UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 7

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

TÍTULO: *Herramienta para la estimación de proyecto*

AUTORES: Rafael Dacal Diaz

Jorge G. Tamayo Hernández

TUTORAS: Ing. Yaima Anido González

Ing. Yanitza Ramírez Stambor

Ciudad de la Habana, junio de 2010

"Año 52 de la Revolución"

Ante la necesidad de realizar un buen proceso de estimación de proyectos en el Centro de Informática Médica (CESIM) 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: “Herramienta para la estimación de proyectos”, 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. Para esto se utilizará la técnica de estimación por analogía, la cual permite comparar proyectos, siempre y cuando estos presenten características similares. Además se empleará el Razonamiento Basado en Casos, el cual permitirá recuperar los proyectos semejantes al nuevo, desde la base de conocimiento, la que a su vez almacenará los datos históricos de los proyectos concluidos.

El uso de esta herramienta facilitará la estimación de los proyectos en el CESIM, así como, permitirá a los planificadores realizar una mejor planificación de los recursos con que se cuenta para el desarrollo del software.

Tabla de Contenido

Introducción

El Proceso de gestión para la creación de un Sistema o software, es una tarea importante para poder controlar un proyecto y asegurar su éxito. Comienza con una serie de actividades que se denominan planificación del proyecto, las cuales tienen como objetivo proporcionar un marco de trabajo que permita al gestor de planificación hacer estimaciones razonables, proporcionando valores aproximados de recursos, costos y planificación temporal. Esa aproximación, implica un cierto grado de riesgo e incertidumbre; sin embargo, contar con dicha información en etapas tempranas de desarrollo, permite tomar decisiones importantes antes llevarlo a cabo.

El desarrollo del software requiere de la estimación como una herramienta para controlar y administrar los recursos que se necesitan y que se utilizan antes y durante el proyecto. No se puede considerar a la estimación como una ciencia exacta ya que existen numerosas variables humanas, técnicas, del entorno y políticas, entre otras, que intervienen en su proceso y que pueden afectar los resultados finales. Sin embargo, cuando es llevada a cabo en forma sistemática, se pueden lograr resultados con un grado aceptable de riesgo y convertirla en un instrumento útil para la toma de decisiones. (Ovejero, 2006)

La estimación de proyectos, es un aspecto que en el Centro de la Informática Médica (CESIM) se tiene en cuenta, para garantizar el éxito del desarrollo de los mismos. A pesar de que en el CESIM se realizan las estimaciones correspondientes a los productos a desarrollar, los resultados obtenidos no son satisfactorios, ya que, estos proveen tiempos irreales de terminación de las aplicaciones, que luego el planificador debe ajustar. Actualmente las pocas estimaciones que se realizan, se hacen empleando como herramienta un sistema basado en Microsoft Office Excel 2007, que no cumple con las necesidades y en la cual se utilizan parámetros que los planificadores no comprenden en su totalidad.

Además cuando un proyecto es liberado, en el centro no se guardan los datos referentes a su estimación correspondiente. Igualmente no se guardan referencias históricas, que permitan poseer un conocimiento sobre las actividades efectuadas durante su desarrollo, ni los recursos empleados en su implementación. Por lo que actualmente no se sabe reconocer dado un nuevo proyecto, si

este tiene similitudes en cuanto a su duración, cantidad de recursos, así como otras especificaciones que permitan establecer una comparación con proyectos previamente desarrollados, es decir no existe un mecanismo que sea capaz de evaluar el software desarrollado y a partir de esta experiencia estimar el nuevo producto.

Luego de concluir el análisis de la situación existente, se presenta el siguiente **problema a resolver:** ¿Cómo estimar los proyectos del CESIM a partir de datos históricos?

Se define como **objeto de estudio** el proceso de gestión de proyectos en el CESIM y como **campo de acción** el proceso de estimación de proyectos en el CESIM. Siendo el **objetivo general** de la investigación desarrollar una herramienta que permita estimar los proyectos en el CESIM a partir de los datos históricos.

Para satisfacer las necesidades planteadas es necesario establecer las siguientes actividades como las **tareas de la investigación**:

1. Analizar el proceso de estimación de proyectos en CESIM.
2. Analizar las soluciones informáticas existentes relacionadas con la estimación de proyectos, basadas en datos históricos.
3. Diseñar la Base de Conocimientos.
4. Definir las métricas de estimación de proyectos a utilizar.
5. Seleccionar el motor de búsqueda basado en Razonamiento Basado en Casos.
6. Seleccionar las herramientas y tecnologías más adecuadas para el desarrollo de la aplicación.
7. Diseñar la aplicación de estimación de proyectos.
8. Implementar la aplicación.

CAPÍTULO 1

Fundamentación Teórica

El presente capítulo tiene como objetivo abordar los principales conceptos y aspectos más significativos relacionados con las temáticas abordadas en el trabajo de diploma. A partir de disímiles fuentes bibliográficas se profundizó en el estado del arte de la base de conocimientos, de las herramientas automatizadas de estimación, así como la técnica del razonamiento basado en casos. Se definen las variables y las métricas capaces de caracterizar proyectos. Se abordan además las herramientas a utilizar para el desarrollo de la aplicación.

1.1. Técnicas de estimación. (Capuchino, 1996)

Existen técnicas útiles para la estimación de costos de tiempo. Y dado que la misma es la base de todas las demás actividades de planificación del proyecto y sirve como guía para una buena Ingeniería de Sistemas y Software. Al estimar se toman en cuenta no solo los recursos, costos y planificación, sino que también el procedimiento técnico a utilizar en el proyecto. El tamaño del proyecto es otro factor importante que puede afectar la precisión de las estimaciones. A medida que el tamaño aumenta, crece rápidamente la interdependencia entre varios elementos del Software. La disponibilidad de información histórica es otro elemento que determina el riesgo de la estimación.

Aunque cada una tiene sus puntos fuertes y débiles, todas tienen en común los siguientes atributos:

1. Se han de establecer de antemano el ámbito del proyecto.
2. Como bases para la realización de estimaciones se utilizan métricas del software de proyectos pasados.
3. El proyecto se desglosa en partes más pequeñas que se estiman individualmente.

Algunas de estas técnicas son:

Modelado del algoritmo de costos: Se desarrolla un modelo utilizando información histórica de costos que relaciona alguna métrica de software (por lo general, su tamaño) con el costo del proyecto. Se hace una estimación de esa métrica y el modelo predice el esfuerzo requerido.

Opinión de expertos: Se consultan varios expertos en las técnicas de desarrollo de software propuestas y en el dominio de aplicación. Cada uno de ellos estima el costo del proyecto. Estas estimaciones se comparan y discuten. El proceso de estimación se itera hasta que se acuerda una estimación.

Estimación por analogía: Esta técnica es aplicable cuando otros proyectos en el mismo dominio de aplicación se han completado. Se estima el costo de un nuevo proyecto por analogía con estos proyectos completados.

Ley de Parkinson: La Ley de Parkinson establece que el trabajo se extiende para llenar el tiempo disponible. El costo se determina por los recursos disponibles más que por los objetivos logrados. Si el software se tiene que entregar en 12 meses y se dispone de cinco personas, el esfuerzo requerido se estima en 60 personas-mes.

Asignar costos para ganar: El costo del software se estima dependiendo de lo que el cliente esté dispuesto a pagar por el proyecto. El esfuerzo estimado depende del presupuesto del cliente y no de la funcionalidad del software.

Cada técnica de estimación tiene sus propias fortalezas y debilidades. Para proyectos grandes, se deben utilizar varias técnicas de estimación de costos y comparar sus resultados. Si éstos predicen costos radicalmente diferentes, esto indica que no se tiene suficiente información para generar los costos. Se debe buscar más información y repetir el proceso hasta que la estimación converja.

Un gran error en la estimación del costo puede ser lo que marque la diferencia entre beneficios y perdidas, la estimación del costo y del esfuerzo del software nunca será una ciencia exacta, son demasiadas las variables: humanas, técnicas, de entorno, políticas, que pueden afectar el costo final del software y el esfuerzo aplicado para desarrollarlo.

Para realizar estimaciones seguras de costos y esfuerzos se tienen varias opciones posibles:

- Dejar la estimación para más adelante (obviamente podemos realizar una estimación al cien porciento fiable después de haber terminado el proyecto).
- Base la estimación en proyectos similares ya terminados.
- Utilice las técnicas de descomposición relativamente sencillas para generar estimaciones de costos y esfuerzo del proyecto.
- Desarrolle un modelo empírico para el cálculo de costos y esfuerzo del software.

Desdichadamente la primera opción, aunque atractiva no es práctica, ya que el objetivo de la estimación es brindar una aproximación de los recursos, costo y tiempo que se necesitarán para el desarrollo del proyecto por lo que realizarla al concluir dicho proyecto no cumpliría con su objetivo.

La segunda opción puede funcionar si el proyecto actual es bastante similar a los esfuerzos pasados y si otras influencias del proyecto son similares. Las opciones restantes son métodos viables para la estimación de proyectos de software.

Para poder realizar una estimación utilizando los datos históricos de proyectos ya concluidos, se hace necesario aplicar la técnica de estimación por analogía, ya que es la que permite establecer comparaciones entre proyectos según sus características.

1.2. Herramientas Automáticas de Estimación

Las herramientas automáticas de estimación permiten al planificador estimar costos y esfuerzos, así como llevar a cabo análisis, con importantes variables del proyecto, tales como la fecha de entrega o la selección del personal. Aunque existen muchas herramientas automáticas de estimación, todas exhiben las mismas características generales y todas requieren de una o más clases de datos.

A partir de estos datos, el modelo implementado por la herramienta automática de estimación proporciona estimaciones del esfuerzo requerido para llevar a cabo el proyecto, los costos, la carga de personal, la duración, y en algunos casos la planificación temporal de desarrollo y riesgos asociados.

Además el planificador debe predecir los recursos de hardware y software que va a requerir y el riesgo implicado.

Para obtener estimaciones para un proyecto, generalmente se utilizan al menos dos de las tres técnicas referidas anteriormente. Mediante la comparación y la conciliación de las estimaciones obtenidas con las diferentes técnicas, el planificador puede obtener una estimación más exacta. La estimación del proyecto de software nunca será una ciencia exacta, pero la combinación de buenos datos históricos y técnicas puede mejorar la precisión de la estimación.

1.2.1 Ámbito Internacional

Internacionalmente existen disimiles herramientas que permiten o ayudan a estimar proyectos, a continuación se muestran algunas de estas. (Ver tabla. 1.1.).

Nombre de la Herramienta	Plataforma	Empresa Vendedora	Comentario
20s Reference Estimation	Excel	20smackers www.20smackers.com	Estima en base a proyectos históricos
20s Estimation Calculator	Excel	20smackers www.20smackers.com	Permite calcular esfuerzo y costos en etapas tempranas del desarrollo
Estimacs	Excel	Computer Associates International Inc. www.ca.com/products/stimacs.htm	Permite realizar estimaciones de puntos de función
Costar		Softar systems Inc. www.softarsystems.com	Cocomo

Tabla. 1.1. Herramientas automáticas de estimación.

De las mismas sólo se analizará el 20s Reference Estimation por ser la única que utiliza los datos históricos de proyectos.

20s Reference Estimation (20smackers)

Aplicación privativa perteneciente a la empresa 20smacker, la cual no permite su comercialización con países con los cuales, las leyes de los Estados Unidos lo prohíban Fig. 3. Esta herramienta crea estimaciones bastante precisas de los proyectos futuros mediante una referencia a los proyectos históricos. Especifica los elementos de información que identifican las características de los proyectos por el medio ambiente. Guarda la hora actual y hora original estimada necesaria para la entrega de los proyectos. Referencia los esfuerzos del proyecto anterior para determinar con exactitud las estimaciones para futuros proyectos. Este producto funciona con las versiones de Excel 97, 2000, 2002.

1.2.2. Ámbito Nacional

En la búsqueda realizada no se encontraron herramientas en Cuba, que estimen proyectos a partir de datos históricos. En la UCI, actualmente se está aplicando el Método de Estimación UCI v 3.4 que fue desarrollado en la misma universidad como parte del proceso de mejora, para la estimación de tamaño, costo y esfuerzo requerido para desarrollar un producto de software; y precisamente debido a que no cuenta con una base histórica, para su creación se tuvieron en cuenta los datos dispersos de algunos proyectos y la evaluación de algunos factores que, según los criterios de expertos, pueden influir en las estimaciones del proyecto. Utiliza el Microsoft Office Excel como herramienta para realizar los cálculos pertinentes con la estimación.

1.3. Diseño de la Base de Conocimiento

Para realizar el diseño de la BC, se hace necesario identificar un conjunto de características de los proyectos, para así conformar los casos de proyectos que se guardarán en la misma. Este proceso de caracterización dará como resultados un conjunto de variables que definirán las características a comparar a través de métricas que las evalúen. Además permitirán brindar la información del resultado de la estimación, por lo que se hace necesario identificar dos tipos de variables: Entrada y Salida.

Las variables de entrada son las encargadas de identificar las características de los proyectos que se compararan, como son el tipo de aplicación a desarrollar, tamaño del software a implementar, metodología de desarrollo, tiempo de finalización del proyecto, cantidad de personal involucrado, lenguajes de programación utilizados. Las variables de salida se encargarán de mostrar información referente a los casos de proyectos encontrados, como: Tiempo perdido; aunque puede ocurrir que en ocasiones algunas variables de entrada se comporten como de salida, en dependencia de la información con que se cuente al principio del proyecto. Además de estas, se guardarán en la BC, variables descriptivas como son el nombre del proyecto (aplicación desarrollada).

También las mismas a su vez serán clasificadas según la medición que representen, siendo de dos tipos: variables cualitativas o cuantitativas.

Variables cualitativas: expresan distintas cualidades, características o modalidad. Cada modalidad que representa se denomina atributo o categoría y la medición consiste en una clasificación de dichos atributos. Las variables cualitativas pueden ser ordinales y nominales. Además pueden ser dicotómicas cuando sólo pueden tomar dos valores posibles como sí o no, hombre o mujer, o son politómicas cuando pueden adquirir tres o más valores. Dentro de ellas se puede distinguir:

- **Variable cualitativa ordinal:** La variable puede tomar distintos valores ordenados siguiendo una escala establecida, aunque no es necesario que el intervalo entre mediciones sea uniforme, por ejemplo, *leve, moderado, grave*.
- **Variable cualitativa nominal**: En esta variable los valores no pueden ser sometidos a un criterio de orden como por ejemplo los colores o el lugar de residencia.

