Universidad de las Ciencias Informáticas
Facultad 3

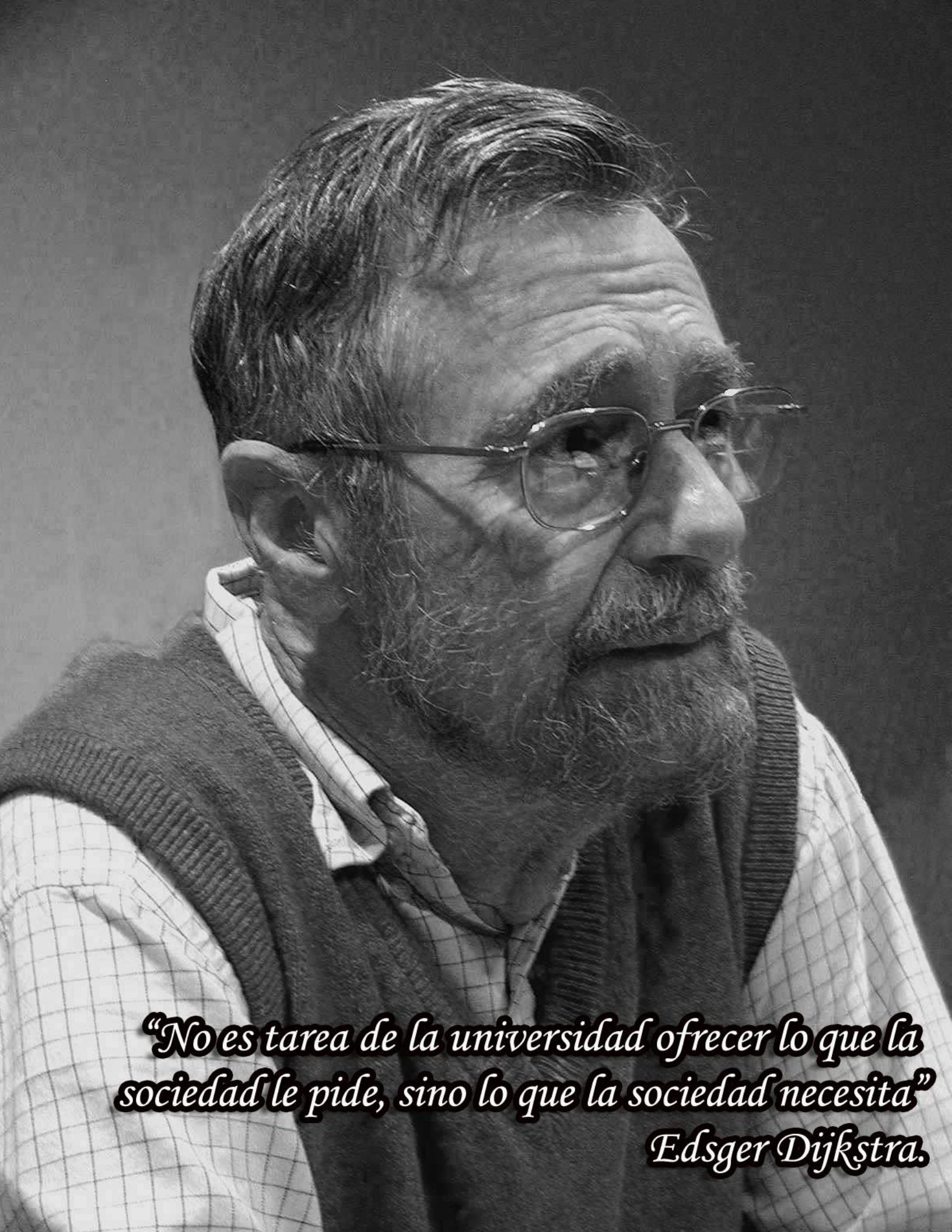
Título: Herramienta para evaluar factibilidad en proyectos de software

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

Autores:
Giorgy Gilberto Cabrera Lamadrid
Pedro Frank Cadenas del Llano

Tutor:
Ing. Marieta Peña Abreu

Ciudad de La Habana, Junio del 2012
"Año 54 de la Revolución"


“No es tarea de la universidad ofrecer lo que la sociedad le pide, sino lo que la sociedad necesita”
Edsger Dijkstra.

Declaración de Autoría

Declaramos ser autores del presente trabajo y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

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

Giorgy Gilberto Cabrera Lamadrid

_____

Firma del Autor

Pedro Frank Cadenas del Llano

_____

Firma del Autor

Marieta Peña Abreu

_____

Firma del Tutor

Dedicatoria

A todo el que ha tomado parte de esta obra maravillosa que es la Revolución cubana, la cual hizo posible la fundación de una Universidad de las Ciencias Informáticas.

De Giorgy:

A mi mamá y mi papá por todo el amor que me han dado.

A toda mi familia por su apoyo y dedicación.

A mis amigos los que siempre han estado mi lado.

De Pedro Frank:

A mis padres y familiares por su entrega y dedicación.

A los amigos que me han apoyado todos estos años.

A todos, gracias.

Agradecimientos

De Giorgy:

A mi mamá y mi papá por estar siempre cuando los he necesitado, por formar en mi el hombre que hoy soy, por todo el amor y cariño que siempre me han dado, por todos los sacrificios que ha hecho durante todos estos años. Los quiero mucho mami y papi.

A mis hermanitas July y Gleysin por ayudarme y apoyarme siempre, por su cariño y estar siempre ahí cuando las he necesitado. Las quiero muchísimo.

A mi tío ñaño por su apoyo y cariño durante estos años.

A mi novia Danelis por todo el amor y el cariño que me ha brindado durante estos meses. Te quiero mucho mi amor.

A mis cuñados Alejandro y Bárbaro por apoyarme y ayudarme siempre en todo.

A Yami y su mami Lisandra por todo el apoyo y cariño que han brindado durante mucho tiempo, siempre las tendré en un lugar especial en mi vida.

A mis suegros Sarita y Héctor a mis cuñados Dani y Wendy por aceptarme en su familia y por el apoyo que me han brindado.

A mis amigos incondicionales Manuel, Katy, Bárbara, mi compañero de tesis Pedro Frank, Claudia, Pedro Enrique, Daniel, Diamy, Martin y Ari. Gracias por todo, siempre los recordaré.

Al comité primario de la UJC con la guía de Yadian, gracias por su apoyo y por contribuir en mi formación como mejor persona.

A todos mis compañeros de estudio que de una forma u otra han estado presente en mi vida como estudiante. Muchas gracias a todos.

Agradecimientos

De Pedro Frank:

A mi mamá, por todo el amor y protección que me ha brindado toda mi vida, por su sacrificio incondicional por sus hijos y por ser la mejor madre del mundo, te adoro mami.

A mi papá, por toda su dedicación y sacrificio, por todo lo que me ha enseñado en la vida, por su apoyo y cariño, te quiero papá.

A mi hermano, por todo su apoyo, por tener tanta fuerza de voluntad, por enseñarme que se puede salir adelante en los malos momentos y superarse en la vida, siempre podrás contar con mi apoyo incondicional.

A mi tío Pancho, por ser mi ejemplo a seguir, por su prestigio, su carácter su apoyo en todo y cuanto necesite, por ser mi segundo padre más que mi tío, muchas gracias Pancho, eres la mejor persona que he conocido.

Al resto de mis familiares, abuelos, tíos, primos que me han enseñado que la familia es lo principal en la vida, y que me han ayudado a ser lo que soñé.

A todos los amigos que me han acompañado estos años de estudios, por los buenos y malos momentos pasados juntos, por el aprecio y la estima de todos, nunca los olvidaré estén donde estén, siempre serán mis amigos.

A todos los profesores que he tenido en mi vida, cada uno de ellos ha dejado en mi su huella de conocimiento tributando a mi formación como profesional y como persona.

A todas las personas que de una forma u otra hicieron posible que llegara a ser lo que hoy soy, muchas gracias.

Resumen

Con el objetivo de fomentar el desarrollo de la industria informática en Cuba se crea la Universidad de las Ciencias Informáticas (UCI). Esta institución cuenta con una plataforma denominada GESPRO para la gestión de los proyectos que en la universidad se desarrollan, pero este sistema carece de herramientas que posibiliten los estudios de factibilidad de proyectos de software, los cuales se realizan de forma manual. En el presente trabajo tiene como objetivo desarrollar e integrar a la plataforma GESPRO una herramienta que permita evaluar la factibilidad de proyectos de software. El sistema fue desarrollado utilizando herramientas y tecnologías libres, se utilizó como metodología de desarrollo programación extrema, el lenguaje de programación Ruby junto al marco de trabajo Ruby on Rails. El módulo obtenido permite realizar estudios de factibilidad económica, técnica y comercial a una cartera de proyectos, obteniéndose finalmente un listado de proyectos ordenados según su factibilidad, atendiendo a los métodos que fueron seleccionados previamente para realizar dicho estudio. El módulo aplica el método de análisis jerárquico para realizar estudios de factibilidad técnica y comercial, mientras que aplica el flujo de caja, valor actual neto, tasa interna de retorno y periodo de recuperación de la inversión como criterios económicos, finalmente unificando todos estos resultado para obtener un resultado único de factibilidad de un proyecto. Todo el desarrollo fue validado y probado mediante pruebas unitarias y de aceptación, obteniéndose una herramienta para evaluar factibilidad en los proyectos de software.

Palabras Clave: evaluación de proyectos, factibilidad económica, factibilidad comercial, factibilidad técnica, herramientas.

Índice de contenidos

Índice de tablas

Índice de figuras

Introducción

El hombre al plantearse nuevos retos avanza más en la evolución de su pensamiento, inmerso en la necesidad de mejorar, busca nuevas técnicas, herramientas y métodos que lo ayuden a perfeccionar lo que ya conoce. A consecuencia del aumento de conocimientos las actividades se hacen más complejas, mayor cantidad de personas se vinculan a estas y se hace uso excesivo de recursos a pesar de que estos son más escasos. Para lograr el adecuado uso de los esfuerzos invertidos y tener garantía de la ejecución y control de los proyectos que se desarrollen, se evidencia la necesidad de una mejor planificación de los mismos.

Las obras o proyectos que son de pequeña magnitud se tornan fáciles de ejecutar, pero cuando se habla de proyectos más complejos es necesario, para el desarrollo de los mismos, la inversión de una gran cantidad de recursos y esfuerzos los cuales tiene un costo y deben ser optimizados para su máximo aprovechamiento. Todo el aseguramiento, organización y planificación de estos recursos se realiza mediante la gestión de proyecto. Esta disciplina logra la coordinación y el trabajo en conjunto de un gran equipo de personas que marchan hacia un mismo objetivo, permite tener el control de todo el desarrollo del proyecto propiciando una buena toma de decisiones frente a cualquier riesgo que se presente.

En sus inicios la gestión de proyecto se limitaba a ciertos servicios, pero la necesidad de darle usos más profesionales, se materializó en los años 50 con el surgimiento de grandes proyectos fundamentalmente militares. La gestión de proyectos ha ido evolucionando y aún continúa haciéndolo con el desarrollo de las tecnologías, las comunicaciones y el avance de los productos. En los mercados cada día aumenta la competencia en la comercialización, lo que exige una mayor calidad de los procesos de producción y el ahorro de los recursos. Para lograr lo anteriormente planteado, es necesario realizar estudios de factibilidad técnica, comercial y económica de los proyectos antes de su puesta en marcha. Con este propósito surge dentro de la disciplina Gestión de proyectos los estudios de factibilidad, los cuales se llevan a cabo fundamentalmente antes de comenzar el ciclo de vida del proyecto.

La actual crisis económica por la que atraviesa el mundo, afecta al país de forma directa, provocando el auge de los estudios de factibilidad de los proyectos que se piensan acometer, de aquí que sea importante conocer la viabilidad y factibilidad de un proyecto y anticipar las posibles dificultades que podrían surgir a fin de que la obra se realice con la requerida calidad y minimizando los costos, pues como el primer secretario del partido Raúl Castro Ruz precisó: “Si planificamos bien, lograremos más ahorro y mayores beneficios”.(1)

Cuba busca nuevas alternativas y nuevos campos de inversión para el rescate de la economía, una de estas alternativas es el desarrollo de software, área que tiene una fuerte competencia a nivel mundial. La correcta selección y gestión de los proyectos a ejecutar en la industria de software es de vital importancia por la gran cantidad de recursos que demandan. Esto provoca que en la ejecución de un proyecto de este tipo sea imprescindible la utilización de modelos que posibiliten mayor competitividad mediante una adecuada planificación, disminuyendo el riesgo de fracaso de los mismos.

La Universidad de las Ciencias Informáticas (en lo adelante UCI), se presenta como una institución de nuevo tipo en Cuba, destinada al desarrollo de la industria de software, mediante la formación de profesionales altamente calificados en el tema y el desarrollo de soluciones informáticas con la vinculación de sus estudiantes y profesores a la producción. Actualmente la UCI se encuentra en una fase de maduración de sus procesos, lo que trae consigo la ejecución de algunos de ellos de manera ineficiente.

