciones (16).

- **Sistemas Basados en Frames (SBF).**

En este tipo de sistema la forma de representación es a través de frames. El método de solución de problemas empleados es la herencia y procedimientos adjuntos y como fuente de conocimiento lo constituyen los expertos, publicaciones y ejemplos.

- **Sistemas Basados en Probabilidades (SBP).**

Los sistemas probabilísticos se consideran otro de los tipos más importantes de SBC. Su forma de representación del conocimiento en la BC es mediante probabilidades o frecuencias, como MSP el teorema de Bayes y su fuente de conocimiento son los ejemplos. El centro de estos sistemas lo constituye su espacio probabilístico, llamado también conocimiento abstracto o de aplicación general.

- **Sistemas expertos conexionistas o redes expertas (RNA).**

El Sistema experto conexionista es un modelo computacional mixto su forma de representación del conocimiento es mediante pesos. Como método de solución de problemas tiene el modelo de neuronas cálculo de activaciones o modelo simbólico y su fuente de conocimiento son los ejemplos.

- **Sistemas Basados en Modelos (SBM).**

En estos sistemas la forma de representación del conocimiento es por el modelo del artefacto. El método de solución de problemas es razonamiento basado en modelos y su fuente de conocimiento son los esquemas estructurales y funcionales del artefacto.

- **Sistemas de Razonamiento Basados en Casos (SRBC).**

El SRBC al igual que el SBR son los más utilizados como SBC, su forma de representación del conocimiento es mediante casos. El mismo denota un método de solución llamado razonamiento basado en caso (RBC) donde su objetivo fundamental es la recuperación (búsqueda por semejanza) y adaptación de la solución y su fuente de conocimiento lo constituyen los ejemplos. Los sistemas que emplean el RBC usan una memoria permanente en lugar de alguna forma de base de conocimientos en la cual se almacene de forma explícita el conocimiento sobre el dominio de aplicación en forma de estructuras conceptuales, reglas de producción u otra forma de representación del conocimiento.

Los SRBC presentan algunas limitantes en la forma de definir la función de semejanza y en lo difícil que resulta encontrar una estructura apropiada para describir el contenido de un caso y decidir cómo la memoria de casos debe ser organizada e indexada para un almacenamiento, recuperación y rehúso efectivo (4).

El tipo de SBC a emplear en el presente trabajo es SRBC debido fundamentalmente a la forma de adquisición del conocimiento que presenta, pues razonan desde episodios específicos, lo cual evita el problema de descomponer el conocimiento del dominio y generalizarlo en reglas. La dificultad de la elaboración de un SBR está precisamente en la cantidad de reglas de producción a emplear según el dominio o características del problema a resolver. Estos sistemas, no logran reconocer qué reglas aplicar en cada ciclo, perdiéndose así la perspectiva global del conjunto de reglas al tener que ser analizadas una a una. Además hay dominios en que las entradas varían mucho y requerirán miles de reglas para considerar todas las situaciones.

Los SRBC presentan flexibilidad para representar el conocimiento a través de los casos, la organización de la BC y de las estrategias de recuperación y adaptación de los casos y que el usuario puede ser capaz de agregar nuevos casos a la BC sin la intervención experta (4). Mientras que los SBP no son factibles para todo tipo de dominio, pues se dificulta construir las redes con ayuda de expertos humanos cuando existen carencias de conocimiento.

Además los SRBC reúsan las soluciones previas al resolver un problema, y el almacenar casos que resultó un fracaso, lo que permite advertir sobre problemas potenciales a evitar. Así como también poder fundamentar las soluciones derivadas a partir de casos reales. Mientras que los sistemas expertos

conexionistas se dificultan en necesitar una gran cantidad de ejemplos y no poder explicar cómo alcanzan los resultados.

Por otro lado los SBM no manipulan con facilidad incertidumbre, ni manejan conocimiento heurístico lo cual puede hacerlos inaplicables en algunos dominios y son aplicables donde se pueda construir un modelo. Los SBF no se aplican mecanismos de aprendizaje automático por lo que no aumentan su conocimiento.

1.2.2 Sistema de Razonamiento Basado en Caso (SRBC).

Razonamiento Basado en Casos significa razonar sobre la base de experiencias o "casos" previos. El RBC es una alterativa entre otras metodologías para construir sistemas basados en el conocimiento. Al razonar basado en casos el solucionador de problemas recuerda situaciones previas similares a la actual y las usa para ayudar a resolver el nuevo problema (4).

- **Arquitectura de los SRBC.**

Un Sistema Basado en Casos tiene tres componentes fundamentales: la Base de Caso que contiene las descripciones de los problemas resueltos o no previamente. Cada caso puede describir un episodio particular o una generalización de un conjunto de episodios relacionados; el Módulo de Recuperación, recupera un caso semejante al nuevo y la solución del problema recuperado se propone como solución potencial del nuevo problema y el Módulo de Adaptación que adecua la vieja solución a la nueva situación.

- **Ciclo de vida de un SRBC.**

Los sistemas basados en casos definen una serie de pasos y componentes que interactúan en un ciclo de razonamiento (Ver figura 3). Inicialmente se presenta en el sistema como entrada la descripción de un problema convirtiéndose en un nuevo caso a resolver y se extrae de la base de caso por medio de un proceso de recuperación el o los casos más semejantes a dicho problema actual, luego pasa por un proceso de adaptación donde se adecua la solución del caso recuperado a la nueva situación obteniéndose una posible solución y por un proceso de evolución para dar un argumento lógico a la solución propuesta y evaluar el caso para saber si es factible su almacenamiento para la retroalimentación del sistema verificando el éxito de la solución. Posteriormente pasa por una fase de almacenamiento donde la experiencia útil se guarda para una futura reutilización y caso base se actualiza como un nuevo caso aprendido, o se modifican algunos casos existentes, evitando así, tener datos redundantes en la BC.

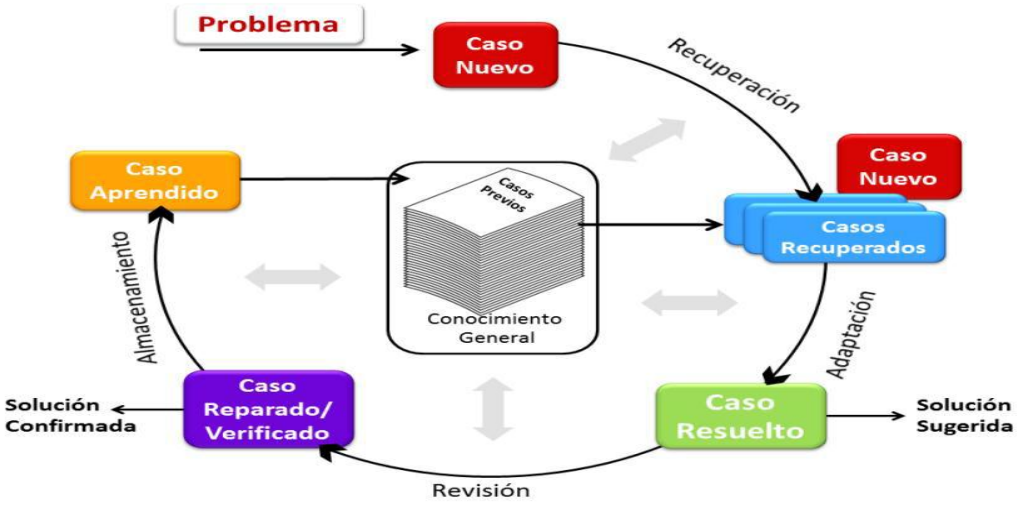

Figura 3: Ciclo de vida de un SRBC (20).

1.3 Herramientas y metodologías.

Dada la necesidad de buscar técnicas mediante las cuales se logren estandarizar el trabajo de las aplicaciones que se desarrollan, es necesario aplicar un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de dichos productos de software para así lograr el resultado informático deseado. Además se hace necesario como un plan de contingencias, en el cual se vaya indicando paso a paso todas las actividades a realizar, indicando además quienes deben participar en el desarrollo de las actividades y qué papel deben de tener y donde también se detalle la información que se debe producir. Es por esto que se hace necesario el uso de herramientas y metodologías para apoyar el desarrollo del software.

1.3.1 Metodología de desarrollo.

Una metodología de desarrollo es una colección de documentación formal referente a los procesos, políticas y procedimientos que intervienen en el proceso de desarrollo del software. La finalidad de una metodología de desarrollo es garantizar la eficacia y la eficiencia en el proceso de generación de software (19).

Para desarrollar la propuesta que presenta este trabajo el colectivo de autores decidió utilizar como metodologías de desarrollo FeatureDrivenDevelopment (FDD) debido a que se enfoca más a funcionalidades y disminuye el riesgo de los proyectos, pues gracias a sus entregas tangibles y al constante monitoreo de su calidad, se asegura el firme avance del mismo dejando satisfechos a los

desarrolladores y clientes sin afectar el proyecto. Además se emplea la metodología RUP (Proceso Unificado de Desarrollo) como apoyo para la generación de la lista de requisitos que constituye la entrada a FDD. RUP captura varias de las mejores prácticas en el avance moderno del software, en una forma que es aplicable para un amplio rango de proyectos y organizaciones.

- **Desarrollo Basado en Funcionalidades / FeatureDrivenDevelopment (FDD).**

FDD es un enfoque ágil para el desarrollo de sistemas (20). Se basa en iteraciones cortas que entregan funcionalidad tangible, hace énfasis en aspectos de calidad durante todo el proceso e incluye un monitoreo permanente del avance del proyecto. FDD también define métricas para seguir el proceso de desarrollo de la aplicación, útiles para el cliente y la dirección de la empresa, y que pueden ayudar, además de para conocer el estado actual del desarrollo, a realizar mejores estimaciones en proyectos futuros. El proceso consiste de cinco pasos secuenciales durante los cuales se diseña y se construye el sistema (Ver figura 4):

Figura 4: Procesos de FDD.

1.3.2 Lenguaje de modelado.

- **Lenguaje Unificado de Modelado (UML).**

Es el más conocido y utilizado en la actualidad en los sistemas de software. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. Se utiliza además para describir métodos o procesos y para definir un sistema y detallar los artefactos en el mismo (19).Cabe aclarar que aunque UML es orientado a objetos preferentemente, es útil en cualquier modelo tecnológico ya que es independiente de lenguajes de programación o tecnología determinada.

Para comprender mejor el sistema que se está construyendo es necesario disponer de un lenguaje de modelado que pueda adaptarse a las necesidades cambiantes del problema en cuestión. Es por ello que se utiliza UML, ya que proporciona plantillas que guían la construcción del sistema. Se utilizará la versión 5.0.

- **Visual Paradigm.**

Visual Paradigm es una herramienta visual de ingeniería de software para el modelado que es soportada por cualquier sistema operativo. Puede ser utilizada por una gran variedad de usuarios como analistas de sistemas, analistas de negocios e ingenieros de software. Provee soporte para la generación de código y la ingeniería inversa. Se integra con algunas herramientas como: Eclipse, Netbeans, Jbuilder, Oracle, entre otras (21).Se empleará Visual Paradigm for UML 8.0 Enterprise Edition v5.0.

1.3.3 Lenguaje de programación Groovy.

Groovy es un lenguaje nacido con la tarea de acoplarse de forma efectiva con Java, trabajar en conjunto con la Máquina Virtual de Java (JVM) y soportar los tipos de datos estándar, pero añadiendo características dinámicas y sintácticas presentes en otros lenguajes como Python, Smalltalk o Ruby (22). El código fuente Groovy se compila a bytecodes, y es posible instanciar objetos Java desde Groovy y viceversa. Este lenguaje aporta una sintaxis que aumenta considerablemente la productividad y un entorno de ejecución que nos permite manejar los objetos de formas que en Java serían extremadamente complicados. Todo lo anterior unido a que la mayor parte de código escrito en Java es totalmente válido en Groovy hace que este lenguaje sea de muy fácil adopción. Se empleará la versión Groovy 2.0.

1.3.4 Marco de Trabajo o Frameworks.

Los frameworks son arquitecturas genéricas integradas por un extensible conjunto de componentes. Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

- **Grails**

Grails es un framework para aplicaciones web libre desarrollado en el que se usa principalmente el lenguaje de Java y Groovy. Pretende ser un marco de trabajo altamente productivo siguiendo paradigmas tales como convención sobre configuración o no te repitas (DRY). Es un framework que funciona bajo el Modelo Vista Controlador, en el que principalmente lo que se hace es crear Controladores que son como servicios que manipulan la aplicación web, y todo el código de estos controladores es ejecutado en el servidor web (23). Se utilizará la versión Grails 2.0.1.