Variables cuantitativas: Son las variables que se expresan mediante cantidades numéricas. Las mismas además pueden ser:

- **Variable discreta:** Es la variable que presenta separaciones o interrupciones en la escala de valores que puede tomar. Estas separaciones o interrupciones indican la ausencia de valores entre los distintos valores específicos que la variable pueda asumir. Ejemplo: El número de hijos (1, 2, 3, 4, 5).

➢ **Variable continua:** Es la variable que puede adquirir cualquier valor dentro de un intervalo especificado de valores. Por ejemplo el peso (2,3 kg, 2,4 kg, 2,5 kg,...) o la altura (1,64 m, 1,65 m, 1,66 m,...), que solamente está limitado por la precisión del aparato medidor, en teoría permiten que siempre exista un valor entre dos cualesquiera. (Universidad de Santiago de Chile, Facultad de Matemática y Ciencia de la Computación, 2005)

1.3.1 Tipo de Aplicación (Tarrillo, 2009)

Tipo de variable según la medición que representa: Discreta.

Para caracterizar un proyecto, es importante conocer el tipo de aplicación que se desarrollará, ya sea Web o Escritorio. Para discernir las diferencias que existen en el desarrollo de uno u otro tipo de aplicación es necesario ver su arquitectura, la cual se mostrará a través del siguiente ejemplo:

Aplicación de Escritorio

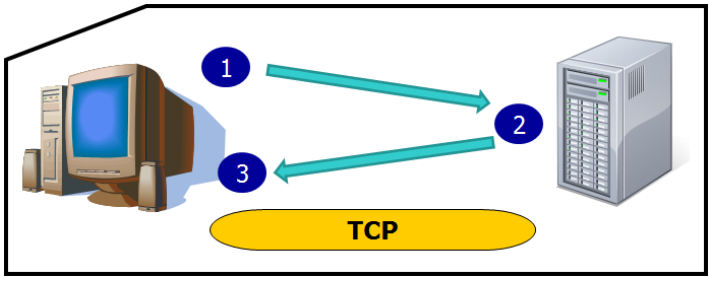

Fig. 1. Arquitectura de una Aplicación de Escritorio.

En una aplicación de escritorio normalmente no se inicia una sesión por cada aplicación que se utilice, sólo se inicia sesión una vez cuando prendemos el sistema operativo, asumiendo que se va a abrir una aplicación para ver una lista de tareas:

1. El usuario carga la aplicación.
2. La aplicación (el código), se conecta a la base de datos y recupera la información del usuario.
3. La aplicación muestra al usuario la información solicitada.

Aplicación Web

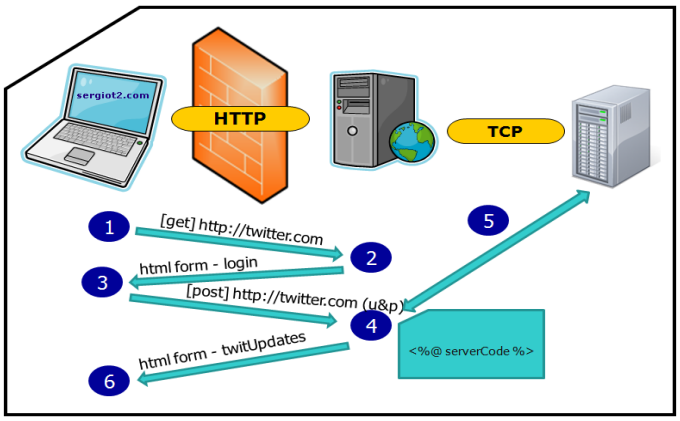

Fig. 2. Arquitectura de una Aplicación Web.

El usuario desde cualquier parte del mundo y desde cualquier dispositivo (PC, laptop, mobile), desea ver donde será el próximo @BeerTwit.

1. El usuario tiene que ingresar la URL de la página en su navegador (*1). El navegador por detrás se encargará de hacer un request (solicitud) al servidor Web usando el protocolo de comunicación HTTP (*2) (internet), y en este caso usará el método GET, por que sólo quiere obtener información.

2. El servidor Web recibe el request y envía un response (sólo html) al navegador. Los navegadores no entienden el código ASP, PHP, o JSP, ellos sólo muestran contenido en html (*3), es por eso que todos los servidores Web después de procesar un request devuelven sólo html (que puede incluir Javascript (*4)), el html generado debe ser un formulario en html, para que el usuario pueda enviar su información. Por otro lado si el usuario ha iniciado sesión con anterioridad es posible que su sesión este activa, y no tenga que iniciar sesión nuevamente.

3. El usuario llena su información, user y password, y hace clic en el famoso botón “Sign in”. El navegador por detrás recolectará esta información, y en este caso que se desea enviar esa información al servidor debe estar usando el método POST. Todos los lenguajes usan POST para enviar información a una página, ya sea ASP.NET, Php, JSP, etc (*5). En el caso especial de ASP.NET cuando están desarrollando por defecto todos los formularios se envían usando POST,

pueden hacer “View Source” de una página en el navegador y verán que el formulario html tiene el método POST. Pueden ver también esto usando la herramienta Fiddler. Con GET también se puede enviar variables, pero no es técnicamente enviar información, es más bien, un obtener información con estos parámetros.

4. El request llega al servidor Web, y se ejecutará el código de servidor Php, Jsp, o ASP, que se conectará con la base para verificar si existe el usuario y si el password coincide con el enviado por el usuario.

5. Si el usuario y el password son validos, el código de servidor (login.php, login.jsp, o login.aspx), redireccionará el request a otra página showUpdates.php, la cual se conecta nuevamente a la base de datos para traer todos los updates de los amigos del usuario, después de procesar la página, el servidor envía el response (sólo html) al usuario.

6. El usuario ve en una página las últimas actualizaciones de sus amigos, y parece que esta semana no habrá @BeerTwit, así que tendrá que inventar alguna excusa para generar uno nuevo.

Analizando el ejemplo anteriormente mostrado, se puede concluir que es determinante conocer el tipo de aplicación a desarrollar, ya que existen especificidades que diferencian ambos tipos de software, por lo que son diferentes en el momento de desarrollarlas.

1.3.2. Metodología de desarrollo

Tipo de variable según la medición que representa: Discreta.

La metodología es otro factor a considerar para caracterizar un proyecto. Esta variable es de gran importancia porque a partir de ella se pude establecer una guía para el desarrollo de la nueva aplicación, a partir de la misma se puede conocer información sobre los artefactos generados o posibles a generar durante el proceso de desarrollo.

1.3.3. Tamaño del software

Tipo de variable según la medición que representa: Cuantitativa.

Teniendo en cuenta que actualmente, cuando los requisitos del software a desarrollar llegan al equipo de proyecto, los analistas calculan los paquetes funcionales para determinar el tamaño del mismo. Se utilizará una versión de las "Métricas de Tamaño" definida en el Método de Estimación UCI v 3.4. (PROGRAMA DE MEJORA UCI, 2010).

Esta a su vez es basada en el método de Puntos de función, la cual plantea que los puntos de función son independientes del lenguaje, herramientas o metodologías utilizadas en la implementación; por ejemplo, no tienen que considerar lenguajes de programación, sistemas de administración de bases de datos, hardware, o cualquier otra tecnología de procesamiento de datos. Segundo, los puntos de función pueden ser estimados a partir de la especificación de requisitos o especificaciones de diseño, haciendo posible de este modo la estimación del esfuerzo de desarrollo en etapas tempranas del mismo. Tercero, como los puntos de función están basados en una visión externa del usuario del sistema, los usuarios no técnicos del software poseen un mejor entendimiento de lo que los puntos de función están midiendo. (Durán Rubio, 2003)

Métricas de Tamaño

El primer paso del método es la identificación de la cantidad de Paquetes Funcionales (PF) que potencialmente tendrá el sistema a desarrollar. Estos PF engloban una serie de funcionalidades y estarán compuestos por una determinada cantidad de puntos de función (pf) que será estimada por el método. Estos PF se clasificarán según la complejidad atendiendo al tamaño. (Ver Tabla 1.2).

Cantidad de Puntos de Función Máximos y Mínimos según clasificación del Paquete Funcional		
Clasificación	**Cota Mínima**	**Cota Máxima**
Pequeño	3	20
Mediano	21	40
Grande	41	70

Tabla 1.2. Cantidad de Puntos de Función según la clasificación de los Paquetes Funcionales.

Para unificar los puntos de función según la clasificación de los paquetes funcionales, se aplicará una métrica donde influye el Factor Cliente (FC) (Epígrafe 1.4.1.). Quedando:

- TPFGpf= TotPFG *CMaxG *FC donde,

  TPFGpf: Tamaño Paquetes Funcionales Grandes (cantidad de punto de función)

  TotPFG: Total de Paquetes Funcionales Grandes.

- TPFMpf = TotPFM *CMaxM *FC donde,

  TPFMpf: Tamaño Paquetes Funcionales Medianos (cantidad de punto de función)

  TotPFM: Total de Paquetes Funcionales Medianos.

- TPFPpf = TotPFP *CMaxP *FC donde,

  TPFPpf: Tamaño Paquetes Funcionales Pequeños (cantidad de puntos de función)

  TotPFP: Total de Paquetes Funcionales Pequeños.

Obtenido el valor del Tamaño para cada tipo de PF se procede a calcular el tamaño total:

- TotalPFA: ∑(TPFGpf + TPFMpf + TPFPpf)

  TotalPFA: Tamaño Potencial del Sistema.

1.3.4. Tiempo de Finalización del Proyecto

Tipo de variable según la medición que representa: Cuantitativa.

Factor que informará el tiempo en que fue terminado o terminará (en dependencia de si el cliente desea el software en un plazo previamente establecido) el proyecto. Además se especificará el tiempo invertido en cada etapa de desarrollo del producto. El mismo se especificará en semanas.

1.3.5. Tiempo por Fases de Desarrollo.

Tipo de variable según la medición que representa: Cuantitativa.

Esta variable le brindará la posibilidad al planificador de tener una visión más real del tiempo de desarrollo del software, y le permitirá realizar una planificación más acertada. El mismo se especificará en semanas.

Para poder guardar esta información en la base de casos (BC) es necesario primeramente realizar un análisis sobre las fases o iteraciones de las metodologías de desarrollo que actualmente son empleadas en para el desarrollo de aplicaciones informáticas en el CESIM, siendo estas RUP y XP.

Los ciclos de vida de un proyecto en XP y en RUP no son exactamente iguales, aunque sin duda tienen bastantes similitudes, ambas son metodologías iterativas con probado éxito en el desarrollo de software.

En un proyecto XP la primera fase es llamada "Ápice arquitectónico" (Architectural Spike) corresponde bastante con la "Incepción" del RUP. El ápice arquitectónico de la XP suele ser mucho más rápida que en RUP, donde la incepción puede tener varias iteraciones, sin embargo, ambas buscan lo mismo: conceptualizar de manera general el proyecto. En esta fase suelen presentarse los primeros estimados y es normal que estos sean muy poco precisos.

La fase de "Plan de entregas" de la XP podría verse como la "Elaboración" del RUP, en ambas se presentan los "Guiones de usuario" (XP) o "Casos de uso" (RUP) y se establecen con más claridad los requerimientos del sistema generales. Una de las mayores diferencias es la documentación asociada a estas fases y el estilo de la misma, sin embargo, los casos de uso y los guiones de usuario son, en esencia, lo mismo, descripciones del comportamiento esperado del sistema ante las acciones de los usuarios o actores externos.

Después de esto comienza propiamente el desarrollo, o la elaboración del código en sí. En RUP se le llama "Construcción", en XP se le considera como el grupo de iteraciones o "Iteración" a secas. Aquí las iteraciones cobran su verdadera importancia y ambas metodologías comienzan cada iteración con guiones de usuario o casos de uso que deberán cumplirse al final de la iteración y con

un trabajo de arquitectura. Cada iteración debe ser corta, en XP suelen ser notablemente más cortas que en RUP, pero en ninguno de los dos casos es conveniente que duren más de dos semanas. Cada una tiene entregables claros definidos en los guiones o casos y al final de ellas se debe siempre re-estimar el proyecto con la intención de hacer más precisos los estimados generales.

Finalmente en XP se debe cumplir las pruebas de aceptación definidos también en los guiones de usuario donde se cotejan los resultados actuales con lo que se esperaba del sistema. En RUP esta fase se contempla dentro de la Construcción y la "Transición", su fase final.

En la Transición del RUP o entrega de XP las diferencias pueden ser mayores, para RUP la entrega final debe ser algo mucho más definido, mientras que en XP se realizan entregas continuas y discretas que permiten evaluar el sistema conforme se colocan las versiones finales. Sin embargo, por el esquema iterativo de ambas, las dos metodologías contemplan entregas parciales después de cada iteración para una evaluación y monitoreo continuo.

Se puede concluir que RUP y XP comparten: (PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE, 2006)

- **Enfoque iterativo**, buscando resolver riesgos lo más temprano posible en el proyecto.
- **Verificación continua de la calidad.** Si bien RUP lo plantea como procesos de testing frecuentes, XP se enfoca en TDD (técnica de desarrollo en la cual primero se definen e implementan los tests que comprobarán cierta funcionalidad, y luego se implementa la funcionalidad misma.).
- **Administración de requisitos.** RUP define un proceso de levantamiento claro y que se repite en las iteraciones, XP se basa en la definición de Historias que se van priorizando y revisando en cada iteración.
- **Modelo de Proceso**, al menos en forma similar. RUP define 4 fases por las cuales se va implementando el proyecto en forma iterativa. En XP este proceso también sigue un proceso evolutivo de fases en las cuales las iteraciones se van acercando al producto final. Durante estas fases, se realizan actividades similares (Ver Tabla 1.3.):

RUP	XP
Análisis de Requerimientos	Definición Historias y Priorización
Diseño	Definición de tareas de desarrollo
Implementación	Implementación
Prueba	Pruebas de aceptación

Tabla 1.3. Fases de desarrollo comunes entre RUP y XP

Analizando lo anteriormente expuesto se pueden definir como las actividades similares durante las fases de desarrollo que todo proceso de desarrollo de software tendrá, independientemente de la metodología empleada, las siguientes:

- Análisis de Requerimientos.
- Diseño.
- Implementación.
- Prueba.

De cada una de estas fases se guardará en la BC, el tiempo de duración expresado en semanas.