El número de proyectos que se desarrollan en la UCI es elevado, existiendo en ocasiones déficit de personal calificado para trabajar en un determinado proyecto, así como de recursos materiales. También se decide la ejecución de algunos proyectos sin un estudio de factibilidad profundo, lo que unido a la ausencia de una herramienta que informatice y agilice este proceso, trae posteriormente consecuencias negativas. Los principales gerentes de proyectos carecen de conocimientos de la existencia de modelos, procedimientos y herramientas para el estudio de factibilidad, trayendo consigo que temas como la gestión de riesgos y la factibilidad técnica en general no sean tratados adecuadamente. Los análisis de factibilidad de los proyectos actualmente

se desarrollan de manera manual, pero la existencia de una herramienta para informatizar este proceso, ayudaría a los especialistas en la toma de decisiones a la hora de realizar una evaluación de un proyecto, mediante la revisión de un volumen mayor de datos, obteniendo resultados más exactos.

Tomando en consideración lo anterior, en el año 2010 se presenta una tesis de maestría (2) que propone un método para evaluar proyectos y priorizarlos según un orden. La aplicación de estos resultados no se lograron generalizar, dado que tienen como debilidad estar basados sobre métodos deterministas y tratan de manera insuficientemente el ruido de la información, por lo cual esta problemática sigue vigente, provocando que se ejecuten los proyectos sin realizar este importante estudio.

Adicionalmente en la UCI se desarrolló un paquete de herramientas para la gestión de proyectos denominado GESPRO, el mismo es una plataforma a la cual se integran la red de centros productivos de la UCI, con el objetivo de facilitar el trabajo de los especialistas en la gestión de los procesos de desarrollo de software. Este paquete actualmente no incluye facilidades para el análisis de factibilidad posibilitando que se puedan ejecutar proyectos que no brinden los beneficios esperados. Por tal motivo surge el interés de establecer un conjunto de procesos que permita evaluar la factibilidad de los proyectos de software de manera eficiente y adaptable a las personas que toman decisiones de la ejecución de un proyecto o no, apoyados por una herramienta informática que se integre a la suite GESPRO.

Actualmente existen varias herramientas para evaluar la factibilidad de proyectos de software pero son privativas y sus costos dificultan su adquisición, además que no se corresponden en su mayoría con las características de los proyectos desarrollados en la universidad, lo que provoca una mala selección de los proyectos a desarrollar.

Por lo anteriormente expuesto se plantea el siguiente **problema a resolver:** ¿Cómo determinar la factibilidad técnica, comercial y económica en proyectos informáticos para conocer qué beneficios reportará su desarrollo, de manera tal que se realice una mejor planificación de los recursos existentes?, por lo que el **objeto de estudio** son los estudios de factibilidad de proyectos, dirigido a cumplir el **objetivo general** desarrollar

una herramienta que contribuya a realizar el análisis de factibilidad técnico, comercial y económico que se integre al ciclo de vida de los proyectos en la plataforma GESPRO, enmarcado en el **campo de acción** los estudios de factibilidad de proyectos de software en la plataforma GESPRO. Para darle cumplimiento a lo anterior se puede proponer los siguientes **objetivos específicos:**

1. Elaborar la fundamentación teórica de la investigación.
2. Seleccionar el modelo a implementar para realizar el estudio de factibilidad técnica, comercial y económica a proyectos de software.
3. Diseñar e implementar una herramienta para realizar estudios de factibilidad técnica, económica y comercial que se integre a la suite de GESPRO.
4. Validar la herramienta desarrollada a partir de un caso de estudio.

Tomando en consideración el problema planteado se propone la siguiente **hipótesis:** Si se desarrolla una herramienta de análisis de factibilidad para evaluar proyectos de software acorde a las necesidades de los proyectos de la Universidad de Ciencias Informáticas que se integre al GESPRO, entonces se contribuirá en la evaluación y selección de proyectos de software atendiendo a su factibilidad técnica, comercial y económica, así como una mejor planificación de los recursos existentes.

Para darle cumplimiento a los objetivos trazados se plantean las siguientes **tareas de la investigación:**

1. Estudio de las principales tendencias para la evaluación de proyectos de software.
2. Estudio de las principales herramientas existentes para la evaluación de proyectos de software.
3. Estudio de los diferentes modelos existentes para realizar análisis de factibilidad a proyectos de software.

4. Estudio de metodologías, lenguajes de programación y gestores de base de datos para el desarrollo de la herramienta.

5. Desarrollo de los principales artefactos propuestos por la metodología de desarrollo de software en las diferentes fases.

6. Implementación de la herramienta para evaluar proyectos e integración al GESPRO.

7. Definición del caso de estudio para la validación de la herramienta.

8. Introducción del caso de estudio en la herramienta.

9. Análisis de los resultados de validación.

Para la elaboración de esta investigación se presenta el siguiente **diseño metodológico:**

Estrategias de investigación:

Exploratoria y la aplicativa: Se realizará una exploración de las distintas tendencias de la evaluación de proyectos de software, incluyendo procesos, métodos y criterios de evaluación dentro de la gestión de proyectos. Todo con el propósito de identificar fortalezas y debilidades de cada uno de los procesos identificando la factibilidad de cada una de ellos y su adaptabilidad a la investigación, para así poder desarrollar un sistema de evaluación que contribuya a resolver las deficiencias que hoy tiene la universidad y siguiendo las mejores tendencias mundiales.

Métodos teóricos:

Histórico Lógico: En la primera parte de la investigación se realiza un estudio de la problemática anunciada, revisando los beneficios y las deficiencias de los sistemas de evaluación de proyectos de software existentes, estableciendo así una conexión entre su concepción histórica y la actualidad.

Hipotético-Deductivo: Se realiza un análisis hipotético-deductivo ya que a partir de la problemática existente hoy en la UCI se plantean los objetivos generales y específicos de nuestra investigación, proponiendo la hipótesis, a la cual se dará seguimiento dando respuesta en el transcurso de la investigación a la misma, siguiendo métodos científicos fundamentados llegando a introducir nuevos conocimientos que posteriormente serán evaluados.

Sistémico: Se plantea el problema y su solución como un todo, realizando un estudio de cada uno de los componentes de la evaluación de proyectos de software, estableciendo dependencias y conexiones entre cada una de las fases para poder lograr un resultado integral e instaurar así un sistema sostenible.

Capítulo 1: Fundamentación teórica

1.1. Introducción

En el presente capítulo son abordados los principales conceptos utilizados en la investigación. Se hace un estudio de los métodos y las herramientas existentes para evaluar factibilidad de proyectos, así como un estudio de varias tecnologías actuales para el desarrollo de sistemas y cuales serán utilizadas para el futuro desarrollo de la herramienta que se propone.

1.2. Principales conceptos

Para comprender los conceptos de gestión de proyecto y estudios de factibilidad de proyectos se presentan las siguientes definiciones:

Proyecto: Un proyecto es un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único. El término temporal se refiere a que cada proyecto define un comienzo y un final al cual se llega cuando se han cumplido todos los objetivos trazados o cuando queda claro que no se han alcanzado o no se alcanzarán los objetivos. Un proyecto crea servicios productos o resultados únicos mediante una elaboración gradual, lo que significa que se va desarrollando en pasos y este va aumentando cada vez (3).

Gestión de proyecto: La gestión de proyectos de software es una actividad protectora dentro de la ingeniería del software. Empieza antes de iniciar cualquier actividad técnica y continúa a lo largo de la definición, desarrollo y mantenimiento del software (4).

Evaluación de proyecto: Proceso por el cual se determina el establecimiento de cambios generados por un proyecto a partir de la comparación entre el estado actual y el estado previsto en su planificación (5).

Estudio de factibilidad: Consiste en la realización de una valoración técnico-económica del objetivo económico-social a que se debe dar solución y las posibles formas de lograrlo, calculándose el costo de las vías alternativas para obtener el resultado y el aporte que éste dé, o sea, invertir un gasto para obtener un beneficio (6).

Factibilidad económica: Los objetivos de esta área son ordenar y sistematizar la información de carácter monetario que proporcionan los análisis técnicos y comerciales, elaborar cuadros analíticos y antecedentes adicionales para valorar su rentabilidad (7).

Factibilidad técnica: Contendrá toda aquella información que permita establecer la infraestructura necesaria para atender su mercado objetivo, así como cuantificar el monto de las inversiones y de los costos de operación de la entidad en formación (8).

Factibilidad comercial: Se puede definir como la función que vincula a los consumidores con el encargado de estudiar el mercado a través de la información, la cual se utiliza para identificar y definir tanto las oportunidades como las amenazas del entorno; para generar y evaluar las medidas de mercadeo así como para mejorar la comprensión del proceso del mismo. Este, por su carácter preliminar, constituye un sondeo de mercado, antes de incurrir en costos innecesarios (8).

1.3. Modelos para la evaluación de proyectos de software

Existen varios modelos para realizar la evaluación de proyectos, estos se pueden agrupar según su aplicación en dos grupos: para la evaluación de forma individual y a partir de una cartera o portafolio de proyecto. Dentro del método de evaluación individual existen diversas técnicas para realizar esta tarea, dentro de ellas se pueden destacar los modelos económicos y los modelos de puntuación, el modelo que se propone realiza una combinación de ambos para obtener mejores resultados en el proceso de evaluación, lo que permite conocer el impacto que tendrá en la sociedad y la economía.

Para desarrollar una herramienta que contribuya a realizar el análisis de factibilidad de los proyectos de software, que se integre al ciclo de vida de los proyectos en la plataforma GESPRO, se implementará el modelo propuesto por la ingeniera Marieta Peña Abreu titulado: “Modelo para análisis de factibilidad en la evaluación de Proyectos de Software”, como resultado de la propuesta de solución de una tesis de maestría. En este modelo se propone la aplicación de métodos híbridos que permitan realizar la evaluación individual y

de portafolios, destacándose en este el método AHP[1] que permite la valoración de aspectos cuantitativos y cualitativos, así como los métodos económicos(9).

El modelo se aplica antes de comenzar el desarrollo del software, dentro de la fase conceptual del ciclo de vida de los proyectos. En el mismo intervienen actores como el especialista principal que es el que selecciona los evaluadores y da seguimiento a todo el proceso. También participan los expertos que realizan la ponderación de los métodos de análisis jerárquico.

Está estructurado por un flujo de dos fases: Iniciación y Evaluación. Cada una de estas contiene entradas y actividades para realizar el proceso, así como un conjunto de salidas.

En la fase inicial se recoge mediante la ficha de proyecto la información preliminar de estos y se realiza la entrada de los datos necesario para realizar los métodos económicos. Posteriormente se seleccionan los expertos que participarán, para esto se tiene en cuenta su nivel de competencia para tener un mayor grado de aceptación en los resultados obtenidos. Finalmente se determinan los criterios de evaluación teniendo en cuenta fundamentalmente que éstos tributen a los objetivos estratégicos que persigue la organización, quedando como salida de esta fase el listado de expertos que participan en el proceso de evaluación con su índice de competencia en el tema y un listado de criterios que son seleccionados por el especialista principal según las necesidades de los proyectos a evaluar.

La fase de Evaluación tiene por entradas las salidas de la fase anterior y se incorpora un listado de métodos para efectuar la evaluación. Durante la misma se obtienen los resultados finales del proceso por lo que es la fase que tiene mayor peso.

La aplicación de este proceso de evaluación arroja como resultado final un listado de proyectos priorizados según su factibilidad y en función de su valor de certidumbre (9).

[1] **AHP** Analytic Hierarchy Process (proceso de análisis jerárquico en español), está diseñado para resolver problemas complejos de criterios múltiples.

1.4. Herramientas informáticas para la evaluación de factibilidad de proyectos

Actualmente existen varias herramientas informáticas dedicadas a facilitar la toma de decisiones y la evaluación de factibilidad de proyectos. Específicamente que utilizan el método AHP se encuentran Expert Choice, HIPRE 3 + INPRE y Criterium (10). Para la evaluación de factibilidad económica existen software como EvalAs Managerial Analyzer y EasyPlanEx. Finalmente para la evaluación de factibilidad técnica y comercial se encuentra DECIDE. Además existen otros como Decision Lab, EvalAs, Crystall Ball, M-Macbeth, AIM, Electre, McView (2). A continuación se realiza un análisis de las más utilizadas.

1.4.1. Expert Choice

Constituye una suite de herramientas basado en el Proceso Analítico de Jerarquía (AHP). Están diseñadas especialmente para la toma de decisiones. Esta herramienta posee varias alternativas de solución entre las que se encuentran Comparion™ Suite, Comparion™ TeamTime™, estos dedicados al trabajo en la web. Por otra parte está Expert Choice Desktop que corre sobre plataforma de Windows es el principal producto de esta gama y permite la integración con herramientas como Microsoft Project, Microsoft Excel y el gestor de base de datos Oracle, además permite especificar los roles de los que participan en el proceso y una jerarquía alternativa (11). Esta herramienta al igual que otras que utilizan el proceso AHP, presenta algunas dificultades asociadas al uso de este proceso (12). De manera general todas las alternativas de esta herramienta son privativas y es necesario pagar para su uso y soporte.