Entre los objetivos de Grails está:

- Hacer el desarrollo Web lo más simple y atractivo posible de tal forma que un amplio segmento de desarrolladores puedan utilizarlo.
- Puede usarse como un entorno de desarrollo autónomo que esconde todos los detalles de configuración.
- Integrarse estrechamente con la plataforma Java.
- Ser sencillo en la superficie, pero mantener la flexibilidad para acceder a los potentes frameworks Java sobre los que se construye.

JQuery

JQuery, es una biblioteca o framework de JavaScript rápida y concisa que simplifica el tratamiento de documentos HTML, es un software libre y de código abierto que ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código. La característica principal de la biblioteca es que permite cambiar el contenido de una página web sin necesidad de recargarla, mediante la manipulación del árbol DOM y peticiones AJAX (20).Se empleará la versión JQuery 1.8.

Encog

Encog es un framework para Java, .Net y C / C + +, desarrollada por Jeff Heaton; permite emplear tanto redes de neuronas artificiales como arañas (bots) HTTP. Es compatible con una amplia variedad de redes neuronales y arquitecturas. Contiene las clases para crear gran diversidad de redes, así como clases de apoyo para normalizar y procesar los datos para estas redes neuronales. Tiene soporte para redes tipo Feedforward, Hopfield y Mapas Auto-organizativos de Kohonen. El entrenamiento puede realizarse mediante retropropagación, algoritmos genéticos o simulated annealing (24).

La utilización de este framework es para la creación de la red neuronal que se empleará en el mecanismo de aprendizaje del sistema. El entrenamiento se realiza mediante retropropagación del error. Se podrá evaluar una vez entrenada la red si un caso puede ser aprendido y almacenado en la BC.

1.3.5 Sistema Gestor de Base de Datos (SGBD).

Los Sistemas de Gestión de Base de Datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan (25). El propósito general de los sistemas de gestión de base de datos es manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante.

- **PostgreSQL**

El Sistema Gestor de Bases de Datos Relacionales Orientadas a Objetos conocido como PostgreSQL fue desarrollado originalmente desde 1977 en el Departamento de Ciencias de la Computación de la Universidad de California en Berkeley. PostgreSQL es el gestor de bases de datos de código abierto más avanzado hoy en día, ofreciendo control de concurrencia que no es más que una estrategia de almacenamiento que permite trabajar con grandes volúmenes de datos, soportando casi toda la sintaxis SQL (incluyendo sub consultas, transacciones, y tipos y funciones definidas por el usuario), contando también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java, Perl, Tcl y Python) (26).

Para la realización de este trabajo es necesaria la utilización del gestor de base de datos antes mencionado en su versión 8.4.

Conclusiones Parciales.

En el capítulo se elaboró el marco teórico de la investigación identificando las principales tendencias, limitaciones y ventajas de los sistemas de software para estimar la duración de proyectos de desarrollo de software. Se estudiaron algunos de los métodos reconocidos internacionalmente para la estimación de proyectos de software identificando sus ventajas y desventajas utilizándolas como punto de partida en la investigación para la elaboración de la propuesta del sistema. Tras el análisis de las herramientas para estimar proyectos existentes en el mundo, en la región y en el país en particular se concluyó que muchas son privativas y que no están apoyadas en técnicas de inteligencia artificial que pueda a los planificadores y líderes de proyecto en la toma de decisiones. Se analizaron los diferentes SBC profundizando en los sistemas de RBC como técnicas que ayudan en la resolución de problemas de estimación en proyectos de software. También se definió como metodología a utilizar FDD, como frameworks Grails 2.0.1 y JQuery 1.8, como lenguaje de programación Groovy 2.0, como lenguaje de modelado UML 5.0 y Visual Paradigm 8.0 y como SGBD Postgre 8.4.

CAPITULO 2. ANÁLISIS Y DEFINICIÓN DEL SISTEMA.

Introducción del capítulo.

En el presente capítulo se hace un análisis profundo del funcionamiento de los sistemas de razonamiento basado en casos para formalizar un modelo computacional en la estimación de la duración de un proyecto de desarrollo de software como entidad principal. Se revisa y analiza una BC y un modelo de motor de inferencia propuesto para estimar la duración de un proyecto de software en la Universidad de las Ciencias Informáticas. Además se hace un análisis de diferentes técnicas de IA que permitan desarrollar mecanismos de aprendizaje automático. La integración de estos elementos permitirá hacer una nueva propuesta para estimar el tiempo de duración de un proyecto de software de gestión apoyada en RBC.

2.1 Definición de la Base de Caso.

La base de casos contiene las experiencias, ejemplos o casos a partir de los cuales el sistema hace sus inferencias. Esta base puede ser generada por entrevistas a expertos humanos o por un procedimiento automático o semiautomático que construye los casos desde datos existentes registrados, por ejemplo, en una base de datos (4).

2.1.1 Estructura y representación de los casos.

“Un caso es una experiencia previa que ha sido capturada y aprendida para poder ser reutilizada con el fin de resolver un futuro problema. La información que se almacena sobre una experiencia depende tanto del dominio como del propósito para el que el caso se usa”. En este tipo de sistema de RBC para estimar la duración de proyectos de software, “los casos de la BC representan ejemplos de estimaciones de duración de proyectos de desarrollo de software de gestión”. Están formados por los rasgos predictores (representan las características o indicadores fundamentales de estos proyectos) y el rasgo objetivo (el resultado de estimar el tiempo de duración de los mismos).

Selección de los rasgos.

La selección de rasgos es cuestión central tanto en la definición del modelo de la base de conocimiento como en el modelo del método de solución de problemas. Potencialmente, en el conjunto de rasgos podrían estar todas las propiedades que describen los objetos; pero existen rasgos que carecen de importancia de acuerdo al dominio de aplicación (27).

El conjunto de rasgos determina qué información será almacenada en memoria para cada uno de sus elementos, la cual debe permitir la posterior recuperación de los casos semejantes dada la descripción de un nuevo problema. Otro aspecto es que dentro del conjunto de rasgos que se seleccionan no todos tienen la misma importancia y esta diferencia debe tenerse en cuenta para comparar objetos.

- **Rasgos Predictores.**

Los rasgos predictores están compuestos por cuatro factores fundamentales: nombre, valor, relevancia e incertidumbre. En este trabajo para la selección de los rasgos predictores se analizaron los indicadores de los métodos de estimación reconocidos internacionalmente como COCOMO II, Puntos por Casos de Usos y Puntos de Función y también por el método de estimación UCI que es empleado en algunos proyectos de la universidad. Se analizó un modelo de BC propuesta en años anteriores (6), donde se utilizó una descripción por categoría para obtener los indicadores que influyen sobre un proyecto y unir los distintos métodos con los indicadores que se repiten o agrupar los que se refieren a un mismo tema. Esta BC consta de 113 rasgos, representando un número elevado para el trabajo a la hora de estimar y ocuparía demasiado tiempo al planificador en realizar esa actividad. Después de un estudio y como alternativa al problema de la selección de rasgos se realizó una reducción de estos por criterio de experto o método Delphi (Ver Anexo 7).

Algunos de los rasgos obtenidos por criterio de expertos pudieron ser fusionados unos de otros según bibliografías consultadas, debido a que resultaban redundantes (Ver Anexo 3).

- **Rasgo Objetivo.**

El dominio del rasgo objetivo es el resultado de la estimación a partir de los rasgos predictores. Este resultado se da en unidades temporales de tiempo, o sea, días, meses, años, etc.

- **Relevancia del Rasgo.**

La importancia de los rasgos es un elemento que debe tenerse en cuenta a la hora de recuperar los casos más semejantes. Todos los rasgos no tienen la misma relevancia para estimar la duración del desarrollo de software. En el trabajo investigativo anterior (6) se definió la relevancia por la factibilidad de los indicadores mediante el criterio de experto, en el presente por el problema de Selección de Variables en el Reconocimiento de Patrones, se aplicó el cálculo de la Teoría de Testores, para dar un fundamento matemático que dé mayor veracidad a los resultados. Ver Anexo 3.

En la teoría de testores, un testor es definido como un conjunto de rasgos que discrimina objetos de diferentes clases. Un testor se llama irreducible (típico) si ninguno de sus subconjuntos propios es un testor. Cuando se refiere a testores típicos, se restringe solo a los testores típicos de Zhuravlev's, donde las clases son conjuntos disjuntos, el criterio de comparación entre rasgos es booleano y el criterio de semejanza entre objetos asume que dos objetos son diferentes si al menos uno de sus rasgos también lo son. Los testores típicos han sido ampliamente utilizados para evaluar la importancia informacional de los rasgos y como conjunto de apoyo en los algoritmos de clasificación (27).

Las relevancias que se obtuvieron de los rasgos por la Teoría de Testores se reflejan en el Anexo 3.

- **Incertidumbre del Rasgo.**

La incertidumbre involucrada en cualquier situación de solución de problemas es un resultado de alguna información deficiente. La estimación de software conlleva a un riesgo inherente y es este riesgo es el que lleva a la incertidumbre (3).Para estimar la duración de un proyecto de software de gestión la información que se presenta como características del mismo puede ser imperfecta. Resulta necesario definir con qué grado de certeza se obtienen los valores de los indicadores que definen a un proyecto de software para estimar el tiempo y así construir un modelo que pueda tratar toda imprecisión de la información presente.

Las incertidumbres de los rasgos del nuevo problema a resolver serán definidas por el usuario debido a que en cada proyecto estos no tienen la misma magnitud. En caso que no sea conocida por el usuario ya que muchas veces estos pudieran no ser expertos, será el resultado del promedio de todas las incertidumbres de ese rasgo y se calculará utilizando la ecuación:

Ecuación 2:

$\vartheta(O_i) = \frac{\sum_{j=1}^{n} \vartheta(O_j)}{n}$, Donde:

$\vartheta(O_i)$: Incertidumbre no especificada para el rasgo i del nuevo caso O.

$\vartheta(O_j)$: Incertidumbre del rasgo i para el caso j de la BC

n: Cantidad de casos de la BC.

2.1.2 Representación del modelo de memoria.

La tarea de recuperación o selección de casos se facilita si la base presenta una organización apropiada. La organización deberá permitir: primero la recuperación de un subconjunto de casos que puedan ser aplicados al problema planteado, y luego aplicar medidas de similitud para seleccionar de entre el conjunto de casos, el que esté más próximo al problema planteado. Una vez elegida la representación de los casos, la elección del modelo de memoria es el siguiente paso. En la literatura consultada se hace referencia principalmente a dos estructuras de memoria, plana y jerárquica.

- **Memoria plana.**

Los casos se almacenan secuencialmente en una lista simple, un arreglo o un fichero. Este tipo de memoria presenta la ventaja de que añadir nuevos casos resulta muy rápido y fácil de implementar. No ocurre así con la recuperación de casos, ya que, aunque el procedimiento de recuperación de los casos más similares es sencillo, resulta muy lento cuando el número de casos en la base es alto (28).

- **Memoria jerárquica.**

En una estructura con memoria jerárquica se utilizan representaciones en forma de árbol, en los que cada nodo interior representa un atributo del caso y en las hojas se almacenan las soluciones a los mismos. Esta estructura sacrifica sencillez en la inserción de los nuevos casos por eficiencia en los algoritmos de búsqueda (6).

La recuperación en este tipo de organización de memoria jerárquica es mucho más eficiente que una memoria plana. Sin embargo, este procedimiento es computacionalmente más caro, ya que la base necesita un mayor espacio de almacenamiento y su actualización resulta más compleja pues requiere generalmente modificar una parte de la red. Por otro lado, el diseño y mantenimiento de la red óptima es costoso. Los casos recuperados dependen del contenido de los nodos, la jerarquía establecida debe responder a la importancia de las características; algunos casos significativos pueden ignorarse si la ordenación de estos nodos no es óptima.

Para seleccionar la estructura de la memoria que modelará la BC, se tuvo en cuenta características necesarias que definen un proyecto para una correcta estimación sin incluir estructuras que modelen entradas de datos incompletos. Se seleccionó una estructura de memoria plana debido a que no está expuesta a pérdida de información. Cada proyecto en la BC está estructurado por sus propias

características, de existir un cambio o pérdida de datos no se estaría representando el mismo proyecto y pudiera no durar el mismo tiempo, además que ocasionaría estimaciones erróneas en el futuro al referirse a información no existente o incierta. Aunque su tiempo de recuperación resulta más lento que la jerárquica el sistema no se verá afectado debido a que la BC no es grande y los casos que se almacenan son solamente ejemplo de estimaciones de proyectos de gestión.