1.3.6. Tiempo Perdido

Tipo de variable según la medición que representa: Nominal.

Se especificará la cantidad de tiempo perdido, por causas como fallas de fluido eléctrico, tiempo de los integrantes fuera de la Universidad, roturas del hardware, cursos al equipo de desarrollo, etc. Un factor importante a tener en cuenta, porque en dependencia del mismo, el tiempo real de desarrollo de una aplicación podría acortarse o alargarse. El mismo se especificará en semanas.

1.3.7. Integrantes del proyecto

Tipo de variable según la medición que representa: Cuantitativa.

Factor que mostrará la cantidad de personal involucrado en el proceso de desarrollo del software. Para cuantificar de mejor forma la influencia de los integrantes en el tiempo de conclusión del proyecto, se dividirán a los integrantes del mismo según sus roles para así poder tener una información más cercana a las características organizacionales del proyecto.

Primeramente es necesario analizar las similitudes en cuanto a los roles involucrados en el proceso de desarrollo, que definen tanto RUP como XP.

RUP

Define un total de 30 roles agrupados por participación en actividades relacionadas. Estos grupos son: (Universidad Politécnica de Valencia, Mayo 2007)

Analistas:

- Analista de proceso de negocio.
- Diseñador del negocio.
- Analista del sistema.
- Especificador de requisitos.

Desarrolladores:

- Arquitecto de software.
- Diseñador.
- Diseñador de interfaz de usuario.
- Diseñador de cápsulas.
- Diseñador de base de datos.
- Implementador.
- Integrador.

Gestores:

- Jefe de Proyecto.

- Jefe de control de cambios.
- Jefe de configuración.
- Jefe de pruebas.
- Jefe de despliegue.
- Ingeniero de procesos.
- Revisor de gestión de proyectos.
- Gestor de pruebas.

Apoyo:

- Documentador técnico.
- Administrador de sistema.
- Especialista de herramientas.
- Desarrollador de cursos.
- Artista gráfico.

Especialista en pruebas:

- Especialista en prueba (tester).
- Analista de pruebas.
- Diseñador de Pruebas.

Otros Roles:

- Stakeholders.
- Revisor.
- Coordinación de revisiones.
- Revisor técnico.
- Cualquier rol.

XP

Aunque en otras fuentes de información aparecen algunas variaciones y extensiones de roles XP, en este apartado se describirán los roles de acuerdo con la propuesta original de Kent Beck. (Beck, 1999)

Programador:

El programador escribe las pruebas unitarias y produce el código del sistema. Debe existir una comunicación y coordinación adecuada entre los programadores y otros miembros del equipo.

Cliente:

El cliente escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio. El cliente es sólo uno dentro del proyecto pero puede corresponder a un interlocutor que está representando a varias personas que se verán afectadas por el sistema.

Encargado de Pruebas *(tester)*:

El encargado de pruebas ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

Encargado de seguimiento *(tracker)*:

El encargado de seguimiento proporciona realimentación al equipo en el proceso XP. Su responsabilidad es verificar el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones. También realiza el seguimiento del progreso de cada iteración y evalúa si los objetivos son alcanzables con las restricciones de tiempo y recursos presentes. Determina cuándo es necesario realizar algún cambio para lograr los objetivos de cada iteración.

Entrenador *(coach)*

Es responsable del proceso global. Es necesario que conozca a fondo el proceso XP para proveer guías a los miembros del equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

Consultor:

Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto. Guía al equipo para resolver un problema específico.

Gestor *(Big Boss)*:

Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

Esta metodología permite la combinación de roles en un mismo individuo; en XP, los programadores diseñan, programan y realizan las pruebas, así como el gestor puede combinarse con el tracker. No así en RUP, aunque la misma es adaptable a las necesidades propias del proyecto.

Analizando los roles de ambas metodologías se definieron los siguientes roles:

- **Líder de Proyecto:** Este rol se encarga de establecer las condiciones de trabajo. Por tal motivo tiene la función de dirigir y asignar recursos, coordina las interacciones con los clientes y usuarios finales, planifica las iteraciones, asigna el trabajo, define la organización del proyecto, establece las practicas que aseguran la integridad y calidad de los artefactos del proyecto, entre otras responsabilidades.
- **Desarrollador:** Tiene a su cargo la codificación de los componentes en código fuente en algún lenguaje de programación durante cada iteración; debe elaborar y ejecutar las pruebas unitarias realizadas sobre el código desarrollado; es responsable de las clases que ha desarrollado debiendo documentarlas, actualizarlas ante cambios y mantenerlas bajo el control de configuración de las mismas mediante la herramienta utilizada.
- **Analista:** Se encarga de dirigir el proceso de captura de requerimientos, definir los actores y casos de uso y estructurar el modelo de casos de uso, estableciendo la forma en que funcionara el sistema y cuales son las restricciones del mismo.

- **Probador:** La función del probador es realizar las pruebas identificadas y definidas previamente, utilizando las instrucciones, métodos y herramientas necesarias para este rol. Debido a la realización de las pruebas debe obtener los resultados de las mismas.

En resumen se mostrará el total de integrantes del proyecto, así como su especificación por roles, por lo que se hace necesario la existencia de las siguientes variables:

- Integrantes del proyecto (*expresará el total de integrantes*).
- Líder de Proyecto.
- Cantidad de Analistas (*expresará el total de analistas*).
- Cantidad de Desarrolladores (*expresará el total de desarrolladores*).
- Cantidad de Probadores (*expresará el total de probadores*).

1.3.8. Lenguajes utilizados.

Tipo de variable según la medición que representa: Discreta.

El lenguaje es otro elemento importante a tener en cuenta. Esta variable muestra los lenguajes utilizados durante el desarrollo de una determinada aplicación, y además, dado la existencia de varios casos de proyectos similares en la base de casos, y que estos estén implementados en lenguajes diferentes, establecer una comparación en cuanto a cual fue el más utilizado, y así poder ayudar al líder del proyecto a determinar el lenguaje a emplear en el desarrollo del nuevo software.

Además en el caso de las aplicaciones Web especificar los distintos lenguajes utilizados:

- Lenguaje del lado del cliente.
- Lenguaje dinámico.
- Lenguaje del lado del servidor.

1.4. Factores de Complejidad que influyen en las métricas (PROGRAMA DE MEJORA UCI, 2010)

Los factores de complejidad definidos se listan dándoseles un peso determinado en una escala de 0.5 - 1.5 en dependencia del nivel de importancia. La complejidad se da en un primer momento como una valoración cualitativa de cada factor y luego internamente se cuantifica en:

Evaluación	Valoración
Alto (A)	5
Medio Alto (MA)	4
Medio (M)	3
Medio Bajo (MB)	2
Bajo (B)	1

Tabla 1.4. Escala 0-5.

Evaluación	Valoración
Si	1
No	0

Tabla 1.5. Escala 0-1.

Luego de clasificar la complejidad por cada factor se procede a calcular el resultado final para cada uno, al multiplicar el peso predeterminado para cada factor y el valor equivalente que toma cada valoración cualitativa.

El resultado final será la sumatoria del valor calculado para todos los criterios llevándose a una escala de conveniencia en función de su uso en la fórmula final del método. Los máximos se determinan a partir de configuraciones totalmente positivas o negativas.

1.4.1. Factor Cliente (PROGRAMA DE MEJORA UCI, 2010)

Permite determinar un nivel de incertidumbre asociado a la madurez que tiene el cliente en materia de asimilación de proyectos informáticos, se utilizará para determinar la cantidad de puntos de función que se dejan como reserva.

Factor	Peso	Valoración	P*V
Existe sistema anterior.	0.5	(Si, No)	
Existen resultados de informatización.	1	(Si, No)	
Existe Dirección de Informatización o Informática.	1	(Si, No)	
Existe infraestructura tecnológica en la organización.	1	(Si, No)	
Existe base legal en la organización.	1	(Si, No)	
Es el primer proyecto con la organización	0.5	(Si, No)	
Existe una estructura clara en la organización.	1	(Si, No)	
Existe un especialista para atender al proyecto	1	(Si, No)	
Están definidas las funciones de las áreas.	1.5	(Si, No)	
Estabilidad de los requisitos.	1	(Si, No)	
El cliente es el usuario de la aplicación.	0.5	(Si, No)	

Tabla 4. Factores Individuales.

El Factor Cliente se calcula como: uno más la sumatoria del producto de cada factor individual por su respectivo peso convertido a una escala entre 1 y 1.5 mediante la fórmula

FC = (FO*0.5)/FP

Donde:

FO: factor obtenido

FP: factor obtenido asignando valoraciones pesimistas.

1.5. Razonamiento Basado en Casos (De la Torre V., et al., SF)

Esta técnica de I.A. intenta llegar a la solución de nuevos problemas, de forma similar a como lo hacen los seres humanos. Es una tecnología que representa el conocimiento como una base de datos de casos y soluciones.

Una persona cuando se encuentra ante una dificultad o una situación polémica a resolver, comienza por buscar en su memoria si ya se ha enfrentado a una situación igual o al menos similar, luego establece semejanzas y a partir de ahí identifica las soluciones previas, las analiza y determina las posibles variantes de solución del problema actual. Este proceso las personas lo realizan sin darse cuenta. Al final la solución obtenida no es igual a la anterior, pero cumple dos aspectos muy importantes, el primero, da respuesta al nuevo problema y el segundo, ha enriquecido su experiencia anterior con la nueva solución.

El funcionamiento del RBC parte de estos principios y para ello comprende cinco actividades principales (Fig. 5):

1. Recuperación.
2. Adaptación.
3. Crítica y justificación.
4. Evaluación.
5. Almacenamiento.

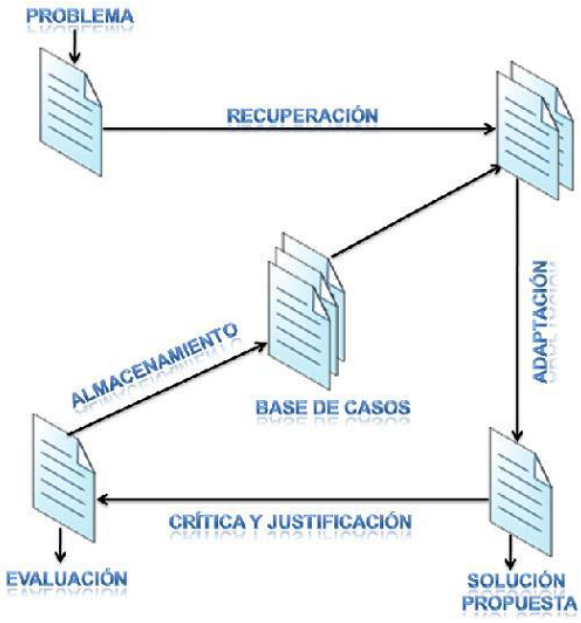

Fig. 5. Ciclo de vida de un Sistema Basado en Casos.

La **recuperación** es la selección, en la base de conocimientos, de aquellos casos cuya descripción se ajusta más a la información presentada en el nuevo caso.

La **adaptación** consiste en adecuar la solución del caso más parecido a las condiciones del nuevo caso. Esto es necesario, dado que normalmente los fenómenos o síntomas que se presentan en un diagnóstico, no son idénticos a los ocurridos en los casos anteriores. Finalmente se propone una solución.

La etapa de **crítica y justificación**, consiste en la validación de la solución propuesta. Esta validación se realiza contrastando diferentes soluciones o simulando la solución para estimar qué tan acertada es. Esta etapa está altamente influenciada por el grado de conocimiento que tiene el experto sobre el fenómeno ocurrido, y es él el que juzga la efectividad de la solución propuesta con base en su experiencia.

En la etapa de **evaluación** se aplica la solución propuesta y se analiza el resultado de su aplicación. Si los resultados son los esperados se confirma la solución, pero si existen diferencias, se debe

averiguar por qué ocurrieron tales diferencias y cómo pueden evitarse. Esta información debe servir para mejorar la definición del caso.

Finalmente, el **almacenamiento** consiste en registrar, en la base de conocimiento, la información derivada del nuevo caso, ya sea como un caso nuevo o un caso mejorado.

La elaboración de un sistema que emplea el RBC presenta dos problemas principales: el primero, saber cómo almacenar la experiencia de tal forma que ésta pueda ser recuperada en forma adecuada, y el segundo, conseguir utilizar la experiencia previa en un problema actual.

La forma de representar y almacenar estas experiencias se realiza a través de casos. Un caso mantiene todos los atributos y características relevantes de un evento pasado. Estas características servirán como índices para la recuperación del caso futuro. De acuerdo a la naturaleza del problema tratado se define la representación del caso, es decir, cuáles son los atributos importantes, qué problemas serán tratados, cuál es la solución propuesta, etc. Además es necesario definir el o los mecanismos de recuperación de casos.

1.5.1. Ventajas del RBC

El enfoque que utilizan los Sistemas Basados en Casos (SBC) para la adquisición del conocimiento es una de las ventajas que se le otorgan a este tipo de sistemas; pues razonan desde episodios específicos, lo cual evita el problema de descomponer el conocimiento del dominio y generalizarlo en reglas. (De la Torre V., et al., SF)

Otras de las ventajas, lo constituyen la flexibilidad para representar el conocimiento a través de casos, la organización de la Base de Casos (BC) y la estrategia de recuperación y adaptación de los casos y que el usuario puede ser capaz de agregar nuevos ejemplos a la BC sin la necesidad de intervención por parte de un experto.

1.5.2. ¿Por qué utilizar el RBC para estimar por analogía?

La figura 6 muestra un esquema general de cómo se corresponden las etapas del ciclo de vida de los Sistemas Basados en Casos (SBC) y el proceso de estimación por analogía.