1.4.2. EasyPlanEx

Es un software desarrollado por la empresa BoraSystems localizada en Santiago de Chile, el lanzamiento de este software se realizó el 17 de diciembre de 2003. Este permite la evaluación y optimización de proyectos de inversión. Tiene entre sus objetivos principales realizar análisis de sensibilidad, medir el riesgo del proyecto considerando la

incertidumbre utilizando la técnica de Montecarlo[2], asegurar que el modelo formulado esté correcto (ecuaciones, fórmulas, etc.), generar automáticamente gráficos y una documentación completa y consistente del proyecto, reutilizar proyectos tipo, calcular la probabilidad de ocurrencia de un resultado como el Valor Actual Neto (VAN) y la Tasa Interna de Retorno (TIR). Además genera las series de valores para los datos del modelo, utilizando reglas predefinidas, autodocumentadas, permite la conversión automática de monedas y permite importación y exportación de datos de plantillas electrónica. Actualmente se encuentra disponible en dos idiomas Inglés y Español, es compatible solo con el Sistema Operativo Windows por lo que no es multiplataforma, este software se entrega en siete versiones similares en funcionalidad pero con capacidades diferentes y limitaciones de uso, propias de cada una, por lo que es un herramienta privativa que tiene un costo de uso, posee una versión de prueba de 15 días sin costo de licencia (13).

1.4.3. DECIDE

Es un software que facilita la elaboración de planes de negocio y la formulación y evaluación de proyectos de inversión proporcionando metodologías y herramientas para la toma de decisión, desarrollado por una empresa Mexicana llamada Soluciones Informáticas y Aplicaciones Crediticias S.A de C.V. Entre sus funciones principales se encuentran que permite realizar evaluación de un proyecto de inversión, elaborar estudios de mercado, análisis técnico, económico, financieros y de riesgos. Tiene entre sus principales ventajas la particularidad de poder evaluar proyectos de cualquier tamaño, cuantía y actividad económica. Utiliza en sus dos versiones de Monousuario/Multiusuario y en la de Servidor, corre sobre la plataforma Windows XP SP3 o superior. Es una herramienta privativa por lo que tiene costo de licencia para su uso (14).

[2] El método de Montecarlo es una herramienta de investigación y planeamiento; básicamente es una técnica de muestreo artificial, empleada para operar numéricamente sistemas complejos que tengan componentes aleatorios.

1.4.4. Valoración de herramientas

Las herramientas analizadas anteriormente presentan un conjunto de ventajas que facilitan la toma de decisiones en el momento de evaluar un proyecto, haciendo este proceso más sencillo y eficaz para los profesionales que los utilicen. Están basadas en el uso de métodos muy utilizados por gran cantidad de especialistas de la disciplina de gestión de proyecto. A pesar de sus ventajas también presentan un conjunto de desventajas, de manera general todas son propietarias por lo que es necesario pagar un costo de licencia para su uso, son compatibles con un sistema operativo en específico lo que lo convierte en dependiente de la plataforma y no todas tienen en cuenta la factibilidad técnica y comercial, dos elementos muy importantes a tener en consideración a la hora de evaluar la factibilidad de proyectos. Estas herramientas no satisfacen completamente las necesidades que existen en la Universidad de las Ciencias Informáticas por las características particulares que presentan los proyectos que se ejecutan en la misma, además no se pueden integrar a la plataforma GESPRO sin que medie un sistema de interoperabilidad entre ellos, haciendo más complejos los estudios de factibilidad al tener que utilizar por parte de los especialistas de la gestión de proyecto, dos sistemas diferentes para realizar dichos estudios. Por lo antes expuesto es necesario, una herramienta que cumpla y satisfaga las necesidades que hoy existen en la UCI y que se integre como un módulo a la plataforma GESPRO.

1.5. Sistema de gestión de proyectos GESPRO

La gestión de proyecto se encarga de organizar y administrar todos los recursos necesarios para un proyecto de manera eficiente, permitiendo que este se termine dentro del alcance, tiempo y con los costos definidos inicialmente. Actualmente a nivel internacional, existe una tendencia a utilizar herramientas informáticas para apoyar la gestión de proyectos y facilitar así el trabajo de los especialistas. La Universidad de las Ciencias Informáticas está al tanto de esta necesidad y por tal motivo desarrolló un Paquete de Gestión de Proyecto (GESPRO) (15) como herramienta para apoyar esta importante disciplina. GESPRO es comercializable desde las empresas comercializadoras asociadas a la universidad. Es un producto registrado en el Centro Nacional de Derecho de Autor (CENDA) con No. Registro 1540-2010 (16).

El GESPRO abarca varias áreas de la Gestión de Proyecto, las cuales están presentes en el sistema mediante funcionalidades y módulos. Entre estas podemos encontrar: La gestión de portafolios de proyectos, la gestión del alcance de productos, la gestión del tiempo, la gestión de riesgos de proyectos, la gestión de comunicaciones, la gestión de recursos humanos y materiales y la gestión documental. En las siguientes imágenes se puede ver los módulos con los que cuenta el GESPRO en la actualidad.

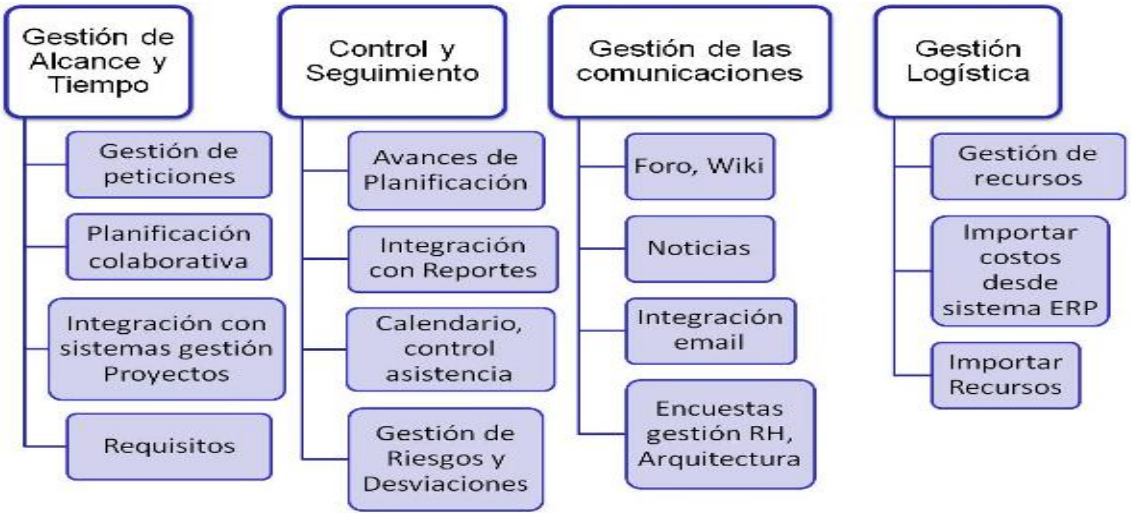


Figura 1. Módulos del GESPRO

Figura 2. Módulos del GESPRO

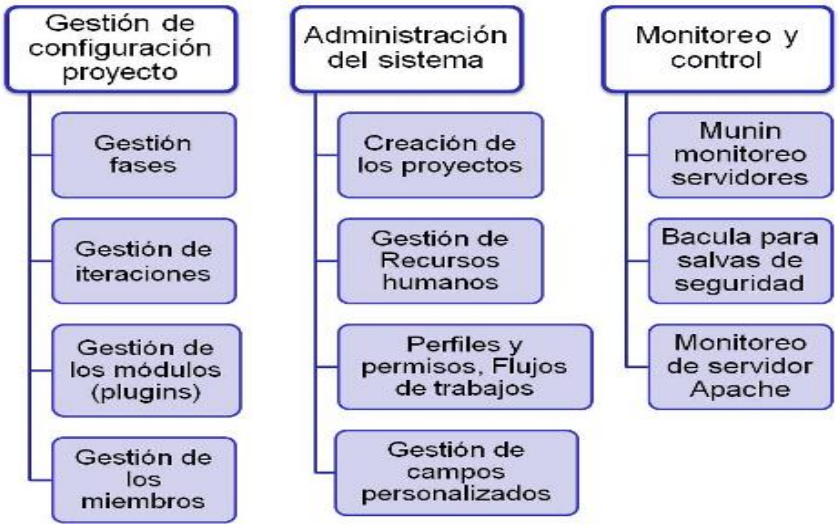


Figura 3. Módulos del GESPRO

Actualmente GESPRO no cuenta con un módulo para realizar estudios de factibilidad de proyectos de software, por lo que existe la necesidad de incluir una funcionalidad que permita realizar tales estudios antes de acometer y gestionar proyectos, asegurando en gran medida el éxito de estos y facilitando así el trabajo de los especialistas, los cuales actualmente realizan los estudios de forma manual.

1.6. Metodologías de desarrollo de software

El desarrollo de software es una tarea compleja y riesgosa que involucra un gran equipo de personas trabajando en común por lo que se hace necesario mantener un estricto control sobre los procesos de manera que se garantice la organización y coordinación de todo el trabajo.

Un proceso de desarrollo de software es un conjunto de actividades para transformar los requerimientos de un usuario en un software, define quién está haciendo qué, cuando y como alcanzar un determinado objetivo.(4)

Para garantizar la gestión del desarrollo de un sistema están definidas varias metodologías, pues de no ser guiado el proceso lo que obtenemos finalmente son clientes insatisfechos y productos inoperables. Pero no solo se trata de regirse por una

metodología sino de seleccionar la adecuada, cada una de ellas está definida para proyectos con características específicas, por lo que se debe analizar bien cuál de las existentes va a resultar más factible emplear pues esta decisión influye directamente en el éxito de un proyecto.

Las metodologías de desarrollo se pueden enmarcar en dos grandes grupos, las llamadas metodologías tradicionales y las metodologías ágiles. Las tradicionales enfatizan en el uso exhaustivo de documentación durante todo el ciclo de vida del proyecto lo que brinda la ventaja de poder efectuar un futuro mantenimiento de manera más eficiente, son recomendadas para los proyectos de grandes dimensiones y con grandes equipos de desarrollo. En tanto las metodologías ágiles dan mayor importancia a la capacidad de respuesta a los cambios, se enfatiza en la satisfacción del cliente y promueven el trabajo en equipo. Su selección depende de qué producto se desee desarrollar, de las dimensiones que tendrá el mismo, del tiempo que se disponga y del equipo de trabajo, entre otros factores.

1.6.1. Proceso Unificado de Desarrollo (RUP)

El proceso unificado de desarrollo de software (RUP[3]) junto al Lenguaje Unificado de Modelado (UML[4]), constituye la metodología más utilizada para el análisis, implementación y documentación de sistemas de software (17).

La metodología RUP está definida por tres aspectos fundamentales:

Procesos dirigidos por casos de usos: Un caso de uso es un fragmento de funcionalidad del sistema que proporciona al usuario un resultado, el conjunto de todos los casos de usos constituye el modelo de casos de uso y describe la funcionalidad total del sistema, por lo que son usados para especificar los requisitos de un sistema, para guiar su diseño,

[3] **RUP:** Acrónimo en inglés de Rational Unified Process

[4] **UML:** Acrónimo en inglés de Unified Model Language

implementación y prueba, de este modo los casos de usos no solo inician el proceso sino que le proporcionan un hilo conductor.

Centrado en la arquitectura: La arquitectura es una vista del diseño completo con las características más importantes resaltadas, dejando los detalles a un lado. Esta se construye con los casos de usos más significativos del sistema que representan las funciones claves del sistema en desarrollo.

Iterativo e incremental: Como el desarrollo de un proyecto es un proceso complejo, es práctico dividir el trabajo en partes más pequeñas o mini proyectos, cada mini proyecto es una iteración que resulta en un incremento, las necesidades de los usuarios no pueden definirse completamente al principio, por lo que se refinan en iteraciones sucesivas.(17)

Esta metodología divide en 4 fases el desarrollo de software:

- Inicio: Su objetivo es determinar la visión del proyecto.
- Elaboración: Su objetivo es determinar la arquitectura óptima.
- Construcción: Su objetivo es obtener la capacidad operacional inicial.
- Transición: Su objetivo es llegar a obtener el resultado del proyecto.

El ciclo de vida que se desarrolla por cada iteración es llevado bajo dos disciplinas:

Disciplina de Desarrollo: Ingeniería de negocio, Requerimientos, Análisis y diseño, Implementación, Pruebas.

Disciplina de Soporte: Configuración y administración del cambio, Administrando el proyecto, Ambiente.

Como se puede comprobar esta metodología divide el trabajo en muchas etapas durante las cuales se genera una gran cantidad de documentación que es muy útil para proyectos de grandes dimensiones, no siendo así para proyectos pequeños para los que se dispone

de poco tiempo y de un equipo de desarrollo pequeño como es el caso de la herramienta para la evaluación de factibilidad de proyectos que se propone en este trabajo.