2.2 Motor de Inferencia. Definición de los módulos de Recuperación y de Adaptación.

El objetivo de un SRBC es encontrar la solución de un problema determinado. El motor de inferencia trabaja con los casos representados en la BC para obtener una solución a un nuevo problema. Para llegar a esta solución el motor de inferencia cuenta con el módulo de recuperación y el módulo de adaptación. El módulo de recuperación cuenta con un algoritmo de recuperación que tiene como misión buscar y seleccionar en la memoria del SRBC los problemas más similares al problema presente. Los casos deben representar tanto los problemas como sus respectivas soluciones. Los casos seleccionados de la memoria son reutilizados para generar una solución inicial. La adaptación de casos se realiza normalmente en esta fase, donde se indexa una solución al problema creándose un nuevo caso que será almacenado en la memoria solo si es factible para utilizarlo en el futuro y que impida información redundante que haga ineficiente la BC.

2.2.1 Módulo de Recuperación.

La primera tarea en el ciclo de un SRBC es la recuperación de uno o más casos similares de la base de casos donde está almacenada la experiencia pasada en forma de casos. La tarea de recuperación comienza con una descripción del problema, y finaliza cuando se encuentra un caso o pequeño conjunto de casos anteriores, que mejor se ajustan. Para llevar a cabo la recuperación es necesario tener un algoritmo de recuperación y una medida de similitud que serán usados para obtener el conjunto de casos similares.

- **Algoritmos de Recuperación.**

La búsqueda en el RBC tiene dos aspectos principales:

- El acceso a un conjunto de elementos de memoria o casos que resulten promisorios para el problema a resolver.

- La recuperación de los elementos de memoria de este conjunto que sean más semejantes al problema.

Los algoritmos de recuperación más investigados son: K-vecinos más cercanos, y las técnicas inductivas entre las que se encuentran los árboles de decisión y sus derivados. Estas técnicas necesitan de una medida de similitud para determinar la proximidad entre casos tales como la distancia euclídea, de haming o de levenshtein (28).

En el trabajo investigativo de Karina Reyna Pupo (29) se explica detalladamente los algoritmos K-vecinos más cercanos y árboles de decisión. La investigadora propone como algoritmo de recuperación árboles de decisión, en el presente trabajo se escoge el K-vecino más cercano debido a que es menos sensible a pérdidas, mientras que las técnicas inductivas presentan como principal desventaja que si los datos de un caso se pierden o son desconocidos no se puede recuperar. La BC contará con casos que representan ejemplos de estimaciones de proyectos de software de gestión con datos reales y si estos se pierden o son desconocidos puede ocasionar una mala estimación a un nuevo proyecto que se desee estimar.

- **K-vecinos más cercanos.**

En este algoritmo el caso recuperado es aquel en el que la suma de los pesos de las características que ajustan con las del caso nuevo es mayor que la de otros casos que ajustan. Esto quiere decir, que si los pesos de las características son iguales, un caso que ajusta con n características será recuperado antes que otro caso que empareja con k características, siendo k<n. Se puede asignar un peso mayor a las características consideradas más importantes (28).

El vecino más cercano es una técnica muy simple que proporciona una medida de cuán similar es un caso objetivo a un caso de la base. Pero su principal desventaja es la velocidad de recuperación ya que para encontrar el caso que mejor ajusta, el caso objetivo debe compararse con cada caso de la base de casos. Esto quiere decir que una comparación de similitud (distancia) debe calcularse para cada característica indexada. Esto hace que el algoritmo sea ineficiente a la medida que aumenta la base de casos.

- **Similitud entre rasgos.**

Para la recuperación de los casos es necesario definir la función de comparación entre rasgos. Esta función (δ) permite una comparación entre los descriptores de los objetos y es definida de diferentes formas en dependencia de los tipos de valores que tengan los rasgos. Como se había mencionado un

caso cuenta con cuatro tipos de rasgos; nominal, binario, por intervalo y ordinal para los cuales se definen funciones distintas de comparación.

Similitud entre rasgos Ordinales.

Inicialmente Pupo en (29) define una función de normalización para valores numéricos que puedan tomar los rasgos en función de los diferentes estados ordenados para la clasificación. El conjunto quedaría conformado {1,…, M}, donde M, es el último valor ordenado. El rasgo que se analiza del nuevo caso, tendrá un valor perteneciente a este intervalo. Es muy conveniente llevar el valor de esas variables aun intervalo de [0.0; 1.0], realizando un proceso de normalización (ecuación 3), de modo que cada variable tenga el mismo peso. El valor normalizado se obtiene de dividir el valor numérico del rasgo menos uno entre el último valor ordenado del conjunto admisibles de valores del rasgo menos uno.

Función de normalización del valor del rasgo i del caso O.

Ecuación 3:

$$V\big(X_i(O)\big) = \frac{R\big(X_i(O)\big) - 1}{M\big(X_i(O)\big) - 1}$$

$X_i(O)$: Valor del rasgo i para el caso O.

$V\big(X_i(O)\big)$: Valor normalizado para el rasgo i del caso O.

$R\big(X_i(O)\big)$: Valor numérico para el rasgo i del caso O.

$M\big(X_i(O)\big)$: Último valor del orden perteneciente al intervalo numérico del rasgo ordinal i para el caso O.

Pupo en (29) para el cálculo de similitud de rasgos ordinales define la función de distancia absoluta o Manhattan, esta brinda la distancia que hay entre dos valores, pero para la similitud entre ellos se calcula como 1 menos la distancia y esta no es más que el módulo de la diferencia de los valores de los rasgos normalizados.

Función de comparación para rasgos ordinales con valores normalizados entre 0 y 1.

Ecuación 4:

$$\delta'\big(X_i(O), X_i(O_t)\big) = 1 - \left|V\big(X_i(O)\big) - V\big(X_i(O_t)\big)\right|$$

$X_i(O)$: Valor del rasgo i para el caso O.

$X_i(O_t)$: Valor del rasgo i para el caso O_t

$V(X_i(O))$: Valor normalizado para el rasgo i del caso O.

$V(X_i(O_t))$: Valor normalizado para el rasgo i del caso O.

El resultado de esta ecuación es un valor entre 0 y 1, pero en el presente trabajo investigativo se trata con valores de 0 y 1 por la necesidad de emplearlos para el cálculo de la MD y la MB en la Teoría de Testores. Se define finalmente la función de similitud para rasgos ordinales con valor 1 si el resultado de (ecuación 4) es mayor o igual que el error que se calcula en (ecuación 6) y 0 en otro caso.

Función de comparación para rasgos ordinales con valores 0 o 1.

Ecuación 5:

$$\delta(X_i(O), X_i(O_t)) = \begin{cases} 1 \ si \ \delta'(X_i(O), X_i(O_t)) \geq \varepsilon \\ 0 \quad e.o.c \end{cases}$$

$\delta'(X_i(O), X_i(O_t))$: Función de comparación para rasgos ordinales con valores normalizados entre 0 y 1 para los casos O y O_t.

ε: Error dado en dependencia de los valores admisibles que tenga cada dominio.

El error εdetermina el menor valor de semejanza que puede existir entre los rasgos y se define como uno menos la división entre uno y el mayor valor del conjunto admisible de valores del rasgo predictor más 1:

Ecuación 6:

$$1 - \frac{1}{M(X_i(O)) + 1}$$

Similitud entre rasgos nominales, binarios y por intervalo.

En la similitud para los rasgos nominales, binarios y por intervalo se emplea una medida absoluta donde es 1 si son exactamente iguales y 0 en otro caso. Esta función fue tomada de (29).

Función de comparación para los rasgos nominales y binarios:

Ecuación 7:

$$\delta(X_i(O), X_i(O_t)) = \begin{cases} 1 \ si \ X_i(O) = X_i(O_t) \\ 0 \ e.o.c \end{cases}$$

$X_i(O)$: Valor del rasgo i para el caso O.
$X_i(O_t)$: Valor del rasgo i para el caso O_t.

- **Función de semejanza.**

La función de semejanza determina una medida numérica del grado de similaridad de cada ítem de la memoria con respecto al problema a resolver; esta función considera las diferencias y semejanzas entre la descripción de ambos problemas (la del problema resuelto y la del nuevo problema) (16). No existe una medida de semejanza única, general, para cualquier dominio, de ahí que la eficiencia del sistema radica en la función de semejanza que se defina (4).

La función de semejanza da como resultado un valor entre 0 y 1 que expresa cuán semejante es el problema a cada uno de los casos contenidos en la BC. Para su trabajo la función de semejanza utiliza una función de comparación por rasgos, esta función compara el valor de un rasgo del problema con el valor de ese rasgo en un caso y obtiene un valor (generalmente entre 0 y 1) que expresa el grado de similaridad entre ambos valores.

Pupo en (29)propone una función de semejanza que tiene en cuenta la incertidumbre de los rasgos. Esta función mantiene el sentido de la función de semejanza y es totalmente lógica su afectación en la comparación de los rasgos debido a que si la varianza de esta incertidumbre tiende a 0 habrá mayor afectación en la semejanza y una menor confiabilidad de los valores y a medida que tienda a 1 habrá una menor afectación de la función de semejanza. En la presente investigación se empleará la función definida en su trabajo, dónde el resultado de la semejanza se obtiene por la división de la sumatoria de la relevancia de cada uno de los rasgos multiplicada con el valor de similitud entre los rasgos predictores del nuevo problema y el caso recuperado y con la variancia de la incertidumbre entre la sumatoria de la relevancia de los rasgos:

Ecuación 8:

$$\beta(O, O_t) = \frac{\sum_{i=1}^{n} p_i * \delta\left(X_i(O), X_i(O_t)\right) * (1 - |\vartheta_i(O) - \vartheta_i(O_t)|)}{\sum_{i=1}^{n} p_i}$$

O: Nuevo problema a resolver.
O_t: Un caso de la BC.
n: Número de rasgos predictores.

p_i: Relevancia del rasgo i.

$\delta(X_i(O), X_i(O_t))$: Función de comparación entre los rasgos i de los casos O y O_t.

$\vartheta_i(O)$: Incertidumbre del rasgo i para el caso O.

$\vartheta_i(O_t)$: Incertidumbre del rasgo i para el caso O_t.

- **Umbral de Semejanza.**

El umbral es la cota inferior de los posibles valores de semejanza existentes para cada uno de los casos recuperados con respecto al nuevo caso. Su verdadera función consiste en restringir el intervalo en caso de que se quiera lograr una mayor precisión en la obtención de los casos más semejantes. Para calcular el umbral se construye una matriz de semejanza cuadrada donde las filas y las columnas están representadas por los casos de la base y la intersección es la semejanza que existe entre ellos. El resultado de esta función es la sumatoria de los elementos de esta matriz que representan la semejanza entre los casos de la BC que están de la diagonal hacia abajo multiplicado por 1 entre la cantidad de semejanzas sumadas.

El umbral de semejanza queda definido como:

Ecuación 9:

$$\mu = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \beta(O, O_t)$$

m: Número de casos

$j\ y\ i$: Recorrerán las filas y las columnas respectivamente.

$\beta(O, O_t)$: Semejanza entre el caso O y O_t

Con la definición del umbral y la semejanza de los casos se recuperan los casos más similares y se pasa al proceso de adaptación de las soluciones.

2.2.2 Módulo de Adaptación.

Después de la determinación de los casos más semejantes, las soluciones contenidas en dichos casos pueden usarse directamente como solución al nuevo problema, pero comúnmente necesitan ser modificadas (30). El proceso de adaptación consiste en la modificación y el combinar los casos recuperados para armar una nueva solución que satisfaga el caso actual.

La forma más simple que puede tomar la adaptación es la *adaptación nula o reinstanciación*. En ella la solución del caso más similar al problema planteado se toma sin cambio alguno. En este caso el criterio de optimalidad da, según el principio de máxima utilidad esperada, una buena justificación para tomar esa decisión (31).

- **Utilidad Esperada.**

Una vez seleccionados los casos más semejantes es posible que éstos propongan soluciones diferentes por lo que se hace necesario determinar cuál de ellas debe tomarse. Se propone un criterio de adaptación para determinar el caso óptimo entre los casos recuperados y a partir de él tomar la decisión adecuada. Este criterio de optimalidad se da a través de una función que combina de forma ponderada el valor de la semejanza entre el nuevo problema y el caso recuperado y el grado de certeza de la solución del caso recuperado. Desde el punto de vista de la teoría de la toma de decisiones, el resultado de esta función puede ser vista como la utilidad esperada del caso, la cual permite establecer sus preferencias sobre un caso para tomar una decisión (31).

Para el cálculo de esta función se tiene en cuenta el grado de certeza de la solución del caso recuperado (rasgo objetivo), es decir la duración de un proyecto de desarrollo de software. La incertidumbre del rasgo objetivo queda definida como la sumatoria de las incertidumbres de cada uno de sus rasgos predictores entre la cantidad de dichos rasgos:

Ecuación 10:

$$\vartheta(O_t) = \frac{\sum_{i=1}^{n} \vartheta_i}{n}$$

$\vartheta(O_t)$: Incertidumbre del rasgo objetivo para el caso O_t.
ϑ_i: Incertidumbre del rasgo predictor i para el caso O_t.
n : Número de rasgos predictores del casoO_t.