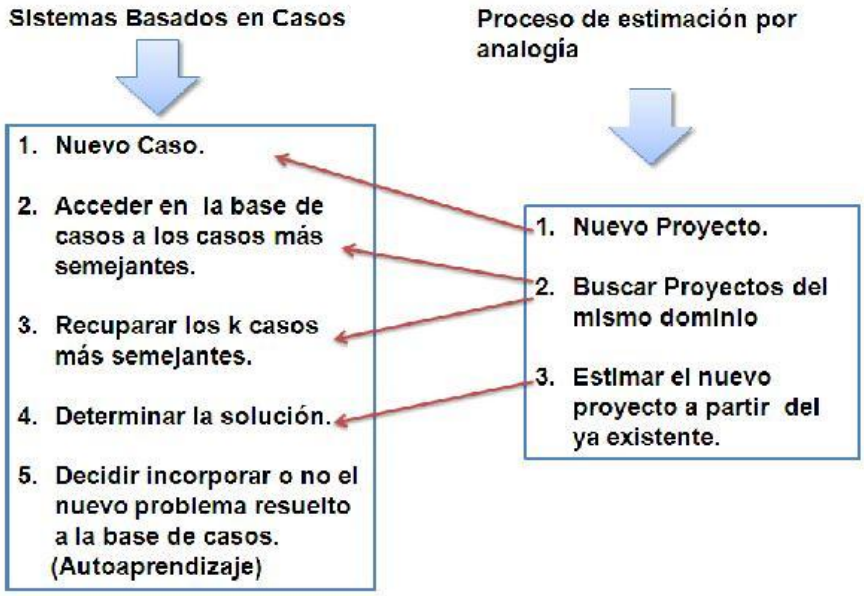

Fig. 6. Correspondencia de las etapas de vida de los SBC y la estimación por analogía.

Existen varias razones que justifican el uso del Razonamiento Basado en Casos en la implementación de una herramienta para estimar proyectos utilizando la técnica de Estimación por Analogía, entre otras se pueden citar:

- ✓ La hipótesis de de que "problemas similares tienen soluciones similares", es común al RBC, y a la estimación por analogía.
- ✓ Es más factible a la hora de determinar cuan semejante es un proyecto a otro.
- ✓ Para posibilitar una mejor interacción entre la herramienta y la base de casos, que contendrá los datos históricos de los proyectos ya realizados.

1.5.3 Técnica de Recuperación de Casos

Las técnicas más comúnmente utilizadas son: vecinos más cercanos y recuperación inductiva. La técnica del **"Vecino más cercano"** consiste en determinar la distancia entre las características del nuevo caso y la de los casos ya existentes, localizando el caso que esté más cerca. (Ver Fig. 7)

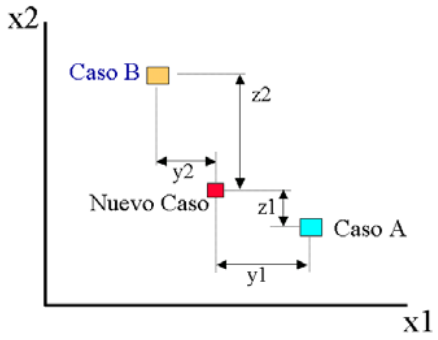

Fig. 7. Diagrama que muestra la distancia entre el nuevo caso y los casos A y B. X1 y x2 son las características que definen a los casos.

Normalmente para destacar una característica, se pondera su distancia, con un peso w_i, para hacer que se acerque o parezca más a uno en particular, como se muestra en la ecuación siguiente:

$$Similitud(T,S) = \sum_{i=1}^{n} f(T_i, S_i) \times w_i$$

Donde f es una función de similitud, T es nuevo caso, S es un caso previo y n es el número de atributos del caso.

1.5.4. Métodos para determinar los pesos de las variables.

Se han propuesto diversos métodos de asignación de pesos, el de la *entropía*, cuyo principal interés reside en su objetividad respecto al decisor, siendo los propios datos del problema los que determinan la importancia relativa de los criterios; los *métodos de asignación directa*, que son aquellos en los que el decisor directamente asigna los pesos. Otra técnica consiste en utilizar un

conjunto de heurísticas que permitan determinar cuáles variables tienen mayor importancia en la determinación del rasgo objetivo, (Redes Neuronales, Algoritmos TDIDT (Top Down Induccion of Decision Trees, Redes Bayesianas). (Cuadrados Rodríguez, et al., 2008)

Debido a que tanto el método de la entropía como la utilización de heurísticas emplean los datos reales para poder establecer los pesos a partir del comportamiento de cada una de las variables, ante los diferentes valores que puede asumir el resto, y teniendo en cuenta que en la actualidad no se cuentan con datos reales en la base de casos se hace necesario utilizar el método de asignación directa, por lo que se determina asignar un peso: $w_i = 5$, siendo este la media del rango de valores que pueden tomar los pesos (0 - 10), exceptuando las siguiente variables que su peso será $w_i = 0$, debido a que no revisten ningún tipo de importancia porque nunca son utilizadas para buscar un caso similar:

- Tiempo perdido.
- Nombre del proyecto.

1.5.5. Motores RBC

Los motores RBC, como su nombre lo indica utilizan el Razonamiento Basado en Casos como técnica para encontrar soluciones para los problema, rehusando conocimiento en forma de las experiencias previamente almacenadas.

FreeCBR 1.1.4

Desarrollado por Lars Johanson en el lenguaje Java bajo la Licencia de Dominio Público. El FreeCBR 1.1.4 puede ser considerado como una aplicación GUI(Graphical User Interface), como una utilidad de línea de comandos, una aplicación web y una API Java (interfaz de programación de aplicaciones provista por los creadores del lenguaje Java, y que da a los programadores los medios para desarrollar aplicaciones Java, como el lenguaje Java es un Lenguaje Orientado a Objetos, la API de Java provee de un conjunto de clases utilitarias para efectuar toda clase de tareas necesarias dentro de un programa.) para generar Razonamiento Basado en Casos, en otras palabras es un

motor de búsqueda RBC que puede ser integrado a cualquier aplicación Java tanto escritorio, consola o web.

1.6. Tecnología y herramientas

1.6.1. Lenguaje a utilizar

Java es un nuevo lenguaje de programación orientado a objetos desarrollado por Sun Microsystems, Sun describe al lenguaje Java de la siguiente manera: simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutral, portable, de alto rendimiento, multitarea y dinámico.

Características de Java (Javier García de Jalón, 2000)

Simple

Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. C++ es un lenguaje que adolece de falta de seguridad, pero C y C++ son lenguajes más difundidos, por ello Java se diseñó para ser parecido a C++ y así facilitar un rápido y fácil aprendizaje.

Orientado a objetos

Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo.

Distribuido

Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales.

Robusto

Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria. Java proporciona:

- Comprobación de punteros.
- Comprobación de límites de arrays.
- Excepciones.
- Verificación de byte-codes.

Arquitectura neutral

Para establecer Java como parte integral de la red, el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (run-time) puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado.

1.6.2. Entornos de Desarrollos

1.6.2.1. Eclipse IDE

Eclipse es un Entorno de Desarrollo Integrado (IDE) desarrollado por IBM y que se distribuye mediante una licencia de código abierto, la EPL o Eclipse Public License.

En un primer momento Eclipse era solo un IDE para Java, pero mediante un sistema de plugins flexible se ha convertido en un IDE genérico soportado por la comunidad que en la actualidad se puede utilizar para desarrollar en C/C++, Java, Python, Groovy, Perl y muchos otros lenguajes soportando la plataforma de desarrollo J2SE y J2EE.

La plataforma Eclipse, cuando se combina con el JDT, ofrece muchas de las características de un IDE de calidad comercial como lo son: un editor de sintaxis que destaca, la compilación de código incremental, un depurador a nivel de fuente thread-aware, un navegador de clases, un archivo jefe de proyecto, y las interfaces con los sistemas estándar de control de código fuente, como CVS y ClearCase.

Eclipse también incluye una serie de características únicas, como la refactorización de código, las actualizaciones automáticas de códigos (a través del Administrador de actualización), una lista de tareas, el apoyo para las pruebas unitarias con JUnit, e integración con la herramienta de construcción Ant Yakarta. La característica más interesante de Eclipse es que el mismo es la plataforma.

1.6.2.2. NetBeans

¿Qué es NetBeans?

NetBeans IDE es una aplicación de código abierto ("open source") catalogada como un IDE (Entorno de Desarrollo Integrado) y un GUI o lo que en inglés se dice: Graphical User Interface (Interfaz Gráfica del Usuario) diseñada para el desarrollo de aplicaciones fácilmente portables entre las distintas plataformas.

El propósito de NetBeans es que podamos trabajar con JAVA y a partir de la versión 6.7, también con JAVA FX lo último que Sun implementó para poder hacer frente a la propuesta de Adobe Air y Silverlight de Microsoft.

Características de NetBeans

NetBeans IDE dispone de soporte para crear interfaces gráficas de forma visual, desarrollo de aplicaciones web, control de versiones, colaboración entre varias personas, creación de aplicaciones compatibles con teléfonos móviles, resaltado de sintaxis y funcionalidades ampliables mediante la instalación de packs.

Cuenta con una inmensa comunidad de programadores y colaboradores (en especial con referencia a la traducción del programa a más de 50 idiomas). NetBeans es gratuito y muy sencillo de seguir, gracias a un menú desplegable muy completo, que agrupa las opciones de manera muy adecuada.

La interfaz gráfica que posee NetBeans es muy completa. El escritorio de trabajo es muy organizado, ya que divide el entorno de desarrollo en marcos o mini ventanas que pueden ajustar su tamaño o incluso ocultarlas para que haya más espacio y claridad.

Cuenta con un editor multicódigo, multiplataforma y autocompletado de código. Otra ventaja de NetBeans es que cuenta con acceso a la base de datos MySQL de manera nativa. Tiene un módulo que permite configurar la conexión con MySQL y los permisos a los usuarios.

Desde el entorno de NetBeans se puede trabajar con grandes proyectos fraccionándolos en otros más pequeños ya que al final se pueden compilar todos en un solo paquete. Contiene un diseñador de UML muy eficaz.

1.6.3. Metodologías de Desarrollo de Software

Actualmente el desarrollo de software ha alcanzado un alto nivel debido a la competencia que existe, por lo que los desarrolladores se han visto en la necesidad de buscar técnicas mediante las cuales se logren estandarizar el trabajo de las aplicaciones que se desarrollan.

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos de software, se les puede llamar como un plan de contingencias en el cual se va indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado, indicando además quienes deben participar en el desarrollo de las actividades y qué papel deben de tener, también se detalla la información que se debe producir

como resultado de una actividad y la información necesaria para comenzarla. Existen dos tipos de metodologías de desarrollo de software a las cuales se les denominan metodologías ágiles y metodologías tradicionales o robustas.

1.6.3.1. Metodologías Tradicionales o Robustas.

Proceso Unificado Rational (RUP) (Universidad Politécnica de Valencia, Mayo 2007)

Es la metodología de desarrollo de software más utilizada para los sistemas basados en objetos y está basado íntegramente en el Lenguaje Unificado de Modelado (UML). RUP es un proceso de desarrollo de software, pero más que un simple proceso es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyectos. El Proceso Unificado Rational está basado en componentes por lo que el software en construcción esta formado por componentes de software y estos están interconectados por interfaces bien definidas, sin embargo, sus principales aspectos se definen como dirigido por casos de uso, centrado en la arquitectura e iterativo incremental.

RUP esta compuesto por 4 fases de desarrollo y 9 flujos de trabajo (6 de ingeniería y 3 de apoyo).

Flujos:

1. Modelado del Negocio
2. Requerimientos
3. Análisis y Diseño
4. Implementación
5. Prueba
6. Despliegue
7. Gestión de la Configuración
8. Gestión de Proyecto
9. Ambiente.

Fases:

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

En la fase de Inicio se describe el negocio, se delimita el proyecto describiendo sus alcances con la determinación de los casos de uso del sistema y se identifican los riesgos. Luego en la Elaboración se hace un plan de proyecto, se completan los casos de uso y se eliminan los riesgos. La fase de Construcción se concentra en la elaboración**,** obteniéndose un producto bien documentado listo para su utilización. En un final en la Transición el release ya está listo para su instalación en las condiciones reales y también se puede implicar reparación de errores.

Características de RUP que la hacen aspirar a tener las mejores prácticas actuales en Ingeniería de Software:

- Forma disciplinada de asignar tareas y responsabilidades.
- Desarrollo iterativo.
- Administración de requisitos.
- Uso de arquitectura basada en componentes.
- Control de cambios.
- Modelado visual del software.
- Verificación de la calidad del software.

1.6.3.2. Metodologías Ágiles

El término "ágil" aplicado al desarrollo del software nace en febrero de 2001 en Utah-EEUU en una reunión donde participan 17 expertos de la industria del software incluyendo creadores e impulsores de metodologías de software. Tras esta reunión se creó The Agile Alliance, una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos.

Las metodologías ágiles se basan en los valores y principios que deberían de permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto, son una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas. Las metodologías ágiles son adecuadas para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico por lo que ya están siendo utilizadas con éxito en proyectos reales, pero les falta una mayor difusión y reconocimiento. (Crystal Methodologies, 2008)

Ejemplos:

- XP
- SCRUM
- DSDM (Dynamic Systems Development Method)
- Crystal Methodologies

Xtreme Programing (XP)

La metodología Xtreme Programing o Programación Extrema fue creada por Kent Beck basado en la suposición de que es posible desarrollar software de gran calidad a pesar de las consecuencias del cambio continuo, es una de variantes de las metodologías ágiles más destacadas y con más aceptación en la comunidad internacional de desarrollo. Una de las herramientas más importantes en la metodología XP es el desarrollo orientado a pruebas, que utiliza las pruebas unitarias como eje de todo desarrollo. Las interacciones suelen ser muy cortas y se promueve a los programadores a buscar soluciones y experiencia con ellas, programar sin miedo a descomponer el sistema.

Características:

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantivo del proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.

- El cliente o el usuario se convierten en miembro del equipo.

Valores que promueve XP:

- **Comunicación:** XP pone en comunicación directa y continua a clientes y desarrolladores. El cliente se integra en el equipo para establecer prioridades y resolver dudas. De esta forma ve el avance día a día, y es posible ajustar la agenda y las funcionalidades de forma consecuente.
- **Feedback rápido y continuo:** Una metodología basada en el desarrollo incremental de pequeñas partes, con entregas y pruebas frecuentes y continuas, proporciona un flujo de retro-información valioso para detectar los problemas o desviaciones. De esta forma los fallos se localizan muy pronto, pues se detectan los errores de planificación que solo se evidencian durante el desarrollo del sistema. La retro-información constituye la herramienta que permite reajustar la agenda y los planes.
- **Simplicidad:** La simplicidad consiste en desarrollar sólo el sistema que realmente se necesita e implica resolver en cada momento sólo las necesidades actuales. Con este principio de simplicidad, junto con la comunicación y el feedback resulta más fácil conocer las necesidades reales.