1.6.2. SCRUM

Es una metodología ágil para el desarrollo de proyectos que brinda un conjunto de buenas prácticas de trabajo con la capacidad de ofrecer valor al producto final y agilidad en el desarrollo.

Sus principios son válidos para entornos que trabajan con requisitos inestables y necesitan agilidad. En esta metodología la gestión no se basa en el seguimiento de un plan sino en la adaptación continua a las circunstancias de la evolución del proyecto, está adaptado a las personas antes que a los procesos y emplea un desarrollo ágil iterativo e incremental.

El desarrollo se inicia desde la visión general del producto, dando detalle solo a las funcionalidades de mayor prioridad, las que se van a desarrollar en primer lugar en un periodo de tiempo breve.

Cada uno de los ciclos de desarrollo es una iteración (sprint) que produce un incremento terminado y operativo del producto (18).

Características de los campos de SCRUM:

Incertidumbre: Como elemento consustancial y asumido en el entorno y en la cultura de la organización.

Auto-organización: No hay roles de gestión que marquen pautas o asignación de tareas.

Fases de desarrollo solapadas: Los trabajos que se llevan a cabo pierden el carácter de fase y son actividades que se realizan en cualquier momento, de forma simultánea, o a demanda según las necesidades en cada iteración.

Control sutil: Se establecen controles para evitar que el ambiente de ambigüedad, inestabilidad y tensión en el que el equipo trabaja derive hacia el descontrol.

Difusión y transferencia del conocimiento: Todos los miembros del equipo aportan y aprenden del resto del equipo.

Para SCRUM los documentos son soporte de documentación, permiten la transferencia del conocimiento, registran información histórica, son obligatorios en cuestiones legales o normativas, pero son menos importantes que los productos que funcionan, por lo que no pueden sustituir, ni pueden ofrecer la riqueza y generación de valor que se logra con la comunicación directa entre las personas y a través de la interacción con los prototipos. Por eso, siempre que sea posible debe preferirse, y reducir al mínimo indispensable el uso de documentación, que genera trabajo que no aporta un valor directo al producto (18).

Las características vistas hasta ahora se adecuan a las necesarias para desarrollar un sistema como el que se propone, pero se debe anotar como desventaja que el equipo de desarrollo en cuestión está compuesto solo por dos personas, por lo que se hace necesario minimizar las tareas para lograr un mejor aprovechamiento del tiempo.

1.6.3. Programación Extrema (XP)

La metodología programación extrema (XP[5]) fue concebida y desarrollada para direccionar las necesidades específicas del desarrollo de software llevado a cabo por pequeños equipos en aras de satisfacer requisitos vagos y cambiantes (19).

Son fundamentos de esta metodología ágil:

- Escribir pruebas unitarias basadas en los principales procesos lo que permite predecir posibles fallas futuras.
- Integrar y probar el sistema en su conjunto varias veces al día.
- La producción de todos los programas de dos en dos, dos programadores una pantalla.

[5] **XP:** Acrónimo en inglés de Extreme Programming.

- Proyectos a partir de un diseño simple que evoluciona constantemente para aumentar la flexibilidad necesaria y eliminar la complejidad innecesaria.

- La reutilización de código para lo cual se crean patrones o modelos estándares, siendo más flexibles al cambio.

Esta metodología brinda como ventaja un mayor aprovechamiento del tiempo, lo que permite agilizar todo el proceso, pues el tiempo que se invierte en documentar todo el ciclo de desarrollo de un proyecto se aprovecha en la implementación del mismo, pero esto trae consigo implícito una desventaja, si el proyecto es muy abarcador y no se documenta adecuadamente en un futuro cuando se desee dar mantenimiento al mismo puede resultar muy engorroso o casi imposible si no se encuentran presentes los integrantes del equipo de desarrollo original.

1.6.4. Fundamentación de la metodología a utilizar

Después de realizado el estudio de algunas metodologías de desarrollo de software, se selecciona para el desarrollo de la herramienta propuesta XP por ser una metodología ágil, diseñada para equipos de trabajo pequeños, centrada en vincular al cliente en el ciclo de desarrollo, incrementando la posibilidad de éxito, minimizando los riesgos de no conformidades y de obtener un producto final rechazado por el cliente por no cumplir con los objetivos y especificaciones trazadas. La metodología XP permite responde de manera eficiente a los cambios que se puedan presentar durante todo el desarrollo de la herramienta, proponiendo un ciclo de vida dinámico. El proceso de prueba de XP posibilita probar cada funcionalidad al finalizar cada iteración comprobando si cumple con los requisitos de la herramienta.

1.7. Herramientas CASE de modelado

Las herramientas Computer Aided Software Engineering (CASE) son un conjunto de programas que dan asistencia a los analistas, ingenieros de Software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente y la documentación. Este puede ser generalmente aplicado a cualquier sistema o colección

de herramientas que ayudan a automatizar el proceso de diseño y desarrollo de software (4).

Las ventajas de la utilización de una herramienta CASE, se evidencian en la mejora de la calidad del desarrollo del software y el aumento de la productividad.

1.7.1. Rational Rose

Es una herramienta basada en modelos. Se integra con las bases de datos y los ambientes integrados de desarrollo, IDE por sus siglas en inglés, de las principales plataformas de desarrollo. Todos sus productos dan soporte al lenguaje UML. Está hecho especialmente para analistas que utilizan la metodología de desarrollo RUP (20).

Esta herramienta constituye un entorno de modelado que permite generar código a partir de modelos para múltiples lenguajes. Ofrece un lenguaje de modelado común que agiliza la creación del software. Algunas de sus características más importantes son:

- Capacidad de análisis de calidad de código.
- Posee complementos los que proveen visualización, modelado y las herramientas para desarrollar aplicaciones de Web.
- Modelado UML para trabajar en diseños de base de datos, con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos.
- Capacidad de crear definiciones de tipo de documento XML[6] para el uso en la aplicación.
- Integración con otras herramientas de desarrollo y generación de informes para optimizar la comunicación del equipo.

[6] **XML:** Acrónimo en inglés de Extensible Markup Language

1.7.2. Visual Paradigm

Es una de las herramientas que utiliza UML como lenguaje de modelado, está considerada como muy completa y fácil de usar, con soporte multiplataforma y que proporciona excelente facilidades de interoperabilidad con otras aplicaciones.

Fue diseñada para automatizar y acelerar el ciclo de desarrollo de software, permitiendo la captura de requisitos, análisis, diseño e implementación. Tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de base de datos a partir del esquema de las clases.

Permite realizar ingeniería inversa, a partir del código fuente de programas, podemos obtener modelos UML, creando de manera simple toda la documentación. Está diseñada para usuarios interesados en sistemas de software de gran escala, sistemas desarrollados bajo el paradigma orientados a objetos y apoya los estándares más recientes de las notaciones de Java y de UML.

Es una herramienta colaborativa, incorpora el soporte para trabajo en equipo, permite a varios desarrolladores trabajar a la vez en el mismo diagrama y ver en tiempo real los cambios hechos por otros.

Posee soporte para aplicaciones Web, generación de código a partir de los diagramas para varias plataformas, integración con otras herramientas de desarrollo como Eclipse, NetBeans, Oracle Jdeveloper y otras (21).

1.7.3. Fundamentación de la herramienta de modelado a utilizar

Como herramienta de modelado se selecciona el Visual Paradigm pues permite trabajar de forma colaborativa, hacer un trabajo más organizado y ágil. Es una herramienta multiplataforma fácil de utilizar, que permite generar código SQL[7], posibilitando crear las tablas y su relaciones en sistema gestor de base de datos a partir de los diagramas

[7] **SQL:** Acrónimo en inglés de Structured Query Language.

entidad relación y modelo de datos permitiendo así ahorrar tiempo durante la implementación. Además actualmente la UCI cuenta con una licencia que permite el uso de esta herramienta.

1.8. Sistemas gestores de bases de datos

PostgreSQL es un Sistema de Gestión de Bases de Datos (SGBD en lo adelante) objeto relacional, distribuido bajo la licencia BSD y con su código fuente libre disponible para el acceso de todos los usuarios. Sus últimas versiones son muy competitivas siendo actualmente una de las alternativas más robustas, potentes en el mercado de los SGBD, y la mejor que existe de código abierto.

Entre las principales características que posee PostgreSQL es que utiliza el modelo cliente/servidor y utiliza tecnología multiproceso en vez de multihilos, lo que asegura mayor estabilidad del sistema. En caso de fallar un proceso no afectará al resto y el sistema continuará estable. Funciona muy bien con grandes volúmenes de datos y con una alta concurrencia de usuarios accediendo simultáneamente al sistema (22).

Otras características de PostgreSQL son:

- Es un SGBD con ACID[8].

  Atomicidad: Es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.

  Consistencia: Integridad. Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper las reglas y directrices de integridad de la base de datos.

[8] **ACID**: Atomicity, Consistency, Isolation, Durability.

Aislamiento: Es la propiedad que asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información sean independientes y no generen ningún tipo de error.

Durabilidad: Es la propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema.

- Completa documentación
- Es un SGBD multiplataforma compatible con Linux y UNIX en todas sus variantes (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) y Windows 32/64bit.
- Posee numerosos tipos de datos y permite definir nuevos tipos, siendo de esta manera escalable para cada una de las necesidades.
- Posee APIs para programar en varios lenguaje entre los que se encuentran C/C++, Java, .Net, Perl, Python, Ruby, PHP por solo mencionar algunos de los más utilizados.

1.8.1. Fundamentación del gestor de base de datos a utilizar

PostgreSQL es actualmente el sistema gestor de base de datos que utiliza la plataforma GESPRO para su explotación, por tal motivo ya que la herramienta a desarrollar es una extensión de este sistema, es necesario utilizar el mismo SGBD. Luego de hacer un análisis de PostgreSQL, podemos constatar la gran cantidad de ventajas que posee este gestor, lo que asegura el correcto funcionamiento y estabilidad del sistema en cualquier circunstancia. Actualmente se encuentra entre los mejores SGBD del mercado y es el mejor de código abierto. Su elección está justificada por todas estas características que lo respaldan.

Para la administración y desarrollo de la base de datos se utilizará pgAdmin III, la más popular de las herramientas de código abierto utilizadas con este propósito para PostgreSQL. Es compatible con todas las versiones de este popular SGBD a partir de su versión 7.3, es multiplataforma, está diseñado para facilitar la administración de la base

de datos y para responder a todas las necesidades de los usuarios (23). Actualmente la UCI pertenece a la Sociedad Latinoamericana de PostgreSQL lo cual facilita el intercambio de información y experiencia referente a este popular gestor de base de datos, garantizando un mejor aprovechamiento del mismo.

1.9. Lenguaje de desarrollo web Ruby

Un lenguaje de programación permite crear programas que posibiliten la interacción entre el humano y el ordenador. La herramienta que se propone es un programa informático que será desarrollada como un componente del paquete GESPRO, el cual se encuentra montado sobre la plataforma Redmine, la que a su vez se encuentra desarrollada en lenguaje Ruby, atendiendo a estas condiciones se tiene que el lenguaje de programación que se debe usar para el desarrollo es Ruby para poder lograr una perfecta integración.

Es un lenguaje de programación dinámico y de código abierto enfocado en la simplicidad y productividad. Su elegante sintaxis se siente natural al leerla y fácil al escribirla. Es interpretado, reflexivo y orientado a objetos (24).

Está diseñado para la productividad y la diversión del desarrollador, siguiendo los principios de una buena interfaz de usuario. Sostiene que el diseño de sistemas necesita enfatizar las necesidades humanas más que las de la máquina (25).

Sigue el principio de la menor sorpresa, lo que significa que el lenguaje debe comportarse de tal manera que minimice la confusión de los usuarios experimentados.

Ruby es orientado a objetos, todos los tipos de datos son un objeto, incluidas las clases y tipos que otros lenguajes definen como primitivas. Toda función es un método. Las variables siempre son referencias a objeto. Ruby soporta herencia con enlace dinámico, pero no soporta herencia múltiple.

Características:

- Orientado a objetos.
- Cuatro niveles de ámbito de variable: global, clase, instancia y local.

- Manejo de excepciones.
- Expresiones regulares nativas similares a las de Perl a nivel del lenguaje.
- Posibilidad de redefinir los operadores (sobrecarga de operadores).
- Recolección de basura automática.
- Altamente portable.
- Carga dinámica de bibliotecas compartidas en la mayoría de las plataformas.
- Amplia librería estándar.
- Soporta alteración de objetos en tiempo de ejecución.
- Continuaciones y generadores.

1.10. Marco de trabajo Ruby on Rails

Rails es un completo marco de trabajo (framework) para el lenguaje Ruby, utilizado para crear aplicaciones Web. Tiene varias características que lo convierten en una herramienta muy potente destinada a crear aplicaciones profesionales con diferentes fines (26). Entre las más significativas están:

- Utiliza la arquitectura Modelo-Vista-Controlador para crear las aplicaciones, donde se separa la lógica del negocio, el acceso a datos y la presentación en capas independientes una de la otra.
- Utiliza varios servidores web, entre los más utilizados y recomendados están Apache y Lighttpd.
- Es independiente del Sistema Gestor de Base de Datos, siendo compatible con base datos desarrolladas en MySQL, PostgreSQL, SQLite, Oracle, SQL Server, DB2, o Firebird.