Una vez obtenida la certeza del rasgo objetivo se define la utilidad esperada como la suma de la semejanza del nuevo caso y el caso recuperado por un valor α y la multiplicación de la incertidumbre del caso recuperado por 1 menos el valor α:

Ecuación 11:

$\mu(O_t) = \alpha\beta(O, O_t) + (1-\alpha) * \vartheta(O_t)$

$\mu(O_t)$: Utilidad del caso O con respecto al caso recuperado.
$\beta(O, O_t)$: Valor de semejanza entre el caso nuevo O y el caso recuperado O_t.
$\vartheta(O_t)$: Incertidumbre del rasgo objetivo para el caso O_t.

A medida que α tiende a 1 significa que se le está dandomayor importancia a la semejanza que al grado de incertidumbre. Muchos autores lo consideran como$\alpha = 0.5$, en este trabajo se define como la cantidad de incertidumbres entradas por el usuario entre la cantidad de rasgos predictores:

Ecuación 12

$\alpha = \frac{n_\vartheta}{n}$

n_ϑ: Cantidad de incertidumbre entradas por el usuario para el nuevo caso.
n: Cantidad de rasgos predictores.

Obtenido el valor de la utilidad será adaptado al nuevo caso la solución del caso recuperado con mayor utilidad esperada. Se ha llegado a argumentar que la reutilización (adaptación) es el paso más importante del RBC (28), ya que incorpora inteligencia al proceso.

2.2.3 Mecanismo de Aprendizaje.

El mecanismo de aprendizaje es el sistema que va a ser capaz de adquirir nuevos conocimientos por medio de un proceso que le permita evaluar su propio aprendizaje. Integra diversas técnicas y algoritmos de aprendizaje que interactúan con el experto para obtener nuevos elementos de conocimiento, modificaciones al existente y sobre todo revisar y corregir la consistencia.

- **Aprendizaje Automático (AA).**

El Aprendizaje Automático trata de construir sistemas informáticos que optimicen un criterio de rendimiento utilizando datos o experiencia previa. El AA transforma los datos en conocimiento y proporciona sistemas de propósito general que se adaptan a las circunstancias (32).

En la presente investigación se desea que el sistema sea capaz de aprender nuevos casos y almacenarlos en la base si factible utilizarlos en futuras estimaciones. Un razonador que combine RBC con otros métodos apropiados de aprendizaje incrementará sustancialmente su poder para almacenar y acceder a sus casos en la memoria. Se hace necesario definir un buen mecanismo de aprendizaje que no permita incluir información repetida y ofrezca mejores soluciones.

- **Algoritmos de Aprendizaje Automático.**

Los algoritmos de aprendizaje automático más investigados son: aprendizaje basado en árboles de decisión , aprendizaje basado en redes neuronales artificiales, aprendizaje probabilístico y Bayesiano, aprendizaje basado en instancias, aprendizaje evolutivo, aprendizaje por refuerzo y aprendizaje por refuerzo.

En este trabajo se empleará redes neuronales artificiales debido a la flexibidad para la adaptación y el aprendizaje. Además por su característica de procesado no lineal que aumenta la capacidad de la red para clasificar patrones.

2.2.4 Redes Neuronales Artificiales (RNA).

Las Redes Neuronales no son más que un modelo artificial y simplificado del cerebro humano, que es el ejemplo más perfecto del que se dispone para un sistema que es capaz de adquirir conocimiento a través de la experiencia. Una red neuronal es "un nuevo sistema para el tratamiento de la información, cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano: la neurona" (33).

- **Topologías de Redes Neuronales.**

Topología de una red neuronal es la organización o arquitectura del conjunto de neuronas que la forman. Esta organización comprende la distribución espacial de las mismas y los enlaces entre ellas. La arquitectura de una RNA depende de cuatro parámetros principales: el número de capas del sistema, el número de neuronas por capa, el grado de conectividad entre las neuronas y el tipo de conexiones neuronales.

Las arquitecturas neuronales pueden clasificarse atendiendo a distintos criterios (34):

Según su estructura en capas:

- ✓ **Redes Monocapa:** están compuestas por una única capa de neuronas, entre las que se establecen conexiones laterales y, en ocasiones, autorrecurrentes.

- ✓ **Redes multicapas:** son aquellas que disponen de un conjunto de neuronas agrupadas en varios niveles o capas (de entrada, oculta (s), y de salida).

El tipo de red a utilizar en la propuesta de solución según su estructura en capas es una red multicapa debido fundamentalmente a que reciben un conjunto de entrada y devuelven una salida diferente a dicha entrada.

Según el flujo de datos en la red:

- ✓ **Redes unidireccionales o de propagación hacia adelante (feedforward):** en las que ninguna salida neuronal es entrada de unidades de la misma capa o de capa precedentes. La información circula en un único sentido, desde las neuronas de entrada hacia las neuronas de salida de la red.

Redes de propagación hacia atrás (feedback): en las que las salidas de las neuronas pueden servir de entradas a unidades del mismo nivel (conexiones laterales) o niveles previos. Las redes de propagación hacia atrás que presentan lazos cerrados se denominan sistemas recurrentes.

El sistema propuesto para estimar el tiempo de duración requiere que se produzca un resultado lo más real posible. En el presente trabajo según el flujo de datos se utilizará una red de propagación hacia atrás debido a la necesidad de entrenamiento de la red donde actualizará los pesos de las conexiones hasta que la red produzca una salida satisfactoria con el menor error posible aun sin antes haber visto la entrada en su fase de entrenamiento.

- **Aprendizaje.**

La principal propiedad de una red neuronal artificial es la capacidad de aprender del entorno en el que opera y mejorar su funcionamiento. El aprendizaje es el proceso mediante el cual los parámetros de una red neuronal artificial se adaptan como consecuencia de un proceso de estimulación llevado a cabo por el entorno en el que la red opera. El tipo de aprendizaje vendrá determinado por la forma en que cambie la configuración de la red (35).

- ✓ **Aprendizaje supervisado:** en este aprendizaje, hay un supervisor que controla el proceso de aprendizaje de la red, el mismo comprueba la salida de la red en respuesta a una determinada

entrada y en el caso de que la salida no coincida con la deseada, se procede a modificar los pesos de las conexiones, con el fin de conseguir que la salida obtenida, se aproxime a la deseada. Si la red utiliza un tipo de aprendizaje supervisado se le proporciona parejas de patrones entrada-salida y la red neuronal aprende a asociarlos.

- ✓ **El aprendizaje no supervisado:** con este aprendizaje, la red no requiere influencia de un supervisor, para ajustar los pesos de las conexiones entre las neuronas. La red no recibe ninguna información por parte del entorno que le indique si la salida generada en respuesta a una determinada entrada es o no correcta.

El aprendizaje a emplear en la propuesta de solución es el supervisado debido a la presencia de expertos, que se encargarán en el sistema de verificar que la salida de la red sea la esperada según la entrada. El conjunto de entrenamiento está en correspondencia con casos reales de proyectos de software de gestión ya culminados, por lo que la información es totalmente verídica y este supervisor podrá comprobar el resultado y saber si fue o no el esperado.

- **Tipologías de Redes Neuronales.**

Una Red Neuronal Artificial (RNA) puede considerarse como un grafo dirigido ponderado en el que neuronas artificiales son nodos y hojas dirigidas y ponderadas con conexiones entre salidas y entradas de neuronas (36). Entre los principales tipos de redes neuronales se encuentran:

Tabla 1: Redes Neuronales.

Tipo de RN	Topología	Capa de Neurona	Aprendizaje
Perceptrón Simple	Feedforward.	1 capa de entrada y 1 capa de salida.	Supervisado.
Perceptrón Multicapa	Feedforward.	1 capa de entrada y 1 o más capas ocultas y 1 capa de salida.	Supervisado
Adeline	Feedforward.	1 capa de entrada y 1 o más capas ocultas y 1 capa de salida.	Supervisado.
Backpropagation	Feedforward.	1 capa de entrada y 1 o más capas ocultas y 1 capa de salida.	Supervisado.
Hopfied	Feedback.	Matriz	No supervisado.
Mapas de Kohonen	Feedforward/feedback.	1 capa de entrada y 1 mapa.	No supervisado.

En el presente trabajo investigativo el tipo de red que se decidió usar es el Backpropagation. Este tipo de red es feedforward, multicapa y presenta aprendizaje supervisado. El Backpropagation es la tipología más usada por su capacidad de aprender patrones. Esta capacidad es fundamental para el sistema, dado que, a partir de datos reales de proyectos culminados que la red pueda aprender, podrá dar soluciones verídicas a nuevos problemas. Enfatizando así, su facilidad de dar salidas satisfactorias a entradas que el sistema no ha visto nunca en su fase de entrenamiento.

- **Red Neuronal Backpropagation.**

El algoritmo Backpropagation para redes multicapa es una generalización del algoritmo Perceptrón, ambos realizan su labor de actualización de pesos y ganancias con base en el error medio cuadrático. Su potencia reside en su capacidad de entrenar capas ocultas y de este modo supera las posibilidades restringidas de las redes de una única capa. El procedimiento de aprendizaje con propagación hacia atrás de los errores (Backpropagation) pertenece a la categoría de supervisado, pues requiere conocer las salidas correctas para cada ejemplo de entrada (30).

- **Arquitectura de la red Backpropagation.**

El entrenamiento de una red neuronal multicapa se realiza mediante un proceso de aprendizaje, para realizar este proceso se debe inicialmente tener definida la topología de la red. La topología de la red que se define contará con tres capas (ver figura 6), la capa de entrada, una capa oculta y la capa de salida.

- ✓ Las neuronas de la capa de entrada se obtuvieron por la cantidad de valores que toman los rasgos predictores para un total de 118 neuronas.

- ✓ No existe forma de determinar el número óptimo de neuronas en la capa oculta sin antes entrenar varias redes y estimar el error de generalización. Existen ciertas reglas llamadas “rules of thumb” (37) utilizadas para escoger la arquitectura y en la que esta investigación se apoya, las cuales, en el cálculo de las neuronas de la capa oculta se escogió la regla general:

  Ecuación 13:

  $h = (2/3) * (n + m)$, donde

n y m son la cantidad de neuronas de la capa de entrada y la capa de salida respectivamente, siendo el resultado final de 88 neuronas.

- ✓ La capa de salida tendrá 1 sola neurona que representará si se puede o no aprender el nuevo caso para almacenarlo en la BC.

En esta red no existen conexiones entre unidades de una misma capa o de unidades situadas en capas posteriores a neuronas situadas en capas anteriores; pero sí pueden existir conexiones entre unidades situadas en la capa i hasta las situadas en la capa j, siempre que j>i. Existen pesos W_{ij}asociados a las conexiones. Por cada presentación los pesos son ajustados de forma que disminuya el error entre la salida deseada y la respuesta de la red.

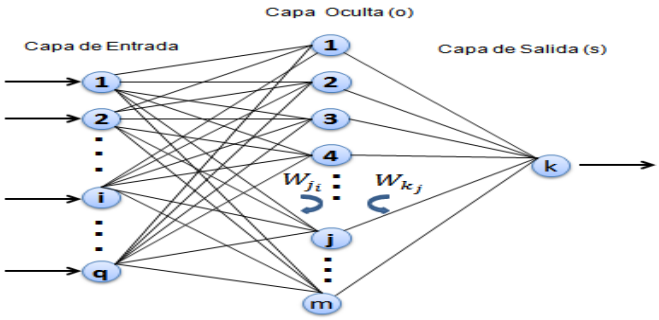

Figura 5: Topología de una red multicapa (3 capas).

Conclusiones Parciales.

En este capítulo se formalizó un modelo computacional aplicando la técnica de razonamiento basado en casos en la consideración de la estimación de la duración de un proyecto de desarrollo de software como entidad principal. Para la mejora de la BC se aplicó el método Selección de variables de Reconocimiento de Patrones y la Teoría de Testores para definir los rasgos y se eliminó el peso dado por expertos para fundamentar con basamento matemático la relevancia de los pesos seleccionados. Se definió además la función de semejanza, así como las dos funciones de comparación para los rasgos. Se definió para el desarrollo del sistema una estructura plana debido a que no modela entrada de datos incompleta y no está expuesta a pérdida de información. Para la mejora del motor de inferencia se escogió como algoritmo de recuperación el k-vecino más cercano dado que es menos sensible a pérdidas. Además para el mecanismo de aprendizaje se utilizó una red neuronal multicapa con el algoritmo de aprendizaje de Backpropagation por su capacidad de aprender patrones.