XP no es un modelo de procesos ni un marco de trabajo, sino un conjunto de 12 prácticas que se complementan unas a otras y que deben implementarse en un entorno de desarrollo cuya cultura se base en los cuatro valores mencionados anteriormente. (Letelier, 2006)

1.6.4. UML (Lenguaje Unificado de Modelado)

Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir. Capta la información sobre la estructura estática y el comportamiento dinámico de un sistema, no es un lenguaje de programación.

De todos los lenguajes de modelados definidos para el diseño Orientado a Objetos, UML es el más expresivo, no garantiza el éxito de los proyectos pero si mejora sustancialmente el desarrollo de los

mismos, al permitir una nueva y fuerte integración entre las herramientas, los procesos y los dominios, permitiendo realizar con claridad nueve diagramas en los cuales se modelan el sistema:

- Diagramas de Casos de Uso para modelar los procesos 'business'.
- Diagramas de Secuencia para modelar el paso de mensajes entre objetos.
- Diagramas de Colaboración para modelar interacciones entre objetos.
- Diagramas de Estado para modelar el comportamiento de los objetos en el sistema.
- Diagramas de Actividad para modelar el comportamiento de los Casos de Uso, objetos u operaciones.
- Diagramas de Clases para modelar la estructura estática de las clases en el sistema.
- Diagramas de Objetos para modelar la estructura estática de los objetos en el sistema.
- Diagramas de Componentes para modelar componentes.
- Diagramas de Implementación para modelar la distribución del sistema. (Corp, Neuron., 2006)

Objetivos del UML:

- Es un lenguaje de modelado de propósito general que pueden usar todos los modeladores. No tiene propietario y está basado en el común acuerdo de gran parte de la comunidad informática.
- No pretende ser un método de desarrollo completo. No incluye un proceso de desarrollo paso a paso, incluye todos los conceptos que se consideran necesarios para utilizar un proceso moderno iterativo, basado en construir una sólida arquitectura para resolver requisitos dirigidos por casos de uso.
- Ser tan simple como sea posible pero manteniendo la capacidad de modelar toda la gama de sistemas que se necesita construir. Necesita ser lo suficientemente expresivo para manejar todos los conceptos que se originan en un sistema moderno, tales como la concurrencia y distribución, así como también los mecanismos de la ingeniería de software, como son la encapsulación y componentes.
- Ser un lenguaje universal, como cualquier lenguaje de propósito general.
- Imponer un estándar mundial. (Vico.org., 2002)

1.6.5. Herramientas CASE (Computer – Aided Software Engineering)

Visual Paradigm

Herramienta profesional que utiliza UML para el modelado, es la opción por excelencia para ser utilizada en un ambiente de software libre y permite crear diferentes tipos de diagramas en un ambiente totalmente visual. Permite dibujar todos los tipos de diagramas de clases, generar código a partir de diagramas, generar documentación y admite código inverso. La herramienta CASE UML también proporciona abundantes tutoriales de UML, demostraciones UML interactivas y proyectos de UML. Soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue.

El software de modelado UML ayuda a una más rápida construcción de aplicaciones de mayor calidad y a un menor coste. Se centra en cómo los componentes del sistema interactúan entre ellos, sin entrar en detalles excesivos, además, permite ver las relaciones entre los componentes del diseño y mejora la comunicación entre los miembros del equipo usando un lenguaje gráfico. Tiene disponible distintas versiones: Enterprise, Professional, Standard, Modeler, Personal y Community.

Sus características más significativas son: licencia gratuita, posee varios idiomas, sus ediciones son compatibles y fácil manejo de la herramienta. Además es una herramienta colaborativa, pues soporta múltiples usuarios trabajando sobre el mismo proyecto. La documentación del proyecto puede ser generada automáticamente en varios formatos (Web o .Pdf), y permite control de versiones. (Freedownloadmanager.org, 2004)

Rational Rose

Rational Rose es la herramienta CASE que comercializan los desarrolladores de UML y que soporta de forma completa la especificación del UML 1.1. Esta herramienta propone la utilización de cuatro tipos de modelo para realizar un diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema, uno lógico y otro físico. Permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software.

Características:

- Desarrollo Iterativo
- Trabajo en Grupo
- Generador de Código
- Ingeniería Inversa

Desarrollo Iterativo: Utiliza un proceso de desarrollo iterativo controlado, donde el desarrollo se lleva a cabo en una secuencia de iteraciones. Cuando la implementación pasa todas las pruebas que se determinan en el proceso, ésta se revisa y se añaden los elementos modificados al modelo de análisis y diseño. Una vez que la actualización del modelo se ha modificado, se realiza la siguiente iteración.

Generador de Código: Se puede generar código en distintos lenguajes de programación a partir de un diseño en UML.

Ingeniería Inversa: Proporciona mecanismos para realizar la denominada Ingeniería Inversa, a partir del código de un programa, se puede obtener su diseño.

Trabajo en Grupo: Permite varias personas trabajando a la vez en el proceso iterativo controlado, para ello posibilita que cada desarrollador opere en un espacio de trabajo privado que contiene el modelo completo y tenga un control exclusivo sobre la propagación de los cambios en ese espacio de trabajo. (De Nobrega, 2005)

1.7. Selección de la Tecnología y Herramientas a Utilizar

Para el desarrollo de la aplicación de estimación por analogía se decide utilizar el motor de búsqueda de razonamiento basado en casos FreeCBR 1.1.4, ya que el mismo es software libre y permite realizar una búsqueda independiente de la cantidad de información que se tenga inicialmente, en otras palabras, permite elegir los atributos del caso por los cuales se va a realizar la búsqueda en la base de conocimientos.

Por ende, para lograr un buen acoplamiento con el motor de búsqueda el lenguaje de programación seleccionado es Java y se va a trabajar en el entorno de desarrollo Netbeans, ya que cuenta con

una interfaz gráfica muy completa, auto-completamiento de código y ofrece un marco de trabajo organizado muy fácil de aprender para desarrolladores principiantes en Java. El sistema estará guiado por la metodología de desarrollo RUP, la cual constituye una de las metodologías estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos, y además está basada en el lenguaje de modelado UML, que permite modelar aplicaciones orientadas a objetos (Universidad Politécnica de Valencia, Mayo 2007).

Como herramienta CASE se escoge a Visual Paradigm por su amplio entorno gráfico. Con el estudio de las tecnologías actuales se seleccionaron las herramientas más adecuadas para la implementación del sistema y bajo un ambiente completamente libre.

Como resultado del análisis realizado durante el presente capítulo se concluye que: las herramientas estudiadas relacionadas con el proceso estimación de proyectos no responden a las necesidades que presenta el CESIM. Debido a esto se hace necesario implementar un sistema informático para la estimación de proyectos, basándose en los datos históricos de los mismos. Para ello se realizó un estudio de las tecnologías actuales y se seleccionaron las más adecuadas para la implementación del sistema.

Capítulo 2

Características del Sistema

En el presente capítulo se tratan los aspectos fundamentales relacionados con el objeto de estudio. Se detallan y explican los procesos del negocio sobre los que se basa la organización para su funcionamiento, se describe el flujo actual de los procesos y se realiza un análisis de cómo estos se efectúan en la actualidad.

Además se presenta el objeto de automatización. Se aborda el tema relacionado con la modelación del negocio, siendo este el primer flujo de trabajo durante el proceso de desarrollo de un software, el cual tiene entre sus objetivos comprender los procesos de negocio de la organización, identificar las mejoras potenciales y señalar puntualmente los requerimientos del sistema que va a soportar la organización. Se muestra la especificación de los requisitos de software (los requisitos funcionales y no funcionales), además de la descripción de los actores y casos de uso resultantes del flujo de trabajo de Requisitos.

2.1. Análisis crítico del proceso actual de la situación problémica

El proceso de gestión de proyectos es una tarea fundamental que se lleva a cabo en el CESIM, con el objetivo de lograr el éxito del mismo. Dentro de la gestión, la estimación juega un rol importante, debido a que en esta etapa, se brinda valores aproximados de los recursos necesarios para el desarrollo del proyecto.

Actualmente no se guardan en el centro ningún tipo de información referente a los proyectos realizados, imposibilitando la realización de una estimación a partir de los datos históricos referente a estos. Además que los métodos de estimación que se utilizan son los tradicionales, en los cuales hay que realizar grandes cantidades de cálculos matemáticos, que pueden ofrecer valores erróneos

si existiera un descuido por parte del planificador del proyecto. Unido a esta situación en el centro no existe un mecanismo, ni una herramienta capaz de realizar este proceso, a partir de los datos que se puedan obtener de proyectos previos.

Con el uso de la herramienta que se propone en este trabajo, se brinda la posibilidad de crear una base de conocimiento que contenga todos los datos necesarios de un proyecto para efectuar una estimación de un proyecto posterior a partir de estos.

2.2. Objeto de Automatización

EL sistema propuesto ha sido concebido con el fin de mejorar el proceso de Estimación que se realiza en el Centro de Informática Médica. La aplicación permitirá a los planificadores de proyectos, realizar dicha tarea utilizando los datos históricos guardados en la Base de Conocimiento de la herramienta.

El Planificador tendrá una mejor visión de los recursos con que cuenta, a partir de la graficación de los resultados obtenidos, logrando así una mejor distribución de los recursos con que se cuenta para el desarrollo del proyecto.

Además la herramienta generará un reporte, permitiéndole tanto, al Jefe de Proyecto como al Planificador ver el resultado del proceso.

2.3. Modelo de Dominio

El modelo de dominio se realiza si no se determinan los procesos de negocio con fronteras bien establecidas, donde se logren ver claramente: quiénes son las personas que realizan cada proceso de negocio, quiénes son los beneficiados con cada uno de estos procesos y quiénes son las personas que desarrollan las actividades en cada uno de estos procesos. Nos permite de manera visual mostrar al usuario los conceptos fundamentales que se manejan en el dominio del sistema en desarrollo. Se representa en UML con un diagrama de clases donde se muestra:

- Conceptos u objetos del dominio del problema: clases conceptuales
- Asociaciones entre clases conceptuales
- Atributos de las clases conceptuales

En este caso durante el desarrollo de la herramienta no se pudo constatar procesos bien definidos en el entorno del negocio. Se hizo difícil determinar los elementos más importantes del sistema y sus interconexiones, así como el establecimiento de las reglas de funcionamiento. Sin embargo se pueden identificar personas, eventos, transacciones y objetos involucrados en ese entorno que no está bien delimitado, por lo que se hizo necesario un modelado del domino perteneciente a la solución.

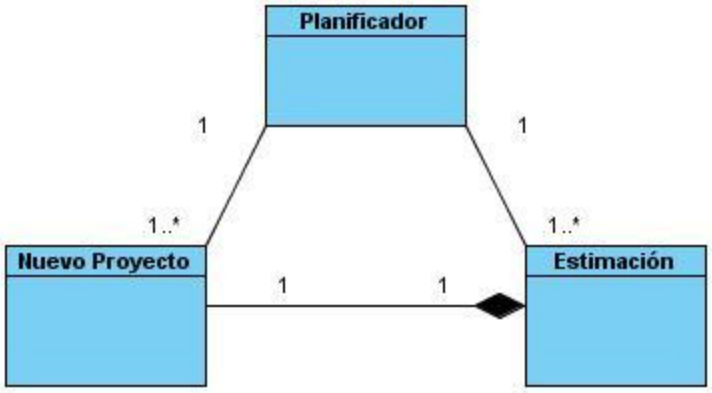

Fig. 8. Modelo de Dominio.

En la anterior figura (Fig. 8.) se muestran tres clases, siendo estas Nuevo Proyecto, Planificador, Estimación. Describiéndose las siguientes relaciones:

Planificador - Nuevo Proyecto (1– 1...*): un planificador recibe uno o muchos proyectos.

Planificador – Estimación (1– 1...*): un planificador realiza una o muchas estimaciones.

Nuevo Proyecto – Estimación (1-1): cada proyecto tiene una estimación.

2.4. Especificaciones de los Requisitos del Software

2.4.1. ¿Qué es un requerimiento?

La lEEE Standard Glosary of Software Terminology define un requerimiento como:

1. Condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo.
2. Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar u otro documento impuesto formalmente.
3. Una representación documentada de una condición o capacidad como en las definiciones 1 ó 2.

Los requerimientos deben ser:

- Especificados por escrito. Como todo contrato o acuerdo entre dos partes.
- Posibles de probar o verificar: si un requerimiento no se puede comprobar, entonces ¿cómo sabemos si cumplimos con él o no?
- Descritos como una característica del sistema a entregar: esto es: que es lo que el sistema debe hacer (y no como debe hacerlo)
- Lo más abstracto y conciso posible: para evitar malas interpretaciones.

2.4.2. Requerimientos Funcionales

Constituyen capacidades o condiciones que el sistema debe cumplir, describen como el sistema debe trabajar. No alteran la funcionalidad del producto, esto quiere decir que se mantienen invariables sin importarles con que propiedades o cualidades se relacionen. (PRESSMAN, 2002)

Listado de requisitos funcionales:

- RF1- Estimar Proyecto.
- RF2- Insertar nuevo Caso de Proyecto.

- RF3- Modificar Caso de Proyecto.
- RF4- Buscar Caso de Proyecto.
- RF5- Graficar Proyecto.

2.4.3. Requerimientos no Funcionales

Son propiedades o cualidades que el producto debe tener. Debe pensarse en ellos como las características que hacen al software atractivo, usable, rápido o confiable. Forman una parte significativa de la especificación. Son importantes para que clientes y usuarios puedan valorar las características no funcionales del producto. (PRESSMAN, 2002)

Listado de Requisitos No Funcionales:

Interfaz Externa:

RNF1- Herramienta Sencilla

RNF1.1- Amigable al usuario y fácil de usar.

RNF1.2- Funcionalidades explícitas.

RNF2- Colores acorde a fines con la aplicación.

Usabilidad:

RF3- Sistema orientado a personas con conocimientos de la estimación de proyectos.

RNF4- La Interfaz es de un flujo sencillo.

Soporte:

RF5- Consta con la documentación necesaria para el aprendizaje sobre el uso de la aplicación.

Portabilidad:

RF6- Máquina Virtual de Java.

Seguridad:

RF7- En caso de algún error la información debe ser guardada de forma que las pérdidas sean mínimas.

Hardware:

RNF8- Rendimiento mínimo del hardware

Procesador Pentium.

512 MB de memoria RAM.

2.5. Diagrama de Casos de Uso del sistema

Este diagrama (Fig. 9) representa gráficamente los procesos y su interacción con los actores.