- Es multiplataforma, lo cual significa que se puede desarrollar una aplicación en cualquier Sistema Operativo (SO en lo adelante). Aunque se hace una recomendación de desarrollar sobre un SO basado en UNIX.

Rails por sus ventajas, ha sido tomado como referencia a la hora de crear marcos de trabajo para otros lenguajes de desarrollo web, como es el caso de Symfony para PHP.

El Paquete de Gestión de Proyecto (GESPRO) es actualmente desarrollado utilizando Ruby on Rails. El análisis realizado anteriormente demuestra muchas de las ventajas que brinda este potente marco de trabajo y que justifican claramente la elección de su uso para el desarrollo de la herramienta que finalmente se obtendrá, la cual formará parte del GESPRO.

1.11. Entorno integrado de desarrollo

Un entorno integrado de desarrollo (IDE[9]) es una aplicación informática, utilizada para editar, compilar, depurar código y construir interfaces gráficas en el proceso de desarrollo de software. Un IDE puede ser una aplicación por si sola o puede ser parte de otra aplicación (27). Existen varios IDE de desarrollo para el lenguaje Ruby entre los que se encuentran Netbeans, Eclipse y Geany. A continuación se realiza un análisis de los más utilizados.

1.11.1. NetBeans

Netbeans es un Entorno de Desarrollo Integrado (IDE) está disponible libre de costo y de código abierto, escrito en lenguaje Java, lo cual lo convierte en multiplataforma. Tiene todas las herramientas necesarias para crear aplicaciones de escritorio y web profesionales, mediante el uso de diferentes lenguajes de programación como Java, C/C++, PHP, Ruby, JavaScript, HTML y Groovy. Entre sus características fundamentales se encuentra un editor de código muy potente, permite depurar y ejecutar programas, tiene integración con marcos de trabajo de desarrollo de PHP, Ruby entre otros, sus

[9] **IDE:** Acrónimo en inglés de Integrated Development Environment.

funcionalidad se pueden extender con el uso de complementos o extensiones que se le incorporan según las necesidades del programador, permite integración con sistemas de control de versiones (28). De manera general es un IDE muy escalable que se adapta a todas las necesidades por su alto nivel de configuración y personalización.

1.11.2. Eclipse

Eclipse es un completo y muy potente IDE, escrito Java, es gratuito y de código abierto. Permite desarrollar en diferentes lenguajes de programación tales como Java, PHP, C/C++, Python, Ruby, HTML entre otros, esto gracias a una de sus principales características, que permite adicionarle extensiones con el objetivo de incorporarle nuevas funcionalidad y características (29).

1.11.3. Fundamentación del entorno integrado de desarrollo a utilizar

De manera general ambos IDEs tanto Netbeans y Eclipse son una buena alternativa a considerar para llevar a cabo la implementación de la herramienta pero es necesario tomar en consideración cual de los dos ofrece más ventajas con el lenguaje de programación elegido para desarrollar la herramienta. La versatilidad de Netbeans y todas las características adicionales que ofrece para el uso de Ruby como lenguaje de programación elegido y Ruby on Rails como marco de trabajo para el de desarrollo, lo convierten en la mejor alternativa de ambos. Características como creación y depuración de proyectos con Ruby on Rails de manera sencilla, instalación y administración de gemas de Ruby y gran soporte para el trabajo con comandos y funciones de Ruby convierten a Netbeans en su versión 6.9 en la elección para llevar a cabo la implementación de la herramienta en cuestión.

1.12. Conclusiones del capítulo

Después de realizado el estudio bibliográfico se puede arribar a las siguientes conclusiones:

- Existen varios modelos y herramientas para la evaluación de proyectos pero no satisfacen completamente las necesidades del entorno de desarrollo de la UCI.

- Se selecciona como modelo de evaluación para el desarrollo de la herramienta el propuesto por la Ing. Marieta Peña en la propuesta de solución de su tesis de maestría “Modelo para análisis de factibilidad en la evaluación de Proyectos de Software”, lo que posibilita satisfacer las necesidades existentes actualmente en la UCI con respecto a los estudios de factibilidad de proyectos de software.

- Se hace necesaria la integración de la herramienta a desarrollar a la plataforma GESPRO, incorporando de esta manera los estudios de factibilidad al ciclo de vida de los proyectos.

- Para el desarrollo de la herramienta se utilizarán tecnologías libres, contribuyendo a la independencia tecnológica por la que aboga la UCI.

Capítulo 2: Descripción de la solución propuesta

2.1. Introducción

En el presente capítulo se describe de forma general el sistema a desarrollar, los roles que interactúan con el mismo. Además se hace una descripción detallada del desarrollo de la herramienta durante todas las fases que propone la metodología XP. Se generan todos los artefactos correspondientes a cada fase como es el caso de las historias de usuario y plan de iteraciones en la fase de planificación, las tarjetas clase responsabilidad colaborador (CRC) en la fase de diseño y finalmente en la fase de codificación se generan las tareas de programación y se obtiene el código funcional de la herramienta.

2.2. Objeto de informatización

Con el presente trabajo se pretende obtener una aplicación que será integrada al GESPRO como un módulo o extensión de este. Mediante la informatización del proceso de estudio de factibilidad de un proyecto informático en la UCI, se ayudará a los especialistas del área de la Gestión de Software en la toma de decisiones, posibilitando además asegurar en gran medida el éxito de los proyectos antes de acometerlos.

2.3. Personas relacionadas con el sistema

Las roles o actores que se relacionan con el sistema son aquellos que participan del proceso de evaluación de factibilidad, estos son:

Personas	Descripción
Administrador	Puede efectuar cualquier funcionalidad de sistema, tiene acceso a todo el proceso de evaluación.
Especialista	Es el que se encarga de realizar el proceso de evaluación, y controla todo el desarrollo de los procesos que estén relacionados con él.
Experto	Se encarga de evaluar los proyectos atendiendo a los criterios que sean seleccionados por el especialista, para ello antes debe realizar una evaluación de competencias.

Tabla#1. Personas relacionadas con el sistema

2.4. Fase de planificación

Para el desarrollo de la herramienta se dividirá el trabajo en cuatro fases como propone la metodología P*rogramación Extrema*.

En la fase de planificación se lleva a cabo el proceso enmarcado dentro de la ingeniería de requerimientos, se crean las historias de usuario para obtener una descripción de las funcionalidades con que debe cumplir el sistema, se listan los requerimientos del mismo para tener una mejor visión y finalmente se planifica el tiempo que durara el desarrollo mediante un plan de iteración.

2.4.1. Historias de usuarios

Durante la fase de planificación se generan las historias de usuario, que son los documentos de especificación funcional de una aplicación, son escritas por el cliente en su propio lenguaje con descripciones cortas de que debe hacer el sistema. A continuación se describen las principales historias de usuario del sistema a desarrollar de un total de 15, el resto se encuentra en los artefactos adjuntos al trabajo:

Historia de Usuario	
Código: HU #3	**Nombre Historias de usuario:** Gestionar fichas de proyectos
Modificación de Historia de Usuario Número: Ninguna	
Referencia: Rf-#4, Rf-#5, Rf-#6, Rf-#7	
Programador: Giorgy G. Cabrera	**Iteración Asignada:** primera
Prioridad: Muy Alta	**Puntos Estimados:** 7 días
Riesgo en Desarrollo: Alta	**Puntos Reales:** 7 días
Descripción: La funcionalidad gestionar fichas de proyectos, debe permitir crear, modificar, mostrar y eliminar una ficha de proyecto, del sistema. Inicialmente se debe mostrar una página donde aparezcan todas las fichas de proyectos que han sido adicionadas con anterioridad en una tabla, con la información del nombre oficial del proyecto, cliente, estado, prioridad, la información económica de este y las opciones para escoger las acciones que se	

puedan realizar sobre estas fichas de proyectos. Además esta página debe existir un buscador que permita filtrar las fichas de proyectos según los criterios que se muestran en la tabla y una opción para agregar una nueva ficha de proyecto.

Crear

Se debe mostrar una página para solicitar todos los datos necesarios de una ficha de proyectos.

Modificar

Cuando se selecciona la ficha de proyecto que se desea modificar, se muestra la información que esta contiene, permitiendo realizar cambios en esta y finalmente permitiendo actualizarla.

Mostrar

Cuando se selecciona la ficha de proyecto que se desea mostrar se muestra la información que esta contiene.

Eliminar

Se elimina la ficha de proyecto seleccionada, siempre preguntando la confirmación de la acción a realizar al usuario para evitar pérdidas de datos accidentales.

Observaciones:

- Las acciones sobre las fichas de proyectos solo pueden ser realizadas por un usuario, que tenga privilegios de especialista en el sistema.
- Algunos de los datos de la ficha de proyecto son seleccionables estos deben existir con anterioridad.

Tabla#2. Descripción de la historia de usuario HU #3

Historia de Usuario	
Código: HU #4	**Nombre Historias de usuario:** Gestionar proceso de evaluación
Modificación de Historia de Usuario Número: Ninguna	
Referencia: Rf-#9, Rf-#10, Rf-#11, Rf-#12	
Programador: Pedro F. Cadenas	**Iteración Asignada:** primera
Prioridad: Muy Alta	**Puntos Estimados:** 6 días

Riesgo en Desarrollo: Alta	**Puntos Reales:** 6 días

Descripción:

La funcionalidad gestionar proceso de evaluación debe permitir agregar, modificar, mostrar y eliminar procesos de evaluación. Inicialmente se debe mostrar una página donde aparezcan todos los procesos de evaluación que han sido adicionados con anterioridad, con el estado actual de la evaluación que puede ser inicio, evaluación, comunicación y concluido, el nombre del especialista que la creó, la fecha en la que fue creada, la fecha en la que finalizó en caso de que esté en este estado y las opciones para escoger las acciones que se puedan realizar sobre este proceso de evaluación. Además esta página debe existir un buscador que permita filtrar los procesos de evaluación según los criterios que se muestran y una opción para agregar un nuevo proceso de evaluación.

Crear

Se debe mostrar una página donde aparezcan los datos iniciales de un proceso de evaluación, finalmente se guardan los cambios.

Mostrar y Modificar

Cuando se selecciona el proceso de evaluación que se desea mostrar o modificar, se muestra la información que este contiene, permitiendo además realizar cambios en esta y finalmente actualizarlo. En esta página se deben mostrar las fichas de proyecto, los métodos a evaluar, los criterios y los expertos que actualmente tiene el proceso de evaluación permitiendo gestionar cada uno de ellos.

Eliminar

Se elimina el proceso de evaluación seleccionado, siempre preguntando la confirmación de la acción a realizar al usuario para evitar pérdidas de datos accidentales.

Observaciones:

- Las acciones de gestionar proceso de evaluación solo pueden ser realizadas por un usuario que tenga privilegios de especialista en el sistema.

Tabla#3. Descripción de la historia de usuario HU #4

<table>
<tr><th colspan="2">Historia de Usuario</th></tr>
<tr><td>Código: HU #11</td><td>Nombre Historias de usuario: Evaluar método AHP para la valoración de criterios técnicos y comerciales</td></tr>
<tr><td colspan="2">Modificación de Historia de Usuario Número: Ninguna</td></tr>
<tr><td colspan="2">Referencia: Rf-#29, Rf-#30</td></tr>
<tr><td>Programador: Pedro F. Cadenas</td><td>Iteración Asignada: segunda</td></tr>
<tr><td>Prioridad: Alta</td><td>Puntos Estimados: 7 días</td></tr>
<tr><td>Riesgo en Desarrollo: Medio</td><td>Puntos Reales: 5 días</td></tr>
<tr><td colspan="2">Descripción:
La funcionalidad evaluar método AHP para la valoración de criterios técnicos y comerciales debe permitir realizar el proceso de análisis jerárquico (AHP) para criterios técnicos y comerciales, a todas las fichas de proyecto que han sido agregadas a un proceso de evaluación. Inicialmente se debe seleccionar la evaluación la cual se va a evaluar. Luego se muestra una página en la que se puede seleccionar el método que se desea evaluar ya sea AHP y métodos económicos. Luego de seleccionado algunos de los métodos AHP, este proceso está dividido en varios pasos, las cuales deben ser realizados por cada uno de los expertos que participa en la evaluación estos pasos son:
1. Evaluar criterios contra criterios.
2. Evaluar proyectos contra proyectos atendiendo a cada criterio.
3. Estimar las ponderaciones o pesos relativos de los elementos de la decisión.
4. Comprobar la consistencia de los juicios de los expertos.
5. Cálculo de la prioridad global en el conjunto de proyectos.
Finalmente el resultado de cada uno de los expertos se unen y se ordenan para decidir cuales proyectos son más factibles.</td></tr>
<tr><td colspan="2">Observaciones:
• Las acciones de evaluar método AHP para la valoración de criterios técnicos y comerciales, solo pueden ser realizadas por un usuario, que sea experto en la</td></tr>
</table>

evaluación, este proceso solo se podrá realizar una vez.