CAPÍTULO 3: CARACTERÍSTICAS Y DISEÑO DEL SISTEMA.

Introducción del capítulo.

En el presente capítulo se tratan los aspectos fundamentales para desarrollar el modelo propuesto cumpliendo con estándares de codificación y calidad definidos en la UCI. Se hace una descripción de la solución propuesta, así como del objeto a automatizar. Se detallan y explican el modelo de dominio y el modelo general. Se define la lista de rasgos informal, así como la lista de rasgos y se realiza una planeación por rasgos. También se hace referencia a los aspectos correspondientes al diseño. Se confeccionan los diagramas de clases del diseño y los diagramas de secuencia. Se describen las clases del diseño más importantes en función de establecer la estructura del sistema. Se especifican los patrones de diseño y se realiza la propuesta de interfaz de usuario para el sistema.

3.1 Propuesta del sistema.

El Sistema Experto de Estimación de Software está apoyado en Razonamiento Basado en Casos para trabajar a partir de experiencias pasadas y que los proyectos puedan obtener estimaciones lo más real posible en etapas iniciales del ciclo de vida de desarrollo del mismo. Consiste en obtener el tiempo que demora un equipo de desarrollo en elaborar un producto de software. Para lograr este objetivo el usuario debe introducir las características referentes a un proyecto de software. Mediante un recuperador, componente del RBC empleando el algoritmo de k-vecino más cercano se compara el nuevo caso que representa las características convertidas en rasgos del proyecto con los almacenados en la base de casos y devuelve los ítems más semejantes. Luego de obtenerlos, mediante el uso de un razonador se adecua mediante la adaptación nula utilizando el criterio de optimalidad de utilidad esperada una solución al nuevo caso de acuerdo con la similitud entre él y los casos obtenidos en la BC, presentando una respuesta final al usuario. Además el sistema también cuenta con un mecanismo de aprendizaje elaborado mediante una red neuronal de Backpropagation que permite guardar el nuevo caso con su solución solo si es factible para utilizarlo en futuras estimaciones.

3.2 Objeto de Automatización.

Actualmente en la UCI se cuenta con una herramienta ubicada en el paquete de herramienta del marco de trabajo del Gespro para estimar los proyectos de desarrollo de software. Esta es empleada solo en una minoría de los proyectos de la universidad y no está apoyada en ninguna técnica de inteligencia artificial

que ayude en la toma de decisiones. Este trabajo tiene como objetivo proveer a los proyectos de la UCI de una herramienta inteligente para la estimación de la duración de proyectos de software, y de esta forma automatizar este proceso; con el fin de obtener en etapas tempranas del ciclo de vida del desarrollo del software una estimación lo más real posible, según las características relevantes del proyecto en cuestión. La aplicación permitirá al personal facultado de los proyectos realizar dicha tarea utilizando los datos históricos guardados en la Base de Casos del sistema.

3.3 Diagrama de Dominio.

Se define el modelo de dominio para conocer los elementos generales, los conceptos fundamentales relacionados con la investigación, establecer sus características y relaciones. Los conceptos y sus relaciones se exponen en un lenguaje natural asequible para el entendimiento tanto de los usuarios como los desarrolladores.

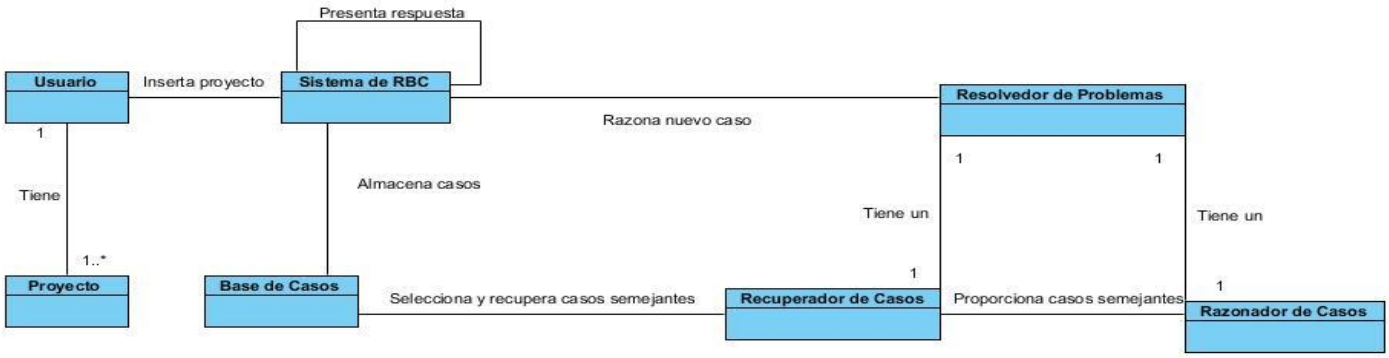

Figura 6: Modelo de Dominio del Sistema Experto de Estimación de Software.

3.3.1 Conceptos fundamentales tratados:

Usuario: Responsable de estimar los proyectos de software, por lo que se encarga de introducir en el sistema las características que definen los proyectos a estimar.

Proyecto: Es la entidad que representa un proyecto de software del que se estimará el tiempo de duración teniendo en cuenta sus características y particularidades.

Sistema de RBC: Sistema de Razonamiento Basado en Casos. Encargado de operar y visualizar el conjunto de características de un proyecto entradas por el usuario, así como dar respuesta a sus peticiones y almacenar casos en la BC.

Base de Casos: Conjunto de casos que representan proyectos estimados previamente almacenados para el análisis de las semejanzas con los rasgos que contienen y los obtenidos de la descripción insertada de un proyecto de software.

Resolvedor de Problemas: Se encarga de manejar los rasgos inicialmente insertados de un proyecto de software. Contiene un recuperador y razonador de casos.

Recuperador de Casos: Encargado de seleccionar y obtener los casos más semejantes a las descripciones de un proyecto de software entrado por el Usuario.

Razonador de Casos: Encargado de adaptar una solución a las descripciones de un proyecto, a partir de los casos semejantes que provee el recuperador de casos.

3.4 Lista de requisitos

Después de haber realizado el modelo de dominio se identificó la lista de requisitos necesaria en el desarrollo del sistema para la estimación de un proyecto de software apoyado en razonamiento basado en caso.

R1. Registrar Usuario
- Nombre
- Apellidos
- Usuario
- Contraseña
- Correo electrónico

R2. Modificar Usuario
- Datos (R1)

R3. Eliminar Usuario
- Usuario

R4. Listar Usuario

R5. Insertar Proyecto
- Nombre
- Rasgos predictores

R6. Modificar Proyecto

- Modificar cualquier dato (R5)

R7. Eliminar Proyecto

R8. Listar Proyecto

R9. Insertar Caso

- Rasgos predictores
- Rasgo objetivo

R10. Modificar Caso

- Modificar cualquier dato (R9)

R11. Eliminar Caso

R12. Listar Caso

R13. Insertar Rasgo Predictor

- Característica
- Certeza
- Valor

R14. Modificar Rasgo Predictor

- Modificar cualquier dato (R13)

R15. Eliminar Rasgo Predictor

R16. Listar Rasgo Predictor

R17. Insertar Rasgo Objetivo

- Valor

R18. Modificar Rasgo Objetivo

- Modificar cualquier dato (R17)

R19. Eliminar Rasgo Objetivo

R20. Listar Rasgo Objetivo

R21. Insertar Característica

- Nombre
- Valor
- Dominio
- Peso

R22. Modificar Característica

- Modificar cualquier dato (R21)

R23. Eliminar Característica

R24. Listar Característica

R25. Insertar Dominio

- Valores

R26. Modificar Dominio

- Modificar cualquier dato (R25)

R27. Eliminar Dominio

R28. Listar Dominio

R29. Insertar Valor

- Valor

R30. Modificar Valor

- Modificar cualquier dato (R29)

R31. Eliminar Valor

R32. Listar Valor

R33. Insertar Rol

- Nombre

R34. Modificar Rol

- Modificar cualquier dato (R33)

R35. Eliminar Rol

R36. Listar Rol

R37. Asignar Rol

R38. Estimar Duración de Proyecto

R39. Aprender Caso

3.5 Lista de Rasgos Informal.

A partir de la lista de requisitos, se procede a la definición de la lista de rasgos informal constituyendo la entrada de FDD como metodología que continúa con el ciclo de vida del software.

1. Autenticar usuario
2. Gestionar usuario
3. Gestionar rol
4. Gestionar caso
5. Gestionar proyecto

6. Gestionar rasgo predictor
7. Gestionar rasgo objetivo
8. Gestionar característica
9. Gestionar dominio
10. Gestionar valor
11. Asignar rol
12. Aprender caso
13. Estimar duración de proyecto
14. Cerrar Sesión

3.6 Modelo General.

El modelo general propuesto por FDD es utilizado para la comprensión de los conceptos fundamentales utilizados durante el trabajo con el sistema. Consiste en un diagrama de clases que representa los tipos de objetos más importantes dentro del dominio del problema, así como las relaciones entre ellos (38).

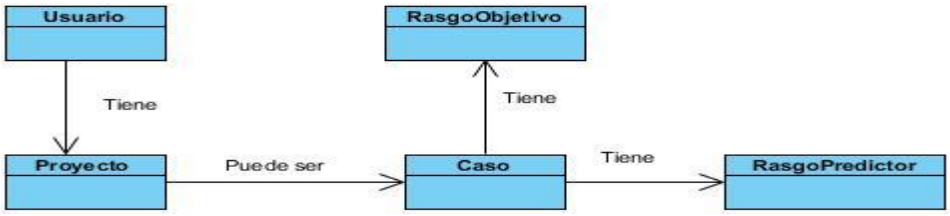

Figura 7: Modelo General del Sistema Experto de Estimación de Software.

3.6.1 Conceptos fundamentales tratados.

Usuario: Responsable de estimar los proyectos de software, por lo que se encarga de introducir en el sistema las características que definen dichos proyectos a estimar. Además si es un usuario experto este puede almacenar casos en la BC.

Proyecto: Es la entidad que representa un proyecto de software que va ser estimada por el Usuario según sus características y particularidades. Después de estimado el proyecto puede convertirse en un Caso almacenado solo sí, es factible utilizarlo en futuras estimaciones.

Rasgo Predictor: Son las características específicas o indicadores que definen a los proyectos de software.

Rasgo Objetivo: Es el resultado de estimar la duración de un proyecto de software representado en unidades temporales: días, meses y años.

Caso: Es un proyecto de software estimado, el cual se almacena en la base de casos del sistema.

3.7 Lista de Rasgos.

FDD define a partir de la lista de rasgos informales la creación de la lista de rasgos formales que resume la funcionalidad del sistema en general.

Tabla 2: Rasgos

No.	Actividades	Rasgos
1.	Autenticar usuario	1.1. Autenticar usuario
2.	Gestionar usuario	2.1. Registrar usuario 2.2. Modificar usuario 2.3. Listar usuarios 2.4. Eliminar usuario
3.	Gestionar rol	3.1. Crear rol 3.2. Modificar rol 3.3. Listar roles 3.4. Eliminar rol
4.	Gestionar caso	4.1. Crear caso 4.2. Modificar caso 4.3. Listar casos 4.4. Eliminar caso
5.	Gestionar proyecto	5.1. Crear proyecto 5.2. Modificar proyecto 5.3. Listar proyectos 5.4. Eliminar proyecto
6.	Gestionar rasgo predictor	6.1. Crear rasgo predictor 6.2. Modificar rasgo predictor 6.3. Listar rasgos predictores 6.4. Eliminar rasgo predictor
7.	Gestionar rasgo objetivo	7.1. Crear rasgo objetivo

		7.2. Modificar rasgo objetivo 7.3. Listar rasgos objetivos 7.4. Eliminar rasgo objetivo
8.	Gestionar característica	8.1. Crear característica 8.2. Modificar característica 8.3. Listar características 8.4. Eliminar característica
9.	Gestionar dominio	9.1. Crear dominio 9.2. Modificar dominio 9.3. Listar dominios 9.4. Eliminar dominio
10.	Gestionar valor	10.1. Crear valor 10.2. Modificar valor 10.3. Listar valores 10.4. Eliminar valor
11.	Asignar rol	11.1. Asignar rol
12.	Aprender caso	12.1. Aprender caso
13.	Estimar duración de proyecto	13.1. Estimar duración de proyecto
14.	Cerrar sesión	14.1. Cerrar sesión

3.8 Planeación por Rasgos.

En las siguientes tablas se muestra el cronograma de diseño y construcción; donde están planificados los rasgos por iteraciones en base a la prioridad y a la dependencia entre ellos. Se detallan la fecha de comienzo y culminación de cada rasgo y se tiene una fecha global de terminación. Este cronograma está determinado por tres iteraciones.