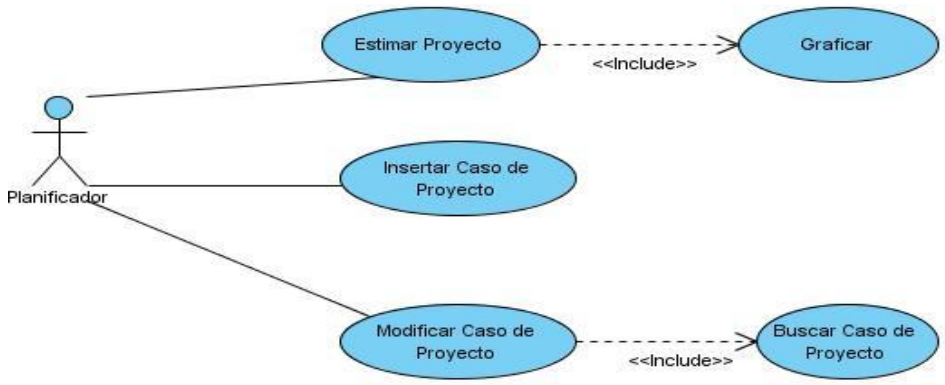

Fig. 9. Diagrama de Casos de Uso del Sistema.

2.5.1. Definición de los Actores

Un actor del sistema es una persona o la abstracción de software que interactúa con el sistema: puede intercambiar información con el, representar el rol que juegan una o varias personas y ser un recipiente pasivo de información. (PRESSMAN, 2002)

A continuación se define y describen los actores del sistema:

Actores del Sistema	Descripción
Planificador	Encargado de realizar la estimación de un proyecto nuevo, y de añadir un nuevo caso a la base de conocimientos de la herramienta.

2.5.2. Especificación de Casos de Uso

CUS 1	Estimar Proyecto
Actor	Planificador
Referencia	RF1
Descripción	Comienza cuando el planificador desea estimar un nuevo proyecto.

CUS 2	Insertar Nuevo Proyecto
Actor	Planificador
Referencia	RF2
Descripción	Comienza cuando el Planificador desea introducir un nuevo proyecto a la base de conocimientos.

CUS 3	Modificar Proyecto
Actor	Planificador
Referencia	RF3
Descripción	Comienza cuando el Planificador desea modificar los datos de un proyecto.

CUS 3	Buscar Proyecto
Actor	Planificador
Referencia	RF4
Descripción	Se inicia cuando el Planificador desea modificar en Caso de Proyecto

CUS 4	Graficar Proyecto
Actor	Planificador
Referencia	RF5
Descripción	Se inicia cuando el Planificador desea ver todos los datos correspondientes de una estimación de un proyecto.

2.5.3. Casos de Uso expandidos

Caso de Uso	Estimar Proyecto
Propósito	Estimar el nuevo proyecto
Actores: Usuario	
Resumen: EL Caso de Uso se inicializa cuando el usuario decide estimar un proyecto, para ello ingresa los datos necesarios para realizar la actividad, el sistema busca un caso semejante en la Base de Casos y devuelve sus datos, brindándole lo opción de ver los detalles del caso encontrado. El Caso de Uso finaliza cuando el sistema le brinda todos los datos pedidos por el usuario.	
Referencias	**RF 1**
Acción del actor	**Respuesta del sistema**

	1. Muestra la vista para la realización de la estimación: Tamaño del software : ○ Paquetes pequeños. ○ Paquetes medianos. ○ Paquetes grandes. Elementos para determinar el factor cliente: ○ Existe sistema anterior. ○ Existen resultados de informatización. ○ Existe dirección de Informatización o Informática. ○ Existe infraestructura tecnológica en la organización. ○ Existe base legal en la organización. ○ Es el primer proyecto con la organización. ○ Existe una estructura clara en la organización. ○ Existe un especialista para atender al proyecto. ○ Están definidas las funciones de las áreas. ○ Estabilidad de los requisitos. ○ El cliente es el usuario de la aplicación. Tecnología: ○ Metodología desarrollo. ○ Entorno desarrollo.

	○ Gestor base datos. ○ Tipo aplicación. ○ Lenguaje del lado del cliente. ○ Lenguaje dinámico. ○ Lenguaje del lado del servidor. ○ Servidor web. ○ Lenguaje de escritorio. Duraciones: ○ Tiempo total. Personal y hardware. ○ Total de integrantes del proyecto. ○ Líder del proyecto. ○ Cantidad de analistas. ○ Cantidad de desarrolladores. ○ Cantidad de probadores. ○ Cantidad de computadoras. ○ Cantidad de servidores. ○ Otros hardware..
2. El usuario introduce los datos. Oprime el botón “Estimar”.	3. Verifica que todos los datos hayan sido proporcionados. 4. Realiza la estimación buscando el caso en al base de casos. 5. Muestra una pantalla con los resultados. Y la opción “Ver detalle”.

Flujo Alterno 1	
Acción del actor	**Respuesta del sistema**
3. Limpiar.	4. Limpia todos los formularios.
Flujo Alterno 2	
Acción del actor	**Respuesta del sistema**
	4. Se muestra un mensaje informando que faltan datos.
5. Acepta el mensaje.	
Flujo Alterno 3	
	5. Muestra un mensaje de error informando de un error en la estimación.
6. Acepta el mensaje.	7. Permite volver a realizar la operación.
Flujo Alterno 4	
	8. Ver Caso de Uso Graficar proyecto

En este capítulo se realizó un análisis crítico de los procesos actuales del negocio. Este permitió precisar el estado actual del Proceso de Estimación del CESIM; con respecto a las actividades para asegurar la calidad del mismo. De la información obtenida, se definieron las necesidades de funcionamiento de la aplicación a implementar.

Capítulo 3

Análisis y diseño del sistema

Este capítulo tiene como objetivo definir la concepción general del diseño del sistema propuesto y cómo se implementa este. Se presentan los diagramas de clases que detallan la interacción de las distintas interfaces de usuario con la clases del sistema y se describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo.

3.1. Flujo de Trabajo de Análisis y Diseño

El objetivo principal de esta disciplina es transformar los requerimientos a una especificación que describa cómo implementar el sistema. El análisis fundamentalmente consiste en obtener una visión de lo que hace el sistema de software a desarrollar, por lo cual este se interesa en los requerimientos funcionales. El diseño es un refinamiento que toma en cuenta los requerimientos no funcionales, por lo cual se centra en como el sistema cumple sus objetivos.

El propósito de esta disciplina es:

- Transformar los requerimientos en un diseño de lo que será el sistema.
- Evolucionar una arquitectura robusta para el sistema.
- Adaptar el diseño para que sea consistente con el entorno de implementación.

Al principio de la fase de elaboración hay que definir una arquitectura candidata, crear un esquema inicial de la arquitectura del sistema, identificar clases de análisis y actualizar las realizaciones de los Casos de Uso con las interacciones de las clases de análisis. Durante la fase de elaboración se va

refinando esta arquitectura hasta llegar a su forma definitiva. En cada iteración hay que analizar el comportamiento para diseñar componentes.

3.2. Análisis

3.2.1. Diagrama de Clases del Análisis

Los diagramas de clases del análisis representan la relación entre las clases, las cuales se centran en los requisitos funcionales, tienen atributos y entre ellas se establecen relaciones de asociación, agregación/composición, generalización/especialización y tipos asociativos. RUP propone clasificar a las clases en: interfaz, controladora y entidad. Las clases interfaz modelan la interacción entre el sistema y sus actores. Las controladoras coordinan la realización de uno o unos pocos casos de uso coordinando las actividades de los objetos que implementan sus funcionalidades. Las clases entidad modelan información que posee larga vida y que es a menudo persistente.

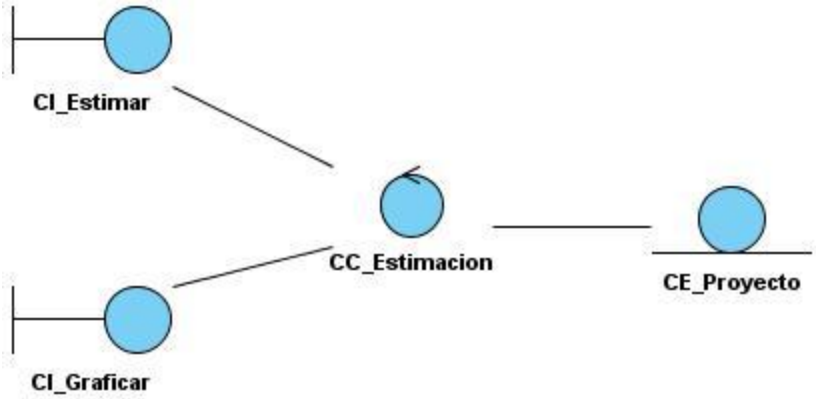

Fig. 10. Diagrama de clases análisis CU Estimar.

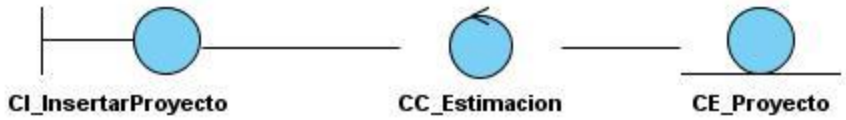

Fig. 11. Diagrama de clases análisis CU Insertar Nuevo Proyecto.

Fig. 12. Diagrama de clases análisis CU Modificar Proyecto.

3.3. Diseño

3.3.1. Principios de Diseño

El Diseño es el antecesor a la implementación del sistema, mediante el Modelo de Diseño se le da cumplimiento a los requerimientos del sistema sin tener un prototipo funcional, además define e identifica las consecuencias del ambiente de implementación; es una guía para la fase de implementación.

3.3.2. Definición de elementos del Diseño

El Modelo Vista Controlador es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Sin lugar a duda ha tenido un gran auge en el desarrollando aplicaciones web.

Modelo: Es la representación específica de la información con la cual el sistema opera. En resumen, el modelo se limita a lo relativo de la vista y su controlador facilitando las presentaciones visuales complejas. El sistema también puede operar con más datos no relativos a la presentación.

Vista: Presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

Controlador: Este responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y, probablemente, a la vista. (Froufe, 2005)

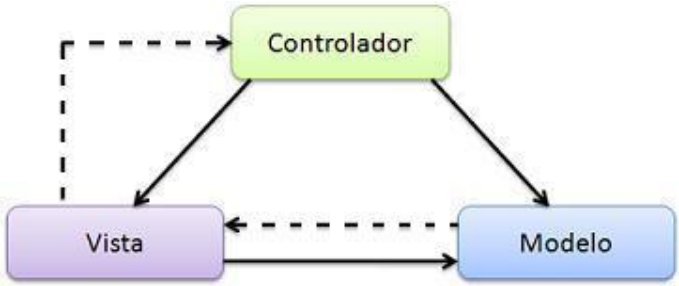

Fig. 3.3. Modelo Vista Controlador 1.

Este modelo de arquitectura presenta varias ventajas:

- La separación del modelo de la vista, es decir, separar los datos de la representación visual de los mismos.
- Sus vistas muestran información actualizada siempre.
- El programador no debe preocuparse de solicitar que las vistas se actualicen, ya que este proceso es realizado automáticamente por el modelo de la aplicación.
- Es mucho más sencillo agregar múltiples representaciones de los mismos datos o información.
- Facilita agregar nuevos tipos de datos según sea requerido por la aplicación ya que son independientes del funcionamiento de las capas.
- Crea independencia de funcionamiento.

- Facilita el mantenimiento en caso de errores.
- Ofrece maneras más sencillas para probar el correcto funcionamiento del sistema.
- Permite el escalamiento de la aplicación en caso de ser requerido.
- La conexión entre el Modelo y sus Vistas es dinámica; se produce en tiempo de ejecución, no en tiempo de compilación.
- Facilita el soporte de nuevos tipos de cliente (móviles, PDAs)

3.3.3. Diagrama de clases del diseño

En un diagrama de clases se presentan las clases del sistema con sus relaciones estructurales y de herencia, así como el contenido dinámico de estas. El diagrama de clases representa las colaboraciones que ocurren entre las interfaces de usuarios, donde cada interfaz puede ser representada como una clase. Es muy importante pues estos son los artefactos que se necesitan modelar para que el desarrollador los implemente y obtener así el producto final con la calidad requerida.

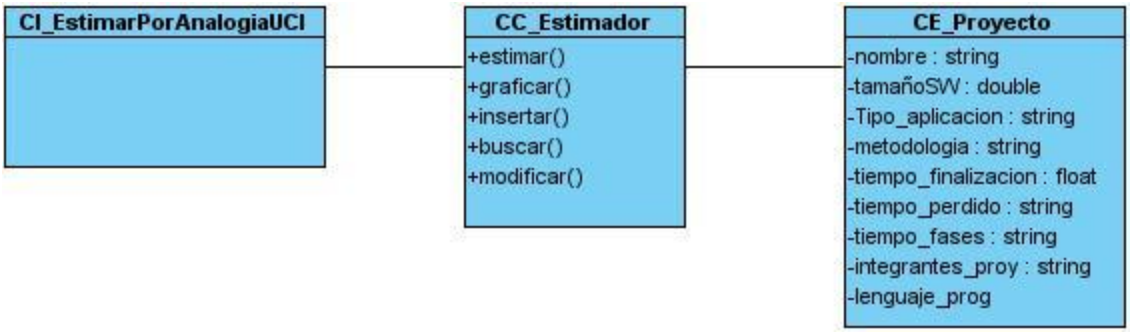

Fig. 3.4. Diagrama de clases de diseño del CU Estimar.

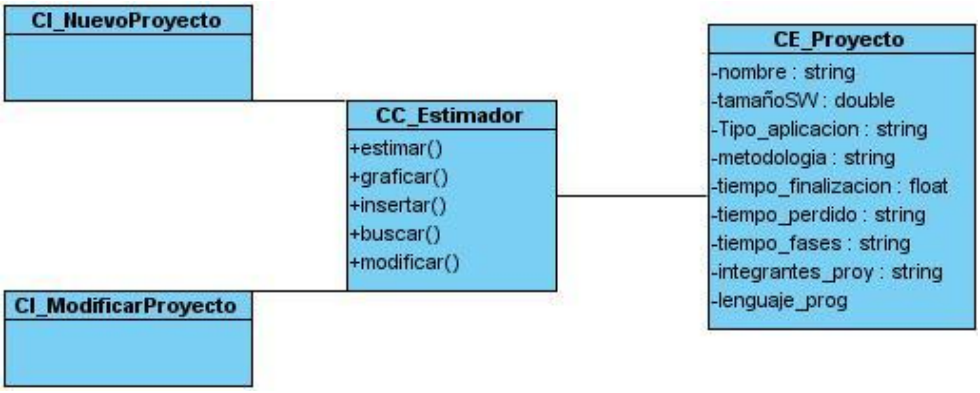

Fig. 3.5. Diagrama de clases de diseño del CU Estimar.