- Es necesario asegurar para que se pueda realizar la evaluación del método AHP para la valoración de criterios técnicos y comerciales en el proceso de evaluación, que el proceso de evaluación esté en estado de evaluación, el experto que vaya a evaluar, haya realizado su evaluación de competencia y haya obtenido en esta nivel medio o alto, que los método AHP comercial o técnicos estén agregados en el proceso de evaluación y que existan más de un criterio de cada tipo por método AHP agregado y más de un proyecto a evaluar.

Tabla#4. Descripción de la historia de usuario HU #11

Historia de Usuario	
Código: HU #12	**Nombre Historias de usuario:** Evaluar métodos FC,TIR,VAN y PRI para la valoración de criterios económicos
Modificación de Historia de Usuario Número: Ninguna	
Referencia: Rf-#31, Rf-#33, Rf-#34, Rf-#35	
Programador: Giorgy G. Cabrera	**Iteración Asignada:** segunda
Prioridad: Alta	**Puntos Estimados:** 7 días
Riesgo en Desarrollo: Medio	**Puntos Reales:** 7 días
Descripción: La funcionalidad evaluar métodos FC, TIR, VAN y PRI para la valoración de criterios económicos, debe permitir calcular cada uno de los método económicos para todas las fichas de proyecto que han sido agregadas a un proceso de evaluación, atendiendo a los flujos iniciales, operacionales y finales del flujo de caja (FC) de cada una. Inicialmente se debe seleccionar la evaluación en la cual se va a realizar el proceso. Luego se muestra una página en la que se puede seleccionar el método que se desea evaluar ya sea AHP y métodos económicos. Luego de seleccionado algunos de los métodos económicos, se muestra la página correspondiente a cada uno de ellos para la ficha de proyecto seleccionada.	

FC: Se calcula de manera automática tomando los flujos iniciales, operacionales y finales del flujo de caja. **VAN:** Se debe mostrar una página para entrar los datos necesarios para el cálculo de este método económico. Tomando además el resultado anterior del FC. **TIR:** Se debe mostrar una página para entrar los datos necesarios para el cálculo de este método económico. Tomando además el resultado anterior del VAN. **PRI:** Se debe mostrar una página para entrar los datos necesarios para el cálculo de este método económico.
Observaciones: • Las acciones de evaluar métodos FC, TIR, VAN y PRI para la valoración de criterios económicos, solo pueden ser realizadas por un administrador o un especialista, este proceso solo se podrá realizar una vez. • Es necesario asegurar para que se pueda realizar la evaluación del los métodos económicos en el proceso de evaluación, el mismo esté en estado de evaluación, que los métodos económicos estén agregados en el proceso de evaluación, que las fichas de proyectos que se vayan a evaluar por los métodos económicos tengan las información económico agregada.

Tabla#5. Descripción de la historia de usuario HU #12

2.4.2. Lista de reserva del producto

A través de la lista de reserva del producto se definen y priorizan las funcionalidades que tendrá el sistema, además se describen los requisitos no funcionales que tendrá el software. Aunque los requisitos funcionales y no funcionales no forman parte de los artefactos que se generan en la metodología XP, se considera que una descripción detallada de estos podría, junto a las historias de usuario facilitar el desarrollo de la herramienta.

2.4.2.1. Requisitos funcionales

Los requisitos funcionales expresan la esencia y definen el funcionamiento del software, establecen como el sistema debe reaccionar a una entrada en particular y como debe comportarse ante tal situación, es el conjunto de funcionalidades que va a realizar la aplicación para automatizar un proceso determinado (30). Los requisitos funcionales a

implementar hacen un total de 89 y se pueden encontrar dentro de los artefactos adjuntos al documento de tesis, a continuación se listan algunos de los principales:

Código	Descripción	Prioridad
	Gestionar usuarios como especialistas	Muy Alta
Rf-#1	Agregar especialista	Muy Alta
Rf-#2	Eliminar especialista	Muy Alta
Rf-#3	Buscar usuario	Muy Alta
	Gestionar fichas de proyectos	Muy Alta
Rf-#4	Crear ficha de proyecto	Muy Alta
Rf-#5	Modificar ficha de proyecto	Muy Alta
Rf-#6	Eliminar ficha de proyecto	Muy Alta
	Gestionar proceso de evaluación	Muy Alta
Rf-#9	Crear evaluación	Muy Alta
Rf-#10	Modificar evaluación	Muy Alta
Rf-#11	Eliminar evaluación	Muy Alta
Rf-#12	Buscar evaluación	Muy Alta
	Gestionar criterios a evaluar durante una evaluación	Muy Alta
Rf-#13	Agregar criterio de evaluación al proceso evaluativo	Muy Alta
Rf-#14	Eliminar criterio de evaluación del proceso evaluativo	Muy Alta
Rf-#15	Buscar criterio de evaluación	Muy Alta
	Gestionar fichas de proyectos a evaluar durante una evaluación	Muy Alta
Rf-#16	Agregar proyectos al proceso evaluativo	Muy Alta
Rf-#17	Eliminar proyectos del proceso evaluativo	Muy Alta
Rf-#18	Buscar proyectos a evaluar	Muy Alta
	Gestionar métodos a evaluar durante una evaluación	Muy Alta
Rf-#19	Agregar método de evaluación al proceso evaluativo	Muy Alta
Rf-#20	Eliminar método de evaluación del proceso evaluativo	Muy Alta
Rf-#21	Buscar método de evaluación	Muy Alta
	Gestionar expertos que participan en una evaluación	Muy Alta
Rf-#22	Agregar experto al proceso evaluativo	Muy Alta

Rf-#23	Eliminar experto del proceso evaluativo	Muy Alta
Rf-#24	Buscar experto	Muy Alta
	Realizar evaluación de proyectos	Alta
Rf-#29	Evaluar Método AHP para la valoración de criterio técnicos	Alta
Rf-#30	Evaluar Método AHP para la valoración de criterios comerciales	Alta
Rf-#31	Evaluar Método de Flujo Neto de Caja (FC)	Alta
Rf-#33	Evaluar Método Valor Actual Neto (VAN)	Alta
Rf-#34	Evaluar Método Tasa Interna de Rendimiento (TIR)	Alta
Rf-#35	Evaluar Método Período de Recuperación de la Inversión(PRI)	Alta
Rf-#36	Aplicar Sistema de inferencia borroso para el análisis de factibilidad	Alta
	Generar documentos después de evaluación	Media
Rf-#41	Generar listado ordenado de proyecto	Media

Tabla#6. Requisitos funcionales

2.4.2.2. Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que los servicios o funciones del sistema a desarrollar deben poseer al final de su elaboración. Estas propiedades no están directamente relacionadas con las funcionalidades que se derivan del negocio. Le confieren al producto final ciertas características como usabilidad, rapidez, confiabilidad y seguridad. Por lo general son aplicado al sistema en su totalidad (30). Los requisitos no funcionales de la aplicación a desarrollar son:

Código	Clasificación
	Apariencia o Interfaz Externa
RNF-#1	La interfaz de usuario en cada una de las páginas de la aplicación debe ser sencilla, intuitiva, que le permita al usuario que interactúa con el sistema realizar cualquier labor en el menor tiempo posible y con pocas acciones.
RNF-#2	Las interfaces contarán con menús que le permitan al usuario acceder desde cualquier lugar de la aplicación a otro lugar que desee con la mínima cantidad de acciones.
RNF-#3	En el momento que ocurra un error con los datos de entrada a la aplicación le

	será informado al usuario de manera detallada.
RNF-#4	La aplicación al ser una extensión del GESPRO debe mantener el diseño y los estilos que este utiliza, con el propósito de mantener la uniformidad en todo el sistema.
	Usabilidad
RNF-#5	Debe satisfacer las necesidades del usuario, proporcionando una funcionalidad adecuada de forma que satisfaga los requisitos funcionales que se plantearon inicialmente
RNF-#6	La aplicación debe tener una interfaz sencilla, que permita a los usuarios que la utilicen acceder a la información que solicita de forma simple, en un tiempo breve y desde cualquier parte de esta.
RNF-#7	Los servidores en los que estará alojada la aplicación tendrán la capacidad de procesamiento necesaria, para brindarle siempre la mejor experiencia al usuario, en el momento de utilizar la aplicación.
	Fiabilidad
RNF-#8	Para asegurar la consistencia de los datos que se procesan en la aplicación y la fiabilidad de estos, se restringirá el nivel de acceso por roles y usuarios al gestor de base de datos y a la aplicación.
RNF-#9	La aplicación deberá estar disponible siempre, asegurando el acceso en todo momento desde cualquier lugar de red a todos los usuarios que estén autorizados.
	Eficiencia
RNF-#10	Los tiempos de respuestas en el procesamiento de datos y solicitud de estos en el sistema tienen que ser breves. En las ocasiones que el sistema esté realizando mayor carga de trabajo por encontrarse procesando y almacenando datos, estos tiempos no deben exceder de los 5 segundos.
	Seguridad
RNF-#11	La aplicación debe estar protegida mediante una política de usuarios y roles que no permitan el acceso no autorizado a esta. De esta forma se asegura la integridad y confiabilidad de los datos que en esta se procesan.
RNF-#12	A cada usuario se le debe mostrar y dar acceso a las funcionalidades a las

	cuales está autorizado a acceder. Asegurando el acceso a los datos que corresponde para cada usuario en el sistema.
	Software
RNF-#13	En el servidor de aplicación para el correcto funcionamiento de la aplicación es necesario tener instalado alguna distribución del Sistema Operativo GNU/Linux, el servidor web Apache Phusion Passenger, Marco de trabajo Ruby on Rails.
RNF-#14	El servidor de datos, debe estar instalado el Gestor de Base de Datos PostgreSQL 8.4 o superior.
RNF-#15	Para el uso en las PC clientes es necesario tener instalado un navegador Web (Mozilla Firefox, Google Chrome Opera, Internet Explorer). Se recomienda Mozilla Firefox 8.0 o superior.
	Hardware
RNF-#16	Para garantizar un buen desempeño de la aplicación en los servidores se necesita una computadora con 2 GB de memoria RAM o superior, un disco duro 320 GB o superior por el volumen de datos que se generan, un procesador Dual Core 2.0 GHz o superior.
RNF-#17	En las PC clientes se necesitan para garantizar un funcionamiento correcto 256 MB de memoria RAM mínimo, un procesador Pentium 3 o superior.
	Restricciones de diseño y la implementación
RNF-#18	Para el desarrollo de la aplicación se deberá utilizar un conjunto de herramientas y tecnología. Como lenguaje de programación se utilizará Ruby con el marco de trabajo Ruby on Rails, el IDE de desarrollo Netbeans 6.9, el servidor web Apache Phusion Passenger, como Gestor de Base Datos PostgreSQL. La metodología de software a utilizar XP y para el modelado se utilizará como herramienta CASE Visual Paradigm. Además como Sistema Operativo para el desarrollo se utilizará una distribución de GNU/Linux.

Tabla#7. Requisitos no funcionales

2.4.3. Plan de iteraciones

Después de identificar las historias de usuario y estimar el esfuerzo para la realización de las mismas se prosigue a confeccionar el plan de iteración para la planificación de la etapa de implementación del sistema. Este plan define cuales historias de usuario serán implementadas en cada iteración y que tiempo se estima que durará el desarrollo de cada una de ellas. El sistema será desarrollado en las siguientes tres iteraciones:

Primera: En esta iteración se van a implementar las historias de usuario que tienen una prioridad muy alta para el negocio.

Segunda: En esta iteración se van a implementar las historias de usuario que tienen una prioridad alta para el negocio.

Tercera: En esta iteración se van a implementar las historias de usuario que tienen una prioridad media para el negocio.

Iteración	Orden de la HU a implementar	Duración de cada HU(días)	Duración Total (semanas)
1ra	Gestionar especialistas	3	5
	Gestionar fichas de proyectos	7	
	Generar ficha para solicitar los flujos iniciales, operacionales y finales del FC	7	
	Gestionar proceso de evaluación	6	
	Gestionar criterios a evaluar durante una evaluación	3	
	Gestionar fichas de proyectos a evaluar durante una evaluación	3	
	Gestionar métodos a evaluar durante una evaluación	3	
	Gestionar expertos que participan en una evaluación	3	

2da	Realizar evaluación de expertos	7	4
	Evaluar método AHP para la valoración de criterios técnicos y comerciales	7	
	Evaluar métodos FC,TIR,VAN y PRI para la valoración de criterios comerciales	7	
	Aplicar sistema de inferencia borroso para el análisis de factibilidad	7	
3ra	Cargar proyecto desde archivo CSV	7	3
	Generar listado ordenado de proyecto	7	
	Gestionar nomencladores	7	
Total			12

Tabla#8. Plan de iteración

2.5. Fase de diseño

Durante la fase de diseño se confeccionan las tarjetas clase responsabilidad colaborador para la descripción de cada una de las entidades, se realiza el diagrama de clases para tener mejor visión de la estructura del sistema y se diseña el modelo de datos.