Tabla 3: Iteración 1

No.	Rasgos Globales	Rasgos Específicos	Fecha de inicio	Fecha de fin	Fecha de prueba	Fecha de culminación
1.	Gestionar rol	1.1. Crear rol 1.2. Modificar rol 1.3. Listar roles	25-03-2013	25-03-2013	26-03-2013	26-03-2013

		1.4. Eliminar rol				
2.	Gestionar usuario	2.1. Registrar usuario 2.2. Modificar usuario 2.3. Listar usuarios 2.4. Eliminar usuario	26-03-2013	26-03-2013	27-03-2013	27-03-2013
3.	Gestionar valor	3.1. Crear valor 3.2. Modificar valor 3.3. Listar valores 3.4. Eliminar valor	28-03-2013	28-03-2013	29-03-2013	29-03-2013
4.	Gestionar dominio	4.1. Crear dominio 4.2. Modificar dominio 4.3. Listar dominios 4.4. Eliminar dominio	1-04-2013	1-04-2013	2-04-2013	2-04-2013
5.	Gestionar característica	5.1. Crear característica 5.2. Modificar característica 5.3. Listar características 5.4. Eliminar característica	3-04-2013	4-04-2013	5-04-2013	5-04-2013

Hito: En esta iteración se obtiene una primera versión del proyecto en base a la lista de rasgos obtenidas en la fase anterior. Quedan definidas las clases Usuario, Rol, Valor, Dominio y Características con cada una de sus clases correspondientes de control y servicio y las funcionalidades definidas para cada una.

Tabla 4: Iteración 2

No.	Rasgos Globales	Rasgos Específicos	Fecha de inicio	Fecha de fin	Fecha de prueba	Fecha de culminación
1.	Gestionar rasgo predictor	1.1. Crear rasgo predictor 1.2. Modificar rasgo predictor 1.3. Listar rasgos 1.4. Eliminar rasgo	8-04-2013	10-04-2013	11-04-2013	11-04-2013

2.	Gestionar proyecto	2.1. 2.2. 2.3. 2.4.	Crear proyecto Modificar proyecto Listar proyectos Eliminar proyecto	12-04-2013	13-04-2013	14-04-2013	14-04-2013
3.	Gestionar rasgo objetivo	3.1. 3.2. 3.3. 3.4.	Crear rasgo objetivo Modificar rasgos objetivos Listar rasgos objetivos Eliminar rasgo objetivo	16-04-2013	16-04-2013	17-04-2013	17-04-2013
4.	Gestionar caso	4.1. 4.2. 4.3. 4.4.	Crear rasgo objetivo Modificar rasgo objetivo Listar rasgos objetivos Eliminar rasgo objetivo	18-04-2013	19-04-2013	20-04-2013	20-04-2013

Hito: Después de esta iteración se obtiene una segunda versión del proyecto donde quedan definidas todas las clases del dominio, control y servicio, en correspondencia con la lista de rasgo definida.

Tabla 5: Iteración 3

No.	Rasgos Globales	Rasgos Específicos	Fecha de inicio	Fecha de fin	Fecha de prueba	Fecha de culminación
1.	Gestionar asignación de roles	1.1. Asignar roles 1.2. Modificar relación 1.3. Listar relación 1.4. Eliminar relación	22-04-2013	23-04-2013	24-04-2013	24-04-2013
2.	Autenticar usuario	2.1. Autenticar usuario	25-04-2013	25-04-2013	26-04-2013	26-04-2013

3.	Cerrar sesión	3.1.	Cerrar sesión	27-04-2013	28-04-2013	29-04-2013	29-04-2013
4.	Aprender caso	4.1.	Aprender caso	30-04-2013	2-05-2013	3-05-2013	3-05-2013
5.	Estimar duración de proyecto	5.1.	Estimar duración de proyecto	6-05-2013	11-05-2013	12-05-2013	12-05-2013

Hito: Después de esta iteración se obtiene el software con todos sus requisitos y la seguridad establecida a nivel de usuario y contraseña.

3.9 Rasgos no Funcionales.

Se determinaron un conjunto de rasgos no funcionales que el sistema debe cumplir. Estas propiedades son las características que hacen al producto atractivo, usable, rápido o confiable.

3.9.1 Rasgos de Apariencia o Interfaz Externa.

La interfaz deberá ser sencilla, amigable, con colores suaves a la vista y sin cúmulo de imágenes u objetos que distraigan al usuario del objetivo de su empleo, brindando facilidades que permitan interactuar con el sistema de forma fácil y rápida.

3.9.2 Rasgos de Software.

- ✓ Se utilizará cualquier navegador para acceder al sistema, se recomienda Mozilla Firefox 17.0 o superior.
- ✓ Como SGBD se utilizará PostgreSQL 8.2.
- ✓ El servidor de aplicación, debe tener instalado el Apache Tomcat 7.0.27 como servidor web y la Máquina Virtual de Java 1.7.0.17

3.9.3 Rasgos de Hardware.

- ✓ El servidor de base de datos debe contar con una memoria RAM igual o superior a 1GB.
- ✓ Es necesario contar con una computadora para el servidor de aplicaciones con una memoria RAM superior a los 256 Mega Bytes.
- ✓ La PC cliente debe contar con una memoria RAM igual o superior a 512 MB.

3.9.4 Rasgos de Seguridad.

Seguridad y control a nivel de usuarios y contraseñas, garantizando el acceso de los mismos acorde con los roles autorizados para el uso de cada funcionalidad tanto a nivel de funciones de la aplicación como de la información de la base de datos. Las contraseñas sólo podrán ser cambiadas por el propio usuario o por el administrador informático del sistema. Además las contraseñas están encriptadas y no deben estar vacías, de ser registrado un usuario y dejar en blanco el campo de la contraseña el sistema debe emitir una alerta.

- ✓ **Confiabilidad:** el sistema debe ser capaz de recuperarse ante la ocurrencia de un fallo, de no ser posible, emitir alertas al personal encargado de la administración del mismo, así como proteger la información y contenidos.

- ✓ **Integridad:** los datos solo pueden ser cambiados por las personas autorizadas en la aplicación, en este caso, en el sistema propuesto, los usuarios solo tienen acceso a los datos a los cuales solo tienen permiso.

- ✓ **Confidencialidad:** la información puede ser accedida únicamente por las personas que tienen autorización para hacerlo.

- ✓ **Disponibilidad:** el sistema debe estar disponible cuando el usuario requiera de su uso, por lo que debe existir una persona encargada de mantener el sistema activo.

3.10 Diseño por Rasgos.

3.10.1 Diagrama de Clases.

Los diagramas de clases que se muestran en el presente trabajo están guiados por los rasgos. Representan las clases que serán utilizadas dentro del sistema y las relaciones estructurales que existen entre ellas. El diseño de una clase incluye definiciones para atributos y operaciones.

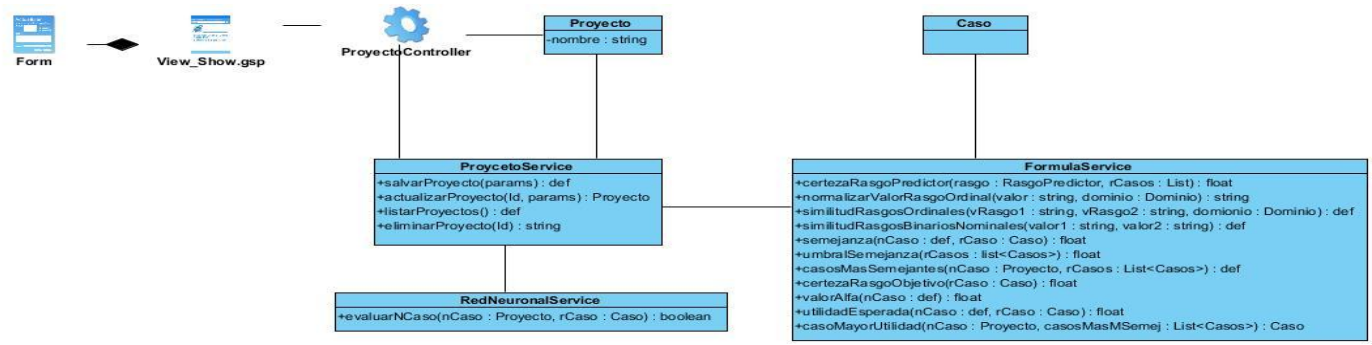

Figura 8: Diagrama de Clase del rasgo Estimar duración de proyecto.

Ver Anexo 4.

3.10.2 Diagrama de secuencia.

Se elaboró un diagrama de secuencia del Sistema Experto de Estimación de Software, el cual constituye una representación que muestra, el escenario del rasgo Estimar duración de proyecto, los eventos generados por los usuarios, su orden, así como los eventos internos del sistema.

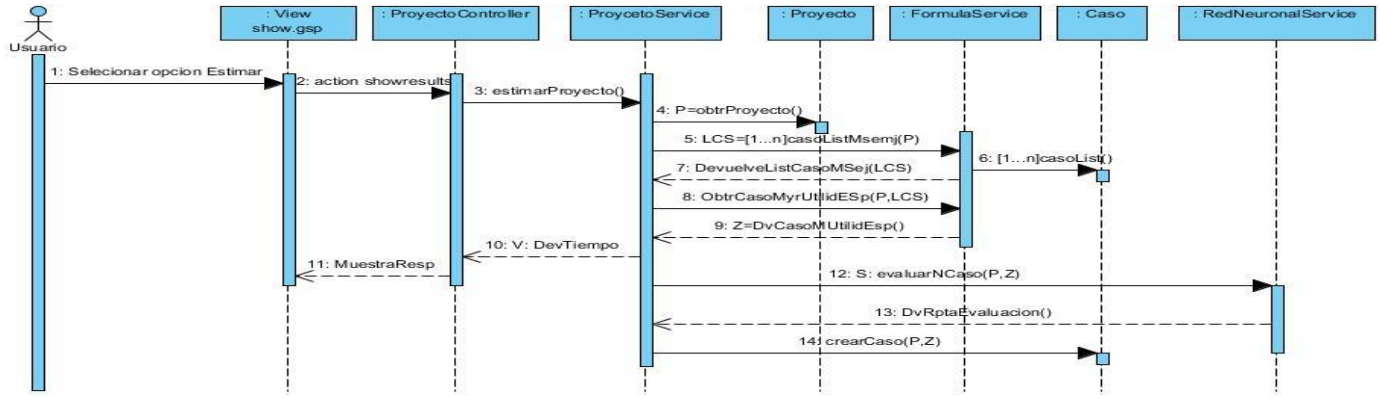

Figura 9: Diagrama de secuencia para el rasgo Estimar duración de proyecto.

- **Descripción del rasgo Estimar duración de proyecto.**

El rasgo Estimar duración de proyecto se inicializa desde la vista de mostrar proyecto después de un usuario haber introducido los datos pertenecientes a un proyecto de software y selecciona la opción “Estimar” mandando una acción al controlador de proyecto y este manda una petición al servicio correspondiente. Desde el servicio de proyecto se obtienen los datos de dicho proyecto mediante una llamada a la clase del dominio de proyecto, a partir de estos se obtiene desde un servicio de fórmulas la

lista de casos más semejantes y luego de esta se adquiere el caso de mayor utilidad esperada. Obtenido este caso se estima y se envía hacia la vista por medio del controlador el resultado y a su vez desde el servicio de proyecto se envía el proyecto con su resultado hacia el servicio de red neuronal para evaluar si se puede almacenar en la base de casos como un nuevo caso, si es afirmativa dicha evolución se crea el caso y se almacena.

3.11 Patrones Utilizados.

Un patrón es una solución reutilizable de problemas recurrentes que ocurren durante el desarrollo del software. Ayudan a entender las soluciones del problema con un vocabulario similar lo que permite un mejor entendimiento.

3.11.1 Patrones arquitectónicos.

Un patrón arquitectónico describe aspectos fundamentales de la estructura de un sistema software. Especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes.

El patrón **Modelo-Vista-Controlador (Model-View-Controller**) es un estilo arquitectónico empleado en la construcción de aplicaciones web. El mismo separa la parte lógica de la aplicación de su presentación. Grails implementa el patrón MVC, facilitando el desarrollo del proyecto web.

Las vistas son un archivo con extensión gsp, representan la capa de Presentación y muestran al usuario el estado actual del modelo de datos y los resultados de las distintas acciones posibles. Los controladores pertenecen a la capa de Control y son los encargados de proporcionar la comunicación entre las vistas y el modelo; el modelo son las clases de dominio de Groovy incluidas en la capa Dominio, que se mapean en la base de datos utilizada y permiten el manejo y almacenamiento de los datos. Adicionalmente se define en Grails una capa de Servicios la cual contiene las entidades encargadas de manejar la lógica de negocio, facilitando el mantenimiento y promoviendo la reutilización de código.