3.3.4. Diagramas de Interacción (Diagramas de Secuencia)

Se presentará el diagrama de interacción del Caso de Uso Estimar, el resto de los diagramas se podrán encontrar en el Anexo 1.

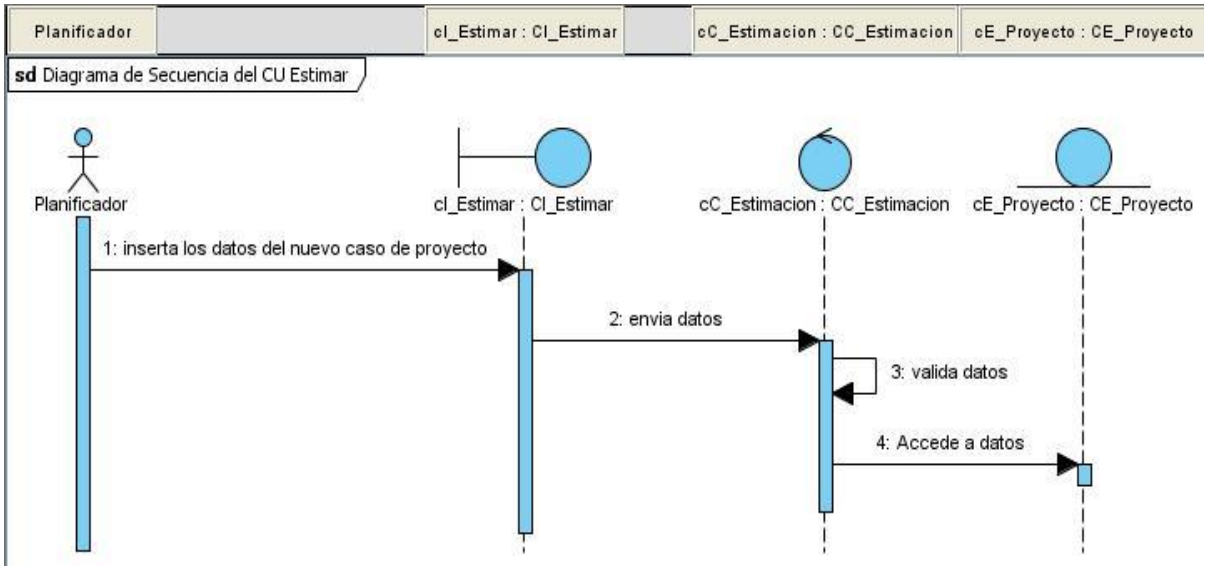

Fig. 3.6. Diagrama de Secuencia del CU Estimar.

3.3.5. Descripción de las clases del diseño

La descripción de las clases del diseño permite conocer los atributos y especificaciones que conforman las mismas, así como sus principales operaciones.

Descripción de las clases para Gestionar Evento

NOMBRE:	Buscar_Listar
Tipo de Clase:	Interfaz
Atributo	**Tipo**
Nombre	Campo de texto
Año	Campo de texto
Buscar	Botón
Cancelar	Botón
Modificar	Botón
Eliminar	Botón
Listado de Eventos	Tabla
Descripción:	Esta clase muestra una interfaz para Buscar un evento. Además muestra el resultado de la búsqueda de los mismos.

NOMBRE:	Modificar
Tipo de Clase:	Interfaz
Atributo	**Tipo**

Nombre	Campo de texto
Año	Campo de texto
Nivel	Lista / Menú
Memorias	Casilla de Verificación
Modalidad	Casilla de Verificación
Listado de modalidades	Tabla
Tipo	Lista / Menú
Nombre	Campo de texto
Insertar	Botón
Eliminar	Botón
Modificar	Botón
Cancelar	Botón
Descripción:	Esta clase muestra una interfaz para Modificar todos los datos relacionados con un evento.

En el presente capítulo se definieron los patrones de diseño que acompañan la documentación e implementación durante el ciclo de desarrollo de software. La arquitectura definida está orientada al logro de una base sólida para la construcción del sistema. Fueron obtenidos los artefactos correspondientes a los flujos de trabajo desarrollados acorde a la metodología utilizada.

Capítulo 4

Implementación

En el presente capítulo, se muestran el diagrama de despliegue y el diagrama de componentes. Además se describe el tratamiento de errores, las acciones para garantizar seguridad y los estándares usados en al codificación. con el objetivo de lograr una mejor comprensión de cómo funciona la herramienta presentada.

4.1 Implementación

La etapa de Implementación comienza con el resultado de la etapa de Diseño y se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares. El objetivo principal de la etapa de implementación es desarrollar la arquitectura y el sistema como un todo. De forma más específica, los propósitos de la Implementación son (Universidad de Granada, 2010):

- Definir la organización del código.
- Planificar las integraciones de sistema necesarias en cada iteración.
- Implementar las clases y subsistemas encontrados durante el Diseño.

4.1.1. Diagrama de Componentes

Un diagrama de componente es un esquema o diagrama que muestra las interacciones y relaciones de los componentes de un modelo. Componente a una clase de uso específico, que puede ser implementada desde un entorno de desarrollo, ya sea de código binario, fuente o ejecutable; dichos

componentes poseen tipo, que indican si pueden ser útiles en tiempo de compilación, enlace y ejecución (Departamento de Sistemas Informáticos,Univesidad de Castilla-La Mancha, 2010).

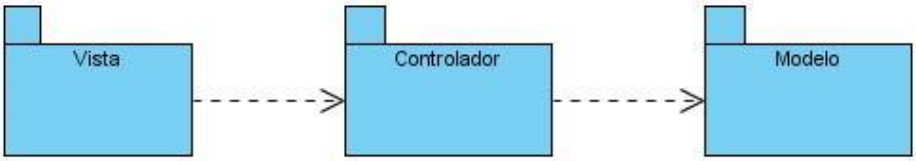

Fig. 4.1. Diagrama de Componentes General

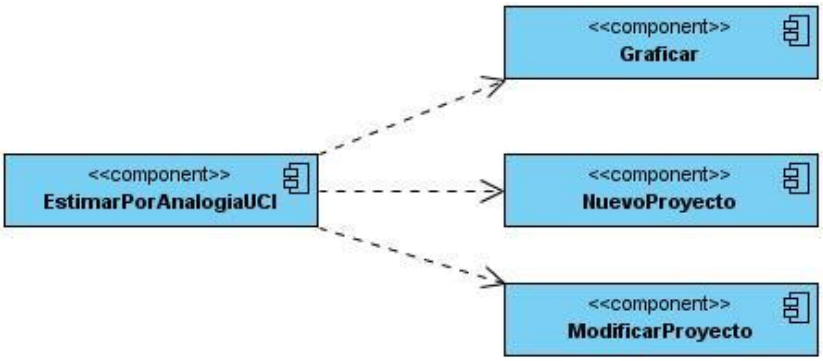

Fig. 4.2. Detalles de las vistas

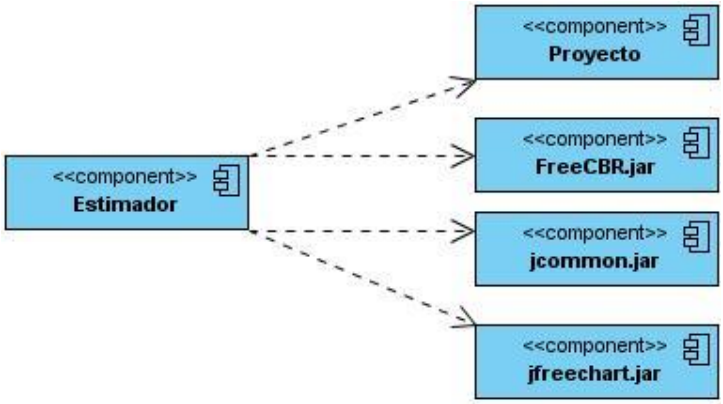

Fig. 4.3. Detalles del Controlador

Fig. 4.4. Detalles del Modelo

4.1.2. Diagrama de Despliegue

Un diagrama de Despliegue muestra cómo y dónde se desplegará el sistema. Las máquinas físicas y los procesadores se representan como nodos, y la construcción interna puede ser representada por nodos o artefactos embebidos. Como los artefactos se ubican en los nodos para modelar el despliegue del sistema, la ubicación es guiada por el uso de las especificaciones de despliegue (Marca Hualpara, et al., 2010)

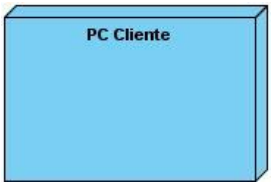

Fig. 4.5. Diagrama de Despliegue

Recursos mínimos de hardware.

- 512 RAM.
- Procesador Pentium.

Recursos mínimos de Software

- JMV (Java Virtual Machine).
- Sistema Operativo: Windows, Linux, etc.

4.1.3. Tratamiento de errores

La interfaz de la herramienta está diseñada para que el usuario realice sus acciones entrando al sistema solo los campos indispensables, ahorrando tiempo al usuario y evitando acciones innecesarias que puedan provocar errores. Para cumplir este objetivo se cuenta con elementos configurables como campos de selección, listas desplegables con varias opciones y campos autogenerados por el sistema de forma transparente al usuario.

El sistema consta con funciones de validación y captación de errores, informando al usuario en momentos importantes sobre la realización de acciones incorrectas y la forma de corregir dicha acción. El sistema, junto al manejo de excepciones, cuenta con la posibilidad de enviar mensajes explícitos al usuario informando sobre la realización exitosa o cancelación de una acción predeterminada, evitando siempre abusar de los mensajes de información.

Una excepción es un evento que ocurre durante la ejecución de un programa que rompe el flujo normal de ejecución. Cuando se habla de excepciones nos referimos a evento excepcional. Muchas cosas pueden generar excepciones: Errores de hardware (falla de disco), de programa (acceso fuera de rango en arreglo), apertura de archivo inexistente, etc.

Cuando se produce una excepción dentro de un método de Java, éste crea un objeto que contiene información sobre la excepción y lo pasa al código llamador. La rutina receptora de la excepción es responsable de reaccionar a tal evento inesperado. Cuando se crea un objeto para la excepción y se pasa al código llamador, lo que ocurre es que se lanza una excepción (Throw an exception). Si el método llamador no tiene un manejador de la excepción se busca hacia atrás en la pila de llamados anidados hasta encontrarlo. Por lo que el manejador atrapa la excepción (catch the exception).

En Java los objetos lanzados deben ser instancias de clases derivadas de Throwable.

Ejemplo:

Throwable e = new IllegalArgumentException(“Stack underflow”);

Throw e;

O alternativamente

Throw new new IllegalArgumentException("Stack underflow");

El manejo de excepciones se logra con el bloque try

```
El manejo de excepciones se logra con el bloque try
try {
        // código
} catch (StackError e )
{
        // código que se hace cargo del error reportado en e
}
El bloque try puede manejar múltiples excepciones:
try {
        // código
} catch (StackError e )
{
        // código para manejar el error de stack
} catch (MathError me)
{
        // código para manejar el error matemático indicado en me.
```

Fig. 4.6. Manejo de Excepciones.

Tipos de Excepciones

Las hay de dos tipos, aquellas generadas por el lenguaje Java. Éstas se generan cuando hay errores de ejecución, como al tratar de acceder a métodos de una referencia no asignada a un objeto, división por cero, etc. Otras no generadas por el lenguaje, sino incluidas por el programador. El compilador chequea por la captura de las excepciones lanzadas por los objetos usados en el código. Si una excepción no es capturada debe ser relanzada.

Jerarquía de Excepciones

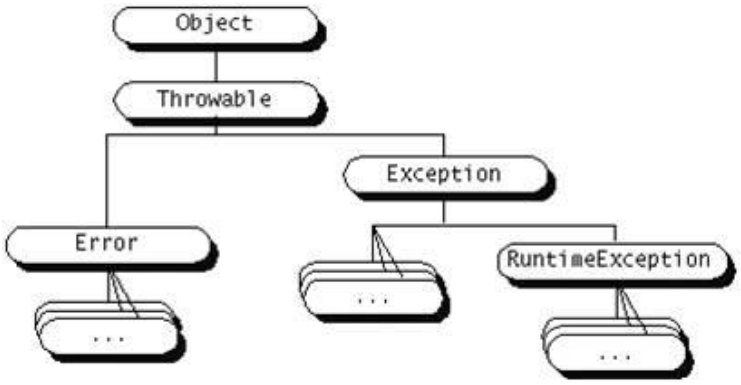

Fig. 4.7. Jerarquía de Excepciones en Java.

4.1.4. Estrategia de codificación. Estándares y estilos a emplear.

Las convenciones o estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. A continuación se presentan algunas de estas convenciones para la programación en el lenguaje Java usadas para la programación de la aplicación implementada.

Comentarios

Deberán incluir:

- Autor.
- Fecha.
- Descripción.

Algoritmo

Cada función debe estar precedida por un encabezado que contenga:

- Objetivo de la función (no describir el procedimiento).
- Comentarios de apoyo a variables y llamadas a funciones.

- Explicación del uso de parámetros no obvios.
- Explicación de uso de valores devueltos (retorno).

Nombre de identificadores

Se considera como identificador a los nombres de variables (arreglos), funciones, así como cualquier tipo de dato definido por el usuario (estructura, clase). Dichos identificadores deberán seguir las siguientes normas, además de las definidas por el propio lenguaje.

- Deberán tener un nombre significativo para que por su simple lectura, pueda conocerse su función, sin tener que consultar manuales o hacer demasiados comentarios.
- Para nombres que se usen con frecuencia o para términos largos, se recomienda usar abreviaturas estándar para que éstos tengan una longitud razonable. Si usa abreviaturas deben manejar la misma lógica en todo el programa.

Identificadores de variables

Para distinguir palabras dentro del nombre deberá emplearse una letra mayúscula o un guión bajo (_), sin mezclar ambas formas en un mismo programa. A partir de la segunda palabra se usará mayúscula; preferiblemente no usar más de 2 palabras por identificador siempre y cuando en el nombre quede implícita la acción.