2.5.1. Tarjetas clase-responsabilidad-colaborador (CRC)

Durante la fase de diseño se elaboran las tarjetas clase-responsabilidad-colaborador (CRC). El uso de este tipo de tarjetas es una técnica de modelado que permite identificar las clases, sus atributos y responsabilidades. El objetivo es obtener un diseño simple, elegante y fácil de comprender por parte de los programadores. A continuación se muestran algunas de las tarjetas del sistema a desarrollar.

Especialista	
Descripción: Guarda información de los especialistas que dirigen el proceso	
Atributos:	
Nombre	**Descripción**
id	Campo identificador
login	
firstname	
lastname	
mail	
Responsabilidades	
Nombre	**Colaborador**
save	
delete	
update	
find	

Tabla#9. Tarjeta CRC "Especialista"

Experto	
Descripción: Guarda información de los expertos que evalúan los proyectos	
Atributos:	
Nombre	**Descripción**
id	Campo identificador
login	
firstname	
lastname	
mail	
nivel_id	
Responsabilidades	
Nombre	**Colaborador**
save	
delete	
update	
find	

Tabla#10. Tarjeta CRC "Experto"

ProcesoEvaluacionController	
Descripción: Se encarga de controlar las evaluaciones a realizar	
Atributos:	
Nombre	**Descripción**
Responsabilidades	
Nombre	**Colaborador**
Nuevo	
Crear	
Mostrar	
Eliminar	Metodo, Experto, Criterio, Proyecto
buscar_evaluacion	
cambiar_estado	Estado
agregar_metodos	Metodo
eliminar_metodos	Metodo
agregar_proyectos	Proyecto
eliminar_proyectos	Proyecto
agregar_expertos	Expertos
eliminar_expertos	Expertos
agregar_criterios	Criterio
eliminar_criterios	Criterio

Tabla#11. Tarjeta CRC "ProcesoEvaluacionController"

2.5.2. Diagrama entidad a partir de tarjetas CRC

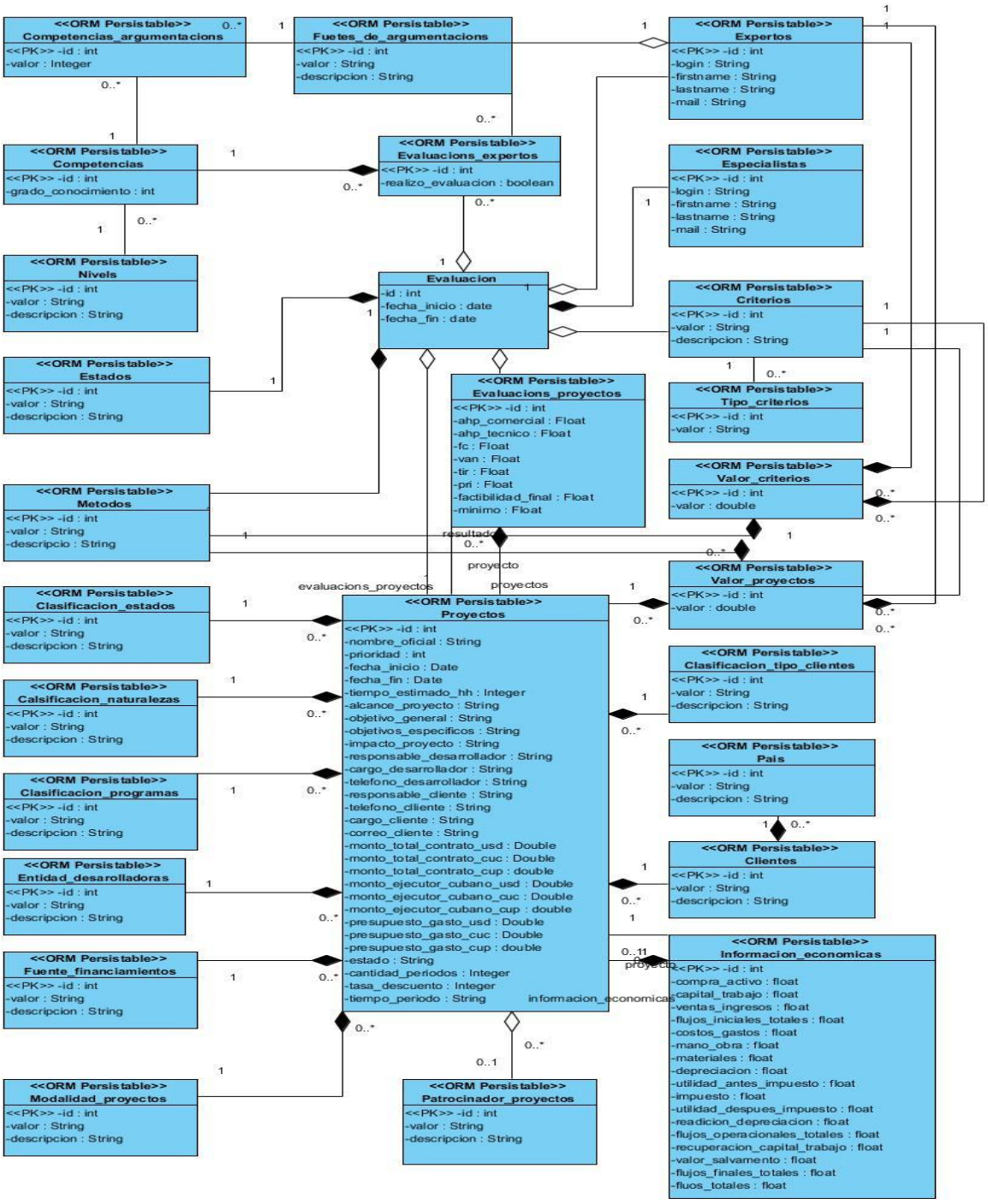


Figura 4. Diagrama Entidad

2.5.3. Modelo de datos

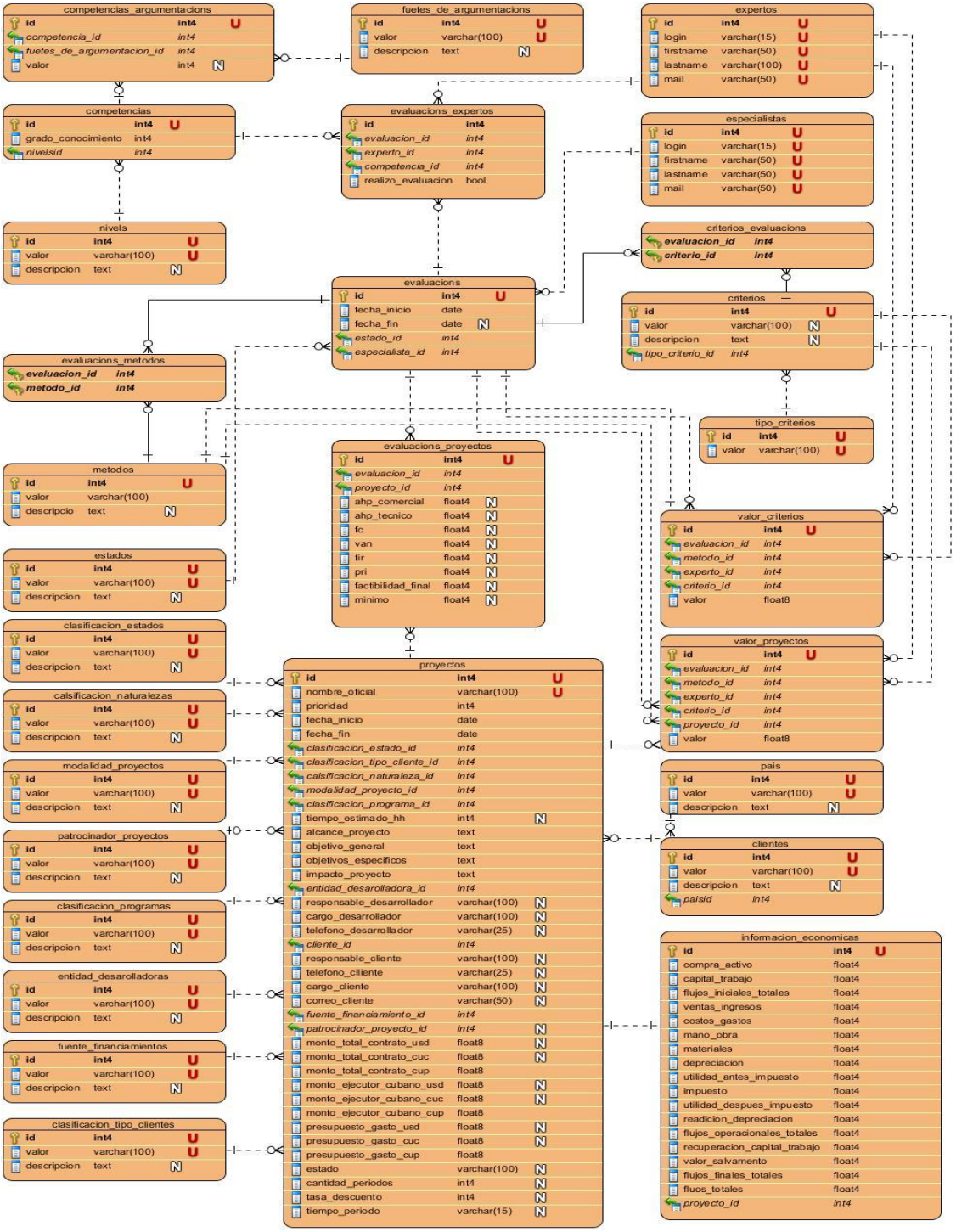


Figura 5. Modelo de Datos

2.5.4. Patrones de diseño

Un patrón de diseño consiste en la descripción de un problema y su solución, el cual recibe un nombre y puede emplearse en otros contextos. Indica o puede tener una sugerencia de cómo aplicarlo y utilizarlos en otros contextos y en situaciones nuevas (31).

Un patrón está compuesto por cuatro elementos fundamentales:

El nombre: Es un identificador que se utiliza para describir un problema de diseño, sus soluciones y consecuencias en una o dos palabras.

El problema: Describe cuando se utiliza un patrón.

La solución: Describe los elementos que componen al diseño.

Las consecuencias: Son los resultados y las compensaciones de aplicar el patrón.

En el diseño de un sistema es muy importante asignar correctamente las responsabilidades, para acometer esta tarea de una forma eficaz se hace uso del los patrones de diseño, específicamente el caso de los patrones GRASP[10]. Estos últimos describen los principios fundamentales de asignación de responsabilidades a objetos, expresados en forma de patrones (31).

El marco de trabajo Ruby on Rails utilizado para desarrollar la aplicación en cuestión hace uso de los cinco patrones GRASP que más se utilizan es el caso de los patrones Experto, Creador, Alta Cohesión, Bajo Acoplamiento y Controlador. A continuación se explica cada uno de ellos.

Experto: Este patrón plantea que la responsabilidad debe ser asignada al experto en la información, es decir a la clase que cuenta con la información necesaria para cumplir esta

[10] **GRASP:** es un acrónimo en inglés de General Responsibility Asignment Software Patterns (patrones generales de software para asignar responsabilidades).

responsabilidad. Esto lo vemos presente en Ruby on Rails mediante el uso de ORM[11], el cual le asigna a cada clase entidad la responsabilidad de manejar cada uno de los objetos que son creados, pues son estas las que tienen toda la información necesaria de cada uno de los objetos que se manejan en la aplicación. Esto permite conservar el encapsulamiento y contribuye al bajo acoplamiento.

Creador: Este patrón plantea asignarle a la clase *B* la responsabilidad de crear una instancia de la clase *A*. Ruby on Rails en cada uno de las clases controladoras que crea para cada uno de los módulos o funcionalidades de la aplicación, se encarga de la creación de las instancias de las clases que describen los objetos que en ellos se manejan, ya que estos o agregan, contienen, registran o utilizan las instancias de estos objetos. Esto favorece la reutilización y favorece el bajo acoplamiento.

Alta Cohesión: Plantea asignar una responsabilidad de manera que la cohesión siga siendo alta. El marco de trabajo Ruby on Rails favorece la alta cohesión asignado responsabilidades a las clases de manera que estas estén estrechamente relacionadas entre sí y sin llegar a realizar un trabajo enorme o excesivo.

Bajo Acoplamiento: Está estrechamente relacionado con los patrones Experto o Alta Cohesión y plantea la baja dependencia que debe existir entre las clases. Ruby on Rails favorece el bajo acoplamiento de las clases en el sistema ya que asigna las responsabilidades a cada una de las clases de manera que estas no dependan en gran medida de otras, abogando por la independencia. Esto favorece el mantenimiento futuro de la aplicación, la reutilización y la escalabilidad del sistema.