3.11.2 Patrones de diseño.

Un patrón de diseño es una solución a un problema en un determinado contexto. Estos son sobre aspectos relacionados con el diseño de los subsistemas. Por tanto se centran en aspectos más específicos.

Patrones GOF (Gang of Four).

Singleton: es una forma de asegurar una única instancia de una clase. Se utiliza en la instanciación de los servicios en los controladores. Por defecto en Grails todos los servicios son Singleton. Ejemplo en la clase ProyectoController se instancia un servicio: def proyectoService, también en la clase CasoController se instancia el servicio mediante def casoService, así se hace por cada uno de los controladores donde se crea una instancia de su servicio correspondiente.

Conclusiones Parciales.

En este capítulo se expuso la descripción de la propuesta del sistema, se definieron las Listas de Rasgos y los Rasgos no Funcionales, permitiendo definir de forma concreta las funcionalidades que automatiza la solución. Se construyeron el modelo de dominio y el modelo general en los que se mostraron los conceptos fundamentales, las clases y sus relaciones. Se realizó la planeación por iteraciones donde están planificados los rasgos en base a la prioridad y a la dependencia entre ellos, organizados en 3 iteraciones que tenían como objetivos guiar el desarrollo, y obtener un avancen del proyecto al finalizar cada una. Se definieron los diagramas de clases del diseño de los rasgos fundamentales, logrando determinar las principales clases del sistema a construir, así como su descripción para un mejor entendimiento. Además se elaboró el diagrama de secuencia para el rasgo Estimar Proyecto, constituyendo este el más representativo del sistema.

IMPLEMENTACIÓN Y PRUEBAS.

Introducción del capítulo.

En el presente capítulo se define el diagrama de despliegue y se continúa con la fase de la metodología FDD Construir por rasgos. Además se definen y describen las pruebas que se realizaron a la aplicación.

4.1 Vista de Despliegue.

La vista de despliegue o también conocida como vista de implementación representa los nodos que forman la topología hardware sobre la que se ejecuta el sistema.

4.1.1 Diagrama de Despliegue.

En la presente investigación se apoya en el diagrama de despliegue UML para mostrar la situación física de los componentes lógicos desarrollados. Es decir situar el software en el hardware que lo contiene. Cada hardware se representa como un nodo.

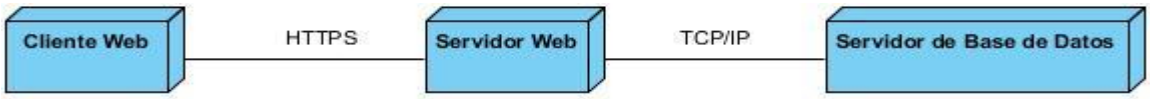

Figura 10: Diagrama de Despliegue del Sistema Experto de Estimación de Software.

Un cliente web, es una PC tanto portátil como de escritorio, la cual tiene un sistema operativo instalado y un navegador disponible, preferentemente Firefox en cualquier versión. El servidor web, es una PC de escritorio o un servidor, la cual tiene instalado el servidor web, que no es más que una aplicación que procesa cualquier aplicación del lado del servidor permitiendo conexiones con clientes generando una respuesta en lenguaje Groovy. Las conexiones con el servidor son realizadas a través del protocolo HTTPS. El servidor de Base de Datos es una computadora que tiene instalado un programa que permite organizar los datos del proceso de la estimación de los proyectos de software en una o más tablas relacionadas.

4.2 Pruebas al sistema.

Las pruebas permiten determinar el estado de la calidad del producto software. En este proceso se ejecutan pruebas dirigidas a componentes del software o al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los rasgos, los resultados se observan y registran y se realiza una evaluación de algún aspecto.

4.2.1 Estrategia de prueba seguida.

La estrategia seguida para la realización de las pruebas al Sistema Experto de Estimación de Software contempla dos niveles: el nivel de pruebas de Unidad y el nivel de prueba de Sistema. A su vez en el primer nivel, se realizaron pruebas automáticas haciendo uso del sistema de Testing o Prueba que brinda el propio Framework utilizado. Para el segundo nivel de pruebas se emplea el método de caja negra. Además el sistema fue validado por medio de una prueba piloto.

- **Prueba de Unidad**

Los test unitarios representan la mejor alternativa para encontrar y corregir la mayoría de los errores de codificación. En ellos se verifica que un método se comporta como debería, sin tener en cuenta su entorno. Esto significa que cuando se ejecutan las pruebas unitarias de un método Grails no inyectará ninguno de los métodos dinámicos con los se cuenta la aplicación cuando se está ejecutando. Así que, cuando se trabaja con entidades o servicios son los desarrolladores los responsables de crear y gestionar todos los objetos. Durante el proceso de desarrollo, se puede probar el estado de la aplicación mediante el comando: **grails test-app.** Ver Anexo 5.

- **Prueba de Caja Negra**

Las pruebas de caja negra son aquellas que se realizan a la interfaz del software para demostrar que las funciones son operativas. Se llevaron a cabo según la planificación en tres iteraciones (Ver sección 3.8). Para estas pruebas se realizaron planillas por cada rasgo global (Ver tabla 8) y posteriormente se verificó que cada funcionalidad mostrara el resultado esperado. En la primera iteración se encontraron 2 no conformidades y en la segunda 1 no conformidad que fueron mitigadas antes de pasar a la siguiente iteración (Ver figura 19).

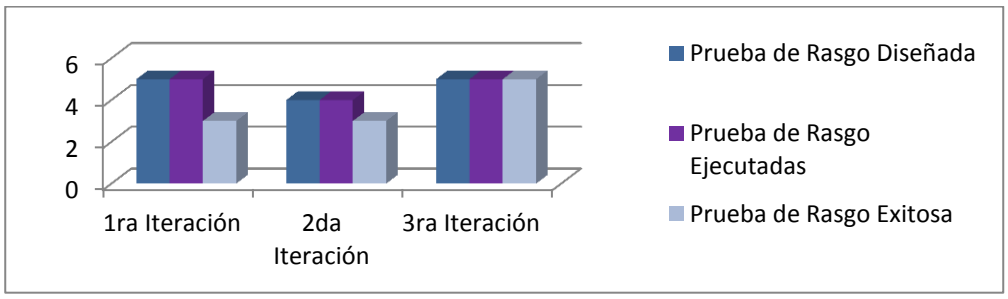

Figura 11: Resultado de las pruebas de caja negra

Tabla 6: Prueba para el rasgo global Gestionar Proyecto.

Rasgo Global	Gestionar proyecto
Rasgo específico	Crear proyecto.
Entrada	Nombre y Conjunto de rasgos predictores que describen un proyecto.
Precondiciones	El usuario debe estar autenticado.
Resultados	Se inserta un proyecto.
Rasgo específico	Modificar proyecto.
Entrada	Nombre o rasgo predictor del proyecto a modificar.
Precondiciones	El usuario debe estar autenticado. Debe existir el proyecto.
Resultados	Se modifica proyecto.
Rasgo específico	Eliminar proyecto
Entrada	Proyecto.
Precondiciones	El usuario debe estar autenticado. Debe existir el proyecto insertado.
Resultados	Se elimina el proyecto.

Rasgo específico	Listar proyectos		
Entrada			
Precondiciones	El usuario debe estar autenticado. Deben existir proyectos.		
Resultados	Muestra los proyectos existentes.		
Rasgo global	Gestionar proyecto.		
Número	**Nombre del campo**	**Nulo**	**Descripción**
1	Nombre del proyecto	No	Se debe introducir nombre del proyecto.
2	Rasgos predictores	No	Se deben introducir los rasgos predictores del proyecto.
3	Proyecto.	No	Se debe seleccionar el proyecto en la lista desplegable.

En el Anexo 6 se muestran las planillas para las pruebas por cada uno de los rasgos que definen el sistema.

- **Prueba Piloto.**

El desarrollo de la prueba piloto tiene como objetivo demostrar que el diseño del Sistema Experto de Estimación de Software funciona de la manera deseada en la Universidad y que cumple con los requisitos necesarios para estimar el tiempo de duración de los proyectos de software de gestión. De esta manera se tiene seguridad plena de que el sistema es realmente útil en el proceso de desarrollo del software y define con un alto grado de certeza el tiempo de duración de los proyectos de software.

La prueba piloto se realizó en el centro de Telemática de la facultad 2 de la UCI a los proyectos de software de gestión. El sistema fue ubicado en un servidor donde los usuarios encargados de realizar la estimación en los proyectos de software podían acceder al mismo.

- **Resultados de la prueba piloto.**

Se realizaron 2 iteraciones de prueba. En la primera iteración se manifestó una no conformidad y un pedido de cambio (Ver tabla 9). La no conformidad presentada se mitigó antes de pasar a la segunda iteración que se llevó a cabo.

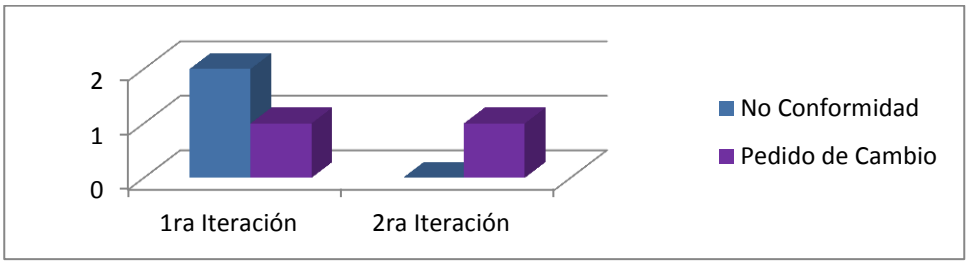

Figura 12: Resultado de la prueba piloto

Conclusiones Parciales.

En el presente capítulo se representó mediante el diagrama de despliegue la topología hardware sobre la que se ejecuta el sistema. Se probó la solución informática mediante los niveles de pruebas de Unidad haciendo uso del sistema de Testing o Prueba que brinda el propio framework utilizado y de Sistema empleando el método de Caja Negra, además de la realización de una prueba piloto.

CONCLUSIONES GENERALES.

En la presente investigación se formalizó y se desarrolló un modelo computacional que se denomina Sistema Experto de Estimación de Software donde se aplicó la técnica de razonamiento basado en casos en la consideración de la estimación de la duración de un proyecto de desarrollo de software como entidad principal. Se elaboró un marco teórico de la investigación identificando las principales tendencias de la estimación de los proyectos de software. Para la elaboración del sistema se utilizaron un conjunto de herramientas entre las que se encuentran IntelliJ IDEA11.1.4 como IDE de desarrollo, Visual Paradigm 8.0para el modelado, Grails 2.0.1, JQuery 1.8 y Encog como framework, Groovy 2.0como lenguaje de programación y UML5.0 de modelado y como SGBD se empleó PostgreSQL. El Sistema de Razonamiento Basado en Casos está compuesto por una Base de Casos con una estructura plana y un motor de inferencia que emplea como algoritmo de recuperación el k-vecino más cercano y una adaptación nula o por reinstanciación. Además el sistema posee un mecanismo de aprendizaje empleando una red neuronal de Backpropagation para saber si un caso puede ser aprendido y almacenado para reutilizarlo en estimaciones futuras. Por otra parte se validó el Sistema Experto de Estimación de Software mediante pruebas unitarias y pruebas pilotos, obteniendo buenos resultados.

RECOMENDACIONES.

Luego de cumplir los objetivos trazados y en vista de mejorar el funcionamiento del sistema e incrementar las funcionalidades brindadas, se recomienda:

- ✓ Aplicar esta propuesta en los proyectos de gestión de la Universidad, con el fin de obtener mejores resultados en el proceso de estimación.
- ✓ Guardar los datos de la BC en una base de datos centralizada, permitiendo mayor seguridad sobre los mismos y que los planificadores de los proyectos puedan contar con información actualizada.
- ✓ Investigar y tener en cuenta otras funciones de comparación para los rasgos predictores.

REFERENCIAS BIBLIOGRÁFICAS

1. **Juan Palacio.** Origen de la gestión de proyectos. *Navegapolis.net.* [En línea] 2006. [Citado el: 15 de 11 de 2012.] http://www.navegapolis.net/files/s/NST-001_01.pdf. NST-0001, Rev. 0.1.

2. **Brito, Dayami Rodríguez.** *Diseño Teórico- Metodológico para la definición del Método de Estimación para los proyectos de desarrollo de software de la Universidad de las Ciencias Informáticas (UCI).* Ciudad Habana : s.n., 2009.