Ejemplo: tiempo_FP, nombre

Identificadores de clases y métodos

Los identificadores de clases comenzarán escritos con letra mayúscula. En caso de tener más de una palabra estas se separarán por un guión bajo (_) y a continuación se comenzará de nuevo con mayúscula.

Nota: las clases serán implementadas por separado, incluyendo en cada una todas las funcionalidades que sean necesarias y pertenezcan al dominio de la clase.

Ejemplos:

Caso_Proyecto

Organización Visual del Programa

Generales

- No manejar en los programas más de una instrucción por línea.
- Declarar variables del mismo tipo en una línea.

Sangrías

- Las sangrías tendrán una longitud de tres espacios.
- Para las llaves que definen el cuerpo de una función, sangría a un nivel.

Líneas y espacios en blanco

- Insertar una línea en blanco antes y después de una declaración de datos que aparezca entre instrucciones ejecutables.
- Las declaraciones de datos dentro de una función, deberán ir al inicio y separadas de las instrucciones ejecutables de la función por medio de una línea en blanco.
- Los operadores unarios (++, --, etc.) deben ponerse junto a sus operandos, sin espacios intermedios.

En este capítulo se implementó la herramienta para la estimación de proyectos, basándose en los datos históricos de los mismos. Con esta aplicación se espera facilitar y brindar una nueva forma de estimación a los planificadores del CESIM.

Conclusiones

Al concluir la investigación se han cumplido el objetivo y las tareas propuestas. Las herramientas existentes relacionadas con la estimación de proyectos no responden a las necesidades que se presentan en el CESIM, fundamentalmente enfocadas a la utilización datos históricos. Se realizó un análisis de las tendencias actuales de las herramientas y tecnologías lográndose seleccionar las más adecuadas para la implementación del sistema teniendo en cuenta las particularidades del mismo. Después de valorar las principales problemáticas existentes en el CESIM, se definieron una serie de aspectos que afectan el proceso estimación de proyectos que se desarrollan en dicho centro. Se definieron las necesidades de funcionamiento de la aplicación así como los artefactos correspondientes a los flujos de trabajo definidos por la metodología seleccionada.

La aplicación de la técnica de Estimación por Analogía permitirá emplear los datos históricos de los proyectos concluidos, para poder establecer las semejanzas entre los mismos y el proyecto a estimar. Además la técnica de Inteligencia Artificial, RBC, resultó ser una vía de solución para modelar los proyectos como casos, la cual posibilitará recuperar los proyectos semejantes al nuevo, desde la base de conocimiento, la que a su vez almacenará los datos históricos de los proyectos concluidos. Y finalmente se obtuvo una aplicación para la Estimación de Proyectos, que podrá ser utilizada para realizar estimaciones de proyectos en el CESIM.

Recomendaciones

Tomando como punto de partida los resultados obtenidos con la realización de este trabajo de diploma, se hacen las siguientes recomendaciones:

- Utilizar otro método para asignar pesos a las variables, cuando la BC contenga datos reales. (Cálculo de entropía o Aprendizaje inductivo)
- 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.
- Incorporar a la herramienta la opción de generar un reporte sobre la estimación realizada en formato pdf.
- Integrar la aplicación con el Sistema de control y planificación que se está creando en la facultad.

Bibliografía

Bibliografía

[En línea] cocomo : http://html.rincondelvago.com/tecnicas-de-estimacion-de-costo-y-esfuerzo.html.

20smackers. www.20smackers.com. *20smackers.* [En línea] http://www.20smackers.com/products.html.

5406_Bases del Método de Estimación UCI v.3.4.

2006. Athenea. [En línea] 2006. [Citado el: 03 de 12 de 2009.] athenea.ort.edu.uy/publicaciones/ingsoft/.../GP06.Estimaciones.pdf.

Beck, Kent. 1999. *Extreme Programming Explained. Embrace Change.* s.l. : Pearson Education, 1999. ISBN 84-7829-055-9.

CAO, M. ING. JOSÉ IGNACIO. 2006. PRINCIPIOS PARA UN MÉTODO DE ESTIMACIÓN DE PROYECTOS DE SOFTWARE BASADO EN LOS ESCENARIOS. [En línea] 27 de 6 de 2006. [Citado el: 05 de 11 de 2009.]

Capuchino, Ana Ma. Moreno Sánchez. 1996. Unidad 4: Estimación de Proyectos. *Control y gestión de proyectos software.* 1996.

Capuchino, Ana María. 1998. Estimación de Proyectos Software. *Instituto Tegnológico de Buenos Aires.* [En línea] ITBA, 1998. www.itba.edu.ar/archivos/secciones/ovejero-tesisdemagister.pdf.

Corp, Neuron. 2006. ¿Qué es UML? [En línea] 2006. http://www.neuronsrl.com.ar/training/uml/uml_intro.html..

Crystal Methodologies. 2008. Crystal Methodologies. [En línea] 2008. http://crystalmethodologies.blogspot.com/..

Cuadrados Rodríguez, MSc. Santiago y Curbelo Hernández, Dra. Haydee. 2008. *Uso del Razonamiento Basado en Casos combinado con técnicas estadísticas para el diagnóstico de la Hipertensión Arterial.* Villa Clara : s.n., 2008.

De la Torre V., H. Octavio, y otros. SF. *Aplicación del Razonamiento Basado en Casos al Diagnóstico de Generadores Eléctricos.* Guanajuato, Mexico : Laboratorio de Prueba de Equipos y Materiales LAPEM,, SF.

De Nobrega, Maria. 2005. Herramientas CASE: Rational Rose. [En línea] Junio de 2005. http://curso_sin2.blogia.com/2005/060401-herramientas-case-rational-rose.-por-maria-de-nobrega.php..

Departamento de Sistemas Informáticos,Univesidad de Castilla-La Mancha. 2010. [En línea] 2010. [Citado el: 20 de 04 de 2010.] http://www.dsi.uclm.es/asignaturas/42530/pdf/M2tema12.pdf..

2009. Download. *http://www.freedownloadmanager.org/.* [En línea] 12 de 11 de 2009. http://www.freedownloadmanager.org/es/downloads/compartir_outlook_gratis/.

Durán Rubio, Sergio Eduardo. 2003. Puntos por Función.Una métrica estándar para establecer el tamaño del software. *www.inegi.gob.mx.* [En línea] 2003. [Citado el: 16 de 04 de 2010.] http://www.inegi.gob.mx/prod_serv/contenidos/espanol/bvinegi/boletin/2003/num6.pdf.

Escobar, Yanvary, Martínez, Yuraima y Yanez, Joel. 2006. [En línea] Septiembre de 2006. http://www.monografias.com/trabajos39/desarrollo-del-software/desarrollo-del-software2.shtml.

Estimación de proyectos de software: un Caso Práctico. **Salazar, Gabriela. 2009.** número 9,, San Pedro, Costa Rica : Universidad EAFIT, 2009, Vol. Volumen 5. ISSN 1794–9165.

Ferrer, Jose. 2005. Servicio de Hosting de la Fundación. *Universidad las Américas de Puebla.* [En línea] UDPLA, 2005. http://hosting.udlap.mx/estudiantes/jose.ferrercz/modulo4_bases_de_conocimiento.pdf.

Freedownloadmanager.org. 2004. Visual Paradigm for UML (ME) 6.0. [En línea] mayo de 2004. Freedownloadmanager.org.

Froufe, Agustín. 2005. ARQUITECTURA Modelo/Vista/Controlador. [En línea] 2005. http://www.ulpgc.es/otros/tutoriales/java/Apendice/arq_mvc.html..

García de Jalón, Javier, y otros. 2000. Aprenda Java como si estuviera en primero. Universidad de Navarra : San Sebastián, 2000.

Gómez, Adriana, y otros. *COCOMO, UN MODELO DE ESTIMACION DE PROYECTOS DE SOFTWARE.*

Hernández, M. Chantal Pérez. 2002. 5.2.2 Bases de datos y bases de conocimiento. *Estudios de Lingüística del Español.* [En línea] 2002. [Citado el: 20 de 11 de 2009.] http://elies.rediris.es/elies18/522.html.

http://www.monografias.com/trabajos11/metods/metods.shtml. [En línea] [Citado el: 01 de 03 de 2010.]

Johanson, Lars. 2009. softpedia. [En línea] 15 de June de 2009. http://mac.softpedia.com/get/Developer-Tools/FreeCBR.shtml.

Letelier, Patricio Orlando y Penadés, María Carmen. 2008. [En línea] 2008. [Citado el: 05 de 04 de 2010.] http://dialnet.unirioja.es/servlet/articulo?codigo=198360. ISSN 1666-1680.

Letelier, Patricio y Penadés, María Carmen. 2006. *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* 2006.

2008. LinuxZone. *www.linuxzone.es.* [En línea] 2008. [Citado el: 15 de 03 de 2010.] http://www.linuxzone.es/2008/02/10/linus-redacta-los-motivos-de-la-poca-aceptacion-de-linux/.

lpa. http://www.lpa.co.uk/cbr.htm. [En línea] http://www.lpa.co.uk/ind_dow.htm.

lpa lasso public alliance. lpa lasso public alliance. [En línea] http://www.lpa.co.uk/cbr.htm.

Marca Hualpara, Hugo Michael y Quisbert Limachi, Nancy Susana. 2010. [En línea] 2010. [Citado el: 20 de 04 de 2010.] virtual.usalesiana.edu.bo/web/practica/archiv/despliegue.doc.

Martínez, Natalia, y otros. 2009. [En línea] Noviembre de 2009. [Citado el: 05 de 03 de 2010.] http://edutec.rediris.es/revelec2/revelec30/. ISSN.

McConnell, Steve. 1998. *Desarrollo y gestión de proyectos informáticos.* s.l. : McGraw-Hill, 1998. 84-481-1229-6.

Ovejero, Ing. Jose Daniel. 2006. ESTIMACIÓN DE PROYECTOS PARA SISTEMAS BASADOS EN CONOCIMIENTO. [En línea] 2006. http://www.centros.itba.edu.a...ster/Anteproyecto-Ovejero.pdf.

Pasi, Kantelien. 2004. ttp___www.centros.itba.edu.a...ster. [En línea] 2004.

Perez Giraldo, Otoniel. 2006. willydev. *Documento De la Universidad de Guadalajara.* [En línea] 2006. [Citado el: 05 de 11 de 2009.] www.willydev.net/descargas/willydev_planeasoftware.pdf.

Planeación y estimación de Proyectos Informaticos. **Hurtado, Lisette. 2006.** 2006.

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE. 2006. [En línea] 16 de 05 de 2006. [Citado el: 20 de 04 de 2010.]

Portal UCI. Portal UCI. [En línea] [Citado el: 20 de 03 de 2010.] http://www.uci.cu/?q=node/46.

PRESSMAN, R. S. 1998. *Ingeniería del Software: Un Enfoque Práctico.* s.l. : McGraw-Hill, 1998.

—. 2000. *Ingeniería del Software: Un Enfoque Práctico.* s.l. : McGraw-Hill, 2000.

—. 2002. *Ingeniería del Software: Un Enfoque Práctico, Quinta Edición.* s.l. : McGraw-Hill, 2002.

PROGRAMA DE MEJORA UCI. 2010. *5406_Bases del Método de Estimación UCI v.3.4.* Universidad de las Ciencias Informáticas : s.n., 2010.

Remi-Armand Collaris,Eef Dekker. 2009. http://www.ibm.com. [En línea] IBM, 15 de 03 de 2009. http://www.ibm.com/developerworks/ssa/rational/library/edge/09/mar09/collaris_dekker/index.html#1.Use%20case%20points|outline.

Restrepo Montoya, Juan Carlos y Suárez Murillo, Jorge Iván. 2009. Calameo.com. [En línea] 03 de 2009. [Citado el: 05 de 03 de 2010.] http://es.calameo.com/read/0000018161cd6f6345c16.

Tarrillo, Sergio. 2009. http://geeks.ms/. *Gekks.* [En línea] 14 de 01 de 2009. [Citado el: 10 de 04 de 2010.] http://geeks.ms/blogs/sergiotarrillo/default.aspx.

Universidad de Granada. 2010. Etapa /Implementación. *http://aporia.ugr.es/lsi/.* [En línea] 2010. [Citado el: 20 de 04 de 2010.] http://lsi.ugr.es/~arroyo/inndoc/doc/implementacion/implementacion_d.php.

Universidad de las Ciencias Informáticas UCI. 2009. Conferencia # 7,Fase de Construcción.Título: Disciplina Prueba. [En línea] 2009. [Citado el: 10 de 05 de 2010.] http://eva.uci.cu/mod/resource/view.php?id=14103.

Universidad de Santiago de Chile, Facultad de Matemática y Ciencia de la Computación. 2005. *Taller Clasificación de variables.* 2005.

Universidad Politécnica de Valencia. Mayo 2007. *Rational Unified Process (RUP).* Valencia : s.n., Mayo 2007.

—. 2007. Universidad Politécnica de Valencia. [En línea] mayo de 2007. https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Introducci%C3%B3n%20a%20RUP.doc.

Usaola, Dr. Macario Polo. 2004. Mantenimiento Avanzado de Sistemas de Información Pruebas del Software. [En línea] 03 de 08 de 2004. [Citado el: 05 de 05 de 2010.] alarcos.inf-cr.uclm.es/doc/masi/doc/lec/.../polo-apuntesp5.pdf.

Valencia, Universidad Politécnica de. 2001. 2001. Introducción a Herramientas CASE y System Architect. Universidad Politécnica de Valencia. [En línea] Abril de 2001. http://users.dsic.upv.es/asignaturas/eui/mtp/doc-practicas/intro_case_SA.pdf..

Vico.org. 2002. Vico.org. [En línea] 2002. http://www.vico.org/FormMentorOutsourcingUML.pdf..

Vico.org. 2002. Unified Modeling Language. *Vico.org.* [En línea] 2002. http://www.vico.org/FormMentorOutsourcingUML.pdf..

wareseeker. wareseeker.com. [En línea] http://wareseeker.com/Home-Education/sim-selector-easy-cbr.zip/3d1dfedce7.

YancyPM Gestión de Proyectos. 2000. *YancyPM Gestión de Proyectos.* [En línea] PMI, 2000. [Citado el: 15 de 12 de 2009.] http://www.yancypm.com.