3. **Pessman, Roger S.** *Software Engineering. A Practitioner's Approach. Seventh Edition.* ISBN 978-0-07-337 597 -7.

4. *Sistemas Basados en Casos & Sistemas de Enseñanza-Aprendizaje Inteligentes.* **Natalia Martínez Sánchez, María Matilde García Lorenzo, Zoila Zenaida García Valdivia.** Numero 11. Revista Iberoamericana de Informática, s.l. : ADIE, Asociación para el Desarrollo de la Informática Educativa, Enero - Junio 2010. ISSN: 1699-4574.

5. **Ayala, Alejandro Peña.** *Sistemas basados en Conocimiento: Una Base para su Concepción y Desarrollo.* Revillagigedo 83, Centro Histórico, 06070, México, D.F. : D.R. ©, Primera Edición 2006. ISBN: 970-94797-4-1.

6. **Pedro Carlos Abreu Jiménez, Jeny Del Sol Chang.** *Base de Casos para estimar la duración de un proyecto de desarrollo de software.* Ciudad de la Habana : s.n., junio 2012.

7. **Ovejero, Jose Daniel.** *Estimación de Proyectos Para Sistemas Basados en Conocimiento.* Buenos Aires : s.n., 2006.

8. **S.Capuchino, Ana Ma Moreno.** Unidad Didáctica 4: Estimación de Proyectos Software. *Control y gestión de proyectos software.* 1996.

9. *Comparative Analysis of Software Effort Estimation Techniques.* **P.K. Suri, PhD Dean, Pallavi Ranjan.** –No.21, s.l. : International Journal of Computer Applications (0975 – 8887), June 2012, Vol. Volume 48.

10. **CAO, M. ING. JOSÉ IGNACIO.** *PRINCIPIOS PARA UN MÉTODO DE ESTIMACIÓN DE PROYECTOS DE SOFTWARE BASADO EN LOS ESCENARIOS PRINCIPALES.* Buenos Aires : s.n., 2006.

11. **Bueno, Carlos Blanco.** Ingenieria de Software II. Tema 06. Gestion del tiempo. *OpenCourseWare.* [En línea] 2011. [Citado el: 28 de 11 de 2012.] http://ocw.unican.es/ensenanzas-tecnicas/ingenieria-del-software-ii/materiales/tema6-gestionTiempo.pdf. Creative Commons BY-NC-SA 3.0.

12. SEER by Galorath. *Proyects Delivered... On Time, On Budget. As Especified.* [En línea] 2011. [Citado el: 10 de 12 de 2012.] http://www.galorath.com/.

13. QSM The Intelligence Behin Successful Software Proyect. *Slim Estimate.* [En línea] Quantitative Software Management., 2010. [Citado el: 12 de 12 de 2012.] http://www.qsm.com/tools/slim-estimate.

14. CONSTRUX. *Construx Estimate.* [En línea] [Citado el: 12 de 12 de 2012.] http://www.construx.com/Resources/Construx_Estimate/. © 2013 Construx Software Builders, Inc.

15. **VALVERDE, DR. NICOLAS KEMPER.** Sociedad de Estudiantes de Ciencias de la Computacion. *INTELIGENCIA COMPUTACIONAL APLICADA EN REDES DE TELECOMUICACIONES.* [En línea] 2006. [Citado el: 13 de 12 de 2012.] http://www.seccperu.org/eaec/slides/inteligenciaComputacional_Redes_Telecomunicaciones-Nicolas_Kemper_Valverde.pdf. ®2011 SECC .

16. **Lio, Dr. Daniel Gálvez.** *Sistemas Basados en Conocimiento.* Villa Clara, Cuba : s.n., 1998.

17. **José A. Alonso Jiménez, Miguel A. Gutiérrez Naranjo.** Dpto de Ciencias de la Computación e Inteligencia Artificial. *Introducciónn a la Ingeniería del Conocimiento. Tema 2: Introducción a los sistemas basados en conocimiento.* [En línea] 2003. [Citado el: 14 de 12 de 2012.] http://www.cs.us.es/cursos/iic-2003/temas/tema-02-iic04.pdf.

18. *APLICACIONES MÉDICAS COMO AYUDA AL DIAGNÓSTICO EN LA MEDICINA. EXPERIENCIA SOFTEL - MINSAP.* **MsC. Ing. Mirna Cabrera Hernández, MsC. Lic. María del Carmen Paderni López, Lic. Ramón Hita Torres, MsC. Dr. Ariel Delgado Ramos, MsC. Ing. María Antonia Tardío López, MsC. Dr. Denis Derivet Thaureaux.** 2, s.l. : Revista Cubana de Informática Médica., 2012.

19. **Ivar Jacobson, Grady Booch, James Rumbaugh.** El Proceso Unificado de Desarrollo de Software. [En línea] [Citado el: 8 de 01 de 2013.] http://bibliodoc.uci.cu/pdf/8478290362.pdf. ISBN: 84-7829-036-2.

20. **Morpeceres, Alberto.** *Procesos de Desarrollo: RUP, XP y FDD .* 15-12-2002. Copyrigh (C) 2002.

21. **Dasiel Cordero Morales, Yoanny Torres Rubio.** *Sistema Inteligente de Mitigación de Riesgos para el Centro ISEC.* La Habana : s.n., 2011.

22. **Dierk Koenig, Andrew Glover, Paul King, Guillaume Laforge, Jon Skeet.** *Groovy in Action.* 2007. ISBN:1932394842 .

23. **GRANADOS, ALEJANDRO GARCÍA.** UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO. *Desarrollo Ágil de Aplicaciones Web con Grails Framework. Caso de estudio: PROMEPUAEH.* [En línea] enero de 2012. [Citado el: 15 de 01 de 2013.] http://www.uaeh.edu.mx/nuestro_alumnado/icbi/monografias/desarrollo%20agil.pdf.

24. JavaHispano. *Tu lenguaje. Tu Comunidad.* [En línea] [Citado el: 13 de 04 de 2012.] http://www.javahispano.org/antiguo_javahispano_org/2008/9/24/encog-10-combinacion-de-redes-neuronales-y-bots.html. © 2001-2012 Asociación javaHispano .

25. **Sánchez, Jorge.** Etnassoft. OPENLIBRA" La biblioteca libre que andabas esperando". *apuntes completos Sistemas de Gestores de Base de Datos.* [En línea] [Citado el: 7 de 01 de 2012.] http://www.etnassoft.com/biblioteca/sistemas-gestores-de-bases-de-datos/.

26. **Yeni Morgado Sánchez, Aldo Cristiá Alvarez.** *Guia para la Personalizacion de PosgreSQL 8.4.* Ciudad de La Habana : s.n., junio de 2010.

27. **José Ruiz Shulcloper, Adolfo Guzmán Arenas, J. Francisco Martínez Trinidad.** *Enfoque lógico combinatorio al reconocimiento de patrones. I. Selección de variables y clasificación supervisada.* Mexico : s.n. ISBN: 970-18-2059-2 (colección), ISBN: 970-18-2384-1.

28. **Laura Lozano, Javier fernández.** Departamento de Informatica (ATC, CCIA, LSI). *Razonamiento Basado en Casos: Una Visión General. .* [En línea] 21 de 05 de 2012. [Citado el: 03 de 02 de 2013.] http://www.infor.uva.es/~calonso/IAI/TrabajoAlumnos/Razonamiento%20basado%20en%20casos.pdf.

29. **Pupo, Karina Reyna.** *Propuesta de motor de inferencia de una base de casos para la estimación de la duración de un proyecto de desarrollo de software.* La Habana : s.n., junio 2012.

30. **Rafael Esteban Bello Perez, Zoila Zenaida Garcia Valdivia, Maria M. Garcia Lorenzo, Antonio Reynoso Lobato.** *Aplicaciones de la Inteligencia Artificial. .* Mexico : s.n., 2002. ISBN: 970-27-0177-5.

31. *El Razonamiento Basado en Casos en el ámbito de la Enseñanza/Aprendizaje.* **MSc. Natalia Martínez Sánchez, Dra. Gheisa Ferreira Lorenzo, Dra. María M. García Lorenzo, Dra. Zenaida García Valdivia.** 10, Villa Clara, Cuba : Revista de Informática Educativa y Medios Audiovisuales , 2008, Vol. Vol. 5. ISSN 1667-8338, © LIE-FI-UBA..

32. **González, Miguel García-Serrano.** *EVOLUCIÓN DE PROYECCIONES LINEALES PARA APRENDIZAJE AUTOMÁTICO.* Leganés, Madrid : s.n., JULIO 2009.

33. **Tamara Martínez Labaut, Yaima Antúnez Ojeda.** *PROPUESTA DE TÉCNICAS DE APRENDIZAJE DE ELEMENTOS VIRTUALES.* Ciudad de la Habana : s.n., junio 2008.

34. **Fernández, Jean-Pierre Lévy Mangin (director). Raquel Flórez López. José Miguel Fernández.** *Redes neuronales artificiales. Fundamentos teóricos y aplicaciones prácticas. Serie Metodología y Análisis de datos en Ciencias Sociales.* Espana : Gesbiblio, S.L., 2008. ISBN: 978-84-9745-246-5.

35. *Sistemas híbridos neuro-simbólicos: una revisión.* **Florentino Fdez-Riverola, Juan M. Corchado.** 11, Espana : Revista Iberoamericana de Inteligencia Artificial, 2000, Vol. Vol. 4. ISSN(Printed version)1137-3601.

36. **Arianna Herrera Bacallao, Yunerkis Prevot Urgellés.** *APLICACIONES DE LAS REDES NEURONALES EN ENTORNOS VIRTUALES.* Ciudad de La Habana : s.n., Junio, 2008.

37. **A., Matías Mattamala.** *EL4106 Inteligencia Computacional. Red Neuronal Feed-Forward con Backpropagation.* Chile : Ingeniería Eléctrica. Facultad de Ciencias Físicas y Matemáticas. Universidad de Chile, 2013.

38. **Stephen R. Palmer, John M. Felsing.** *A Practical Guide to Feature Driven Development.* s.l. : Prentice-Hall, 2002. ISBN-10: 0130676152, ISBN-13: 978-0130676153.

39. el escritorio de Alejandro. *Comportamiento de proyectos TI: Están en deuda!* [En línea] 5 de 01 de 2010. [Citado el: 29 de 11 de 2012.] http://www.alejandrobarros.com/content/view/691759/Comportamiento-de-proyectos-TI-Estan-en-deuda.html.

40. **Constanten, Yadira Ruiz.** *Elementos a considerar en la integración de los métodos de Boehm y Humphrey para la estimación en la duración de un proyecto de software.* La Habana, Cuba : s.n., junio 2007.

41. **Jetbrains.** IntelliJIDEA. [En línea] WebStorm. [Citado el: 10 de 01 de 2013.] http://www.jetbrains.com/idea/. © 2000-2013 JetBrains. All rights reserved.

42. **Carderon, Segundo Federico Arévalo.** UTE Ecuador. Repositorio Digital. *Buscador Inteligente de Informacion en la Web con Procesamiento de Lenguaje Natural y Agentes Inteligentes.* [En línea] 2004. [Citado el: 13 de 01 de 2013.] http://repositorio.ute.edu.ec/bitstream/123456789/5577/1/23189_1.pdf. Copyright © 2002-2004 .

43. **Olabe, Xabier Basogain.** Campo Virtual Airtuala. OpenCourseWare. *Redes Neuronales Artificiales y sus Aplicaciones. .* [En línea] [Citado el: 20 de 03 de 2012.] http://cvb.ehu.es/open_course_ware/castellano/tecnicas/redes_neuro/contenidos/pdf/libro-del-curso.pdf. © 2005 CVB.

44. **Alexander, Christopher.** *A Pattern Language .* Nueva York Editorial : Oxford University Press , 1978. ISBN-10: 0195019199, ISBN-13: 978-0195019193.

45. **Brito, Nacho.** *manual de desarrollo wed con Grails. JavaEE Como siempre debio haber sido.* 2009. ISBN: 978-84-613-2651.

46. **Larman, Craig.** *UML y Patrones. Introducción al Análisis y Diseño Orientado a Objetos.* s.l. : PRENTICE HALL, 2010. ISBN-10: 8420534382, ISBN-10: 8420534382.

47. **Matthew J. Zagumny, Ph.D.** *The SPSS Book. United States of America.* 2001. 0-595-18913-X.

48. **UNIVERSIDAD DE CHILE. VICERRECTORÍA DE ASUNTOS ACADÉMICOS. DEPARTAMENTO DE EVALUACIÓN, MEDICIÓN Y REGISTRO EDUCACIONAL.** *NOCIONES BÁSICAS DE ESTADÍSTICA UTILIZADAS EN EDUCACIÓN.* SANTIAGO : s.n., septiembre de 2008.