Controlador: Ruby on Rails utiliza el patrón arquitectónico Modelo-Vista-Controlador por lo que tiene una capa específica designada para los controladores. Las clases controladoras son un ejemplo de estas y cada una de ellos son las encargadas de manejar los eventos y mensajes que se generan en la aplicación por ejemplo entre la capa de presentación y el modelo.

[11] **ORM:** Acrónimo en inglés de Object Relational Mapping.

Otro conjunto de patrones que son utilizados por el marco de trabajo Ruby on Rails, son los llamados patrones GoF[12] . Dentro de este grupo se utilizan específicamente Solitario y Decorador.

Solitario: Este garantiza que solamente se cree una instancia de la clase y proporciona un punto de acceso global a este objeto. Un ejemplo de este se pone de manifiesto en la clase *User* la cual en cualquier lugar de la aplicación brinda acceso a la información del usuario que está autenticado en el sistema.

Decorador: Este patrón posibilita extender la funcionalidad de un objeto dinámicamente de tal modo que sea transparente a sus clientes. Este ejemplo lo podemos observar en el archivo *estilo.html.erb*, el cual tiene como función decorar con los estilos y con código html, todas las plantillas de las páginas que se utilizan en la aplicación, extendiendo funcionalidades a la todas las vistas de una forma dinámica y con solo realizar la modificación el archivo *estilo.html.erb*.

2.5.5. Arquitectura utilizada

Existen varios patrones arquitectónicos para el desarrollo de software entre estos están el modelo-vista-controlador y la arquitectura en capas. Su elección está condicionada por las necesidades del sistema a desarrollar y los problemas que surjan durante su desarrollo. El marco de trabajo Ruby on Rails el cual será utilizado para el desarrollo de la herramienta, está basado en la arquitectura Modelo-Vista-Controlador (MVC en lo adelante) para el desarrollo de aplicaciones web, por lo que este será al patrón arquitectónico a utilizar (32). Este patrón separa cada una de las capas de la aplicación, haciéndolas independientes una de la otras, facilitando el mantenimiento y la escalabilidad del sistema.

[12] **GoF:** Acrónimo en inglés de Gang of Four (pandilla de los cuatro). Se conoce con este nombre pues el libro en el que se describen por primera vez estos patrones, fue escrito por cuatro autores.

El modelo: Es el encargado de mantener el estado de la aplicación o sea el responsable de la persistencia de los datos que esta maneja. Esta capa es la encargada de hacer cumplir todas las reglas del negocio que se le aplican a estos datos. En esta capa se encuentran todas las clases entidades de la aplicación y el manejo de datos se realiza mediante un ORM.

La vista: Es la encargada de generar las vistas de la aplicación generalmente muy estrechamente relacionados con los datos del modelo. En esta capa se encuentras todas las páginas o interfaces que se utilizan en la aplicación para la interacción con el usuario.

El controlador: Es el encargado de organizar la aplicación, es el intermediario de la comunicación entre la capa del modelo y la vista. Es esta capa la que recibe todos los eventos de la vista, interactúa con el modelo para satisfacer estos eventos y finalmente mostrar una vista con la respuesta esperada por el usuario.

Específicamente en Ruby on Rails el funcionamiento de este patrón es de la siguiente manera. Primeramente la solicitud entrante se envía al enrutador, que resuelve a donde la solicitud debe ser enviada dentro de la aplicación, seleccionando el controlador, la acción que se realizará dentro de este, lo cual puede llevar una interacción con el modelo y finalmente muestra la vista que espera el usuario con los datos correspondientes. A continuación se muestra el patrón MVC en el marco de trabajo Ruby on Rails.

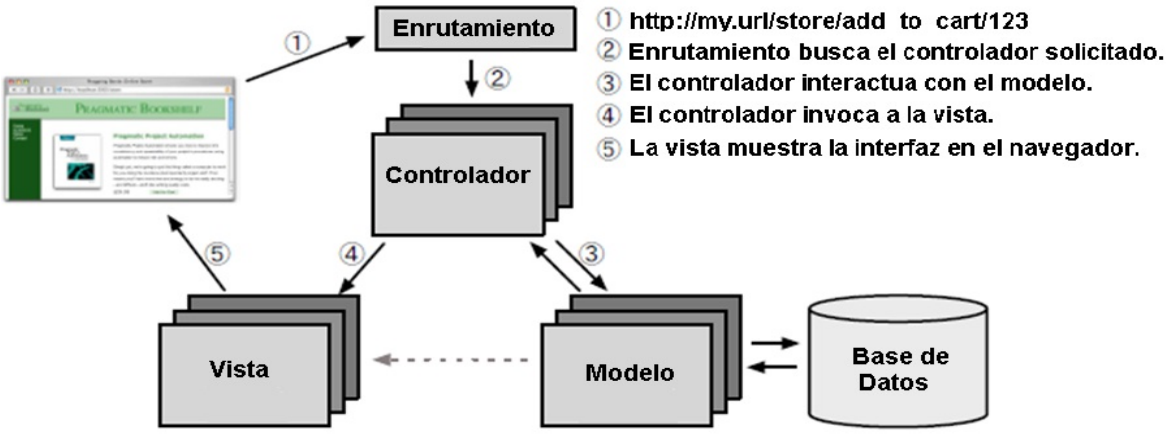


Figura 7. Patrón Modelo-Vista-Controlador en Ruby on Rails

2.6. Fase de codificación

En esta fase se planifican y ejecutan las tareas de programación, durante la misma se codifica todo el sistema diseñado obteniendo como resultado la herramienta.

2.6.1. Tareas de programación

Las tareas de programación son actividades sencillas que se derivan de las historias de usuario para simplificar la implementación de las mismas, se plasman en tarjetas de papel donde se describe que se debe realizar y son muy dinámicas y flexibles, pueden ser cambiadas por otras más generales o más específicas, agregarse nuevas o modificarse según las necesidades existentes. Cada una de estas tareas podrá ser comprobada a través de los casos de prueba.

A continuación se relacionan las tareas a desarrollar para la implementación de la herramienta para evaluar factibilidad propuesta, y la descripción de alguna de ellas, el resto puede ser consultada en los artefactos adjuntos al documento de tesis.

Historia de Usuario	Tareas de programación
Gestionar especialistas	• Agregar especialistas a partir de los usuarios del GESPRO • Eliminar especialistas existentes
Gestionar nomencladores	• Agregar cada uno de los elementos que pueden ser listados en nomencladores • Modificar cada uno de los elementos que pueden ser listados en nomencladores • Eliminar cada uno de los elementos que pueden ser listados en nomencladores
Gestionar fichas de proyectos	• Agregar fichas de proyecto • Crear formulario ficha de proyecto • Modificar y eliminar fichas de proyecto
Generar ficha para solicitar los flujos iniciales, operacionales y finales del FC	• Estudiar qué informaciones económicas son necesarias para el flujo de caja • Agregar informaciones económicas • Crear formulario para flujos operacionales
Gestionar proceso de evaluación	• Mostrar interfaz con los procesos de evaluación realizados

	• Buscar procesos de evaluación • Enviar alertas
Gestionar criterios a evaluar durante una evaluación	• Mostrar interfaz con los criterios a evaluar
Gestionar fichas de proyectos a evaluar durante una evaluación	• Mostrar interfaz con las fichas de los proyectos a evaluar
Gestionar métodos a evaluar durante una evaluación	• Mostrar interfaz con los métodos a evaluar
Gestionar expertos que participan en una evaluación	• Mostrar interfaz con los expertos a evaluar
Realizar evaluación de competencia de un expertos	• Crear formulario para solicitar los datos necesarios para la evaluación • Estudiar métodos para evaluar expertos • Desarrollar los métodos para evaluar los expertos
Evaluar método AHP para la valoración de criterios técnicos y comerciales	• Mostrar Interfaz para ponderar criterio y métodos • Procesar datos de las ponderaciones
Evaluar métodos FC,TIR,VAN y PRI para la valoración de criterios económicos	• Realizar flujo de caja • Realizar tasa interna de retorno • Calcular el valor actual neto • Calcular el periodo de recuperación de la inversión • Sumar resultados de métodos económicos
Aplicar sistema de inferencia borroso para el análisis de factibilidad	• Construir conjuntos borrosos • Construir variables lingüísticas • Construir reglas de inferencia • Realizar clasificación
Cargar fichas de proyectos desde archivo CSV	• Crear panilla • Cargar fichero
Generar listado ordenado de proyecto	• Generar listado • Imprimir y exportar listado
Generar acta de evaluación	• Generar acta • Imprimir y exportar acta

Tabla#12. Tareas de programación por cada historia de usuario

Tareas para la HU# 3: Inscribir Ficha de Proyectos

Tarea	
Número Tarea: HU#3-1	**Historia de Usuario:** HU #3- Gestionar fichas de proyectos
Nombre Tarea: Agregar fichas de proyectos	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 3
Fecha Inicio: 05/02/2012	**Fecha Fin:** 08/02/2012
Programador Responsable: Giorgy G. Cabrera	
Descripción: Se muestran las fichas que ya han sido agregadas y la opción de agregar nuevas	

Tabla#13. Descripción de la Tarea de Programación HU#3-1

Tarea	
Número Tarea: HU#3-2	**Historia de Usuario:** HU #3- Gestionar fichas de proyectos
Nombre Tarea: Crear formulario ficha de proyecto	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 2
Fecha Inicio: 08/02/2012	**Fecha Fin:** 10/02/2012
Programador Responsable: Giorgy G. Cabrera	
Descripción: Se debe crear un formulario con todos los campos para recoger la información necesaria de los proyectos	

Tabla#14. Descripción de la Tarea de Programación HU#3-2

Tarea	
Número Tarea: HU#3-3	**Historia de Usuario:** HU #3- Gestionar fichas de proyectos
Nombre Tarea: Modificar y eliminar fichas de proyecto	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 2
Fecha Inicio: 10/02/2012	**Fecha Fin:** 12/02/2012
Programador Responsable: Giorgy G. Cabrera	
Descripción: Se muestran las fichas que ya han sido agregadas y la opción de modificar la información que hay en las mismas, además también se debe brindar la opción de eliminarlas.	

Tabla#15. Descripción de la Tarea de Programación HU#3-3

Tareas para la HU# 11: Evaluar método AHP para la valoración de criterios técnicos y comerciales

Tarea	
Número Tarea: HU#11-1	**Historia de Usuario:** HU #11- Evaluar método AHP para la valoración de criterios técnicos y comerciales
Nombre Tarea: Mostrar Interfaz para ponderar criterio y métodos	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 2
Fecha Inicio: 21/03/2012	**Fecha Fin:** 23/03/2012
Programador Responsable: Pedro F. Cadenas	
Descripción: Se debe mostrar una interfaz de usuario que permita al experto realizar una ponderación de los criterio y después de realizado esto una ponderación de los proyectos comparándolos unos con otros atendiendo a cada criterio, los datos de las comparaciones deben ser guardados para su posterior procesamiento.	

Tabla#16. Descripción de la Tarea de Programación HU#11-1

Tarea	
Número Tarea: HU#11-2	**Historia de Usuario:** HU #11- Evaluar método AHP para la valoración de criterios técnicos y comerciales
Nombre Tarea: Procesar datos de las ponderaciones	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 5
Fecha Inicio: 23/03/2012	**Fecha Fin:** 28/03/2012
Programador Responsable: Pedro F. Cadenas	
Descripción: Se debe realizar los métodos que se encarguen de procesar la información arrojada por los expertos, de manera que se obtenga un vector de calificación global el cual será guardado como el resultado de la calificación de un experto para los proyectos que estén evaluando en ese momento	

Tabla#17. Descripción de la Tarea de Programación HU#11-2

2.6.2. Estándares de codificación

El uso de estándares o reglas de codificación al comenzar la implementación de la herramienta, trae para el producto final muchos beneficios. Entre estos se pueden mencionar que se genera un código muy legible, muy fácil de mantener y con un alto rendimiento a la hora de ejecutarse. El marco de trabajo Ruby on Rails tiene definido un conjunto de buenas prácticas para el código que se genera en este, las cuales se

mencionan a continuación, estas fueron tomadas del libro *Web Agile Development with Rails* (29):

Nombres de variables, parámetros de métodos y nombre de métodos: Todos los nombres deben comenzar con letra minúscula o con un guión bajo por ejemplo *evaluación_proyectos* o *xr2000* también es válido. Las variables de instancias comienzan con el símbolo de @ como por ejemplo *@proyectos* o *@expertos*. Ruby tiene como convención que los nombre compuestos por varias palabras deben ser separado por el guión bajo como *evaluación_proyectos*, *proceso_evaluacion*. Los nombre de los métodos estarán precedidos por la palabra *def* y los parámetros siempre estarán entre paréntesis.

Nombres de clases, módulos y contantes: Todos los nombres deben comenzar con mayúsculas, se utiliza la mayúscula por convención, en lugar del guión bajo para distinguir el inicio de la palabra dentro del nombre. Los nombres de clases deben verse como objetos por ejemplo *EvaluacionsExperto* y *FuentesDeArgumentacion* y siempre está presidido por la palabra *class*. De igual manera el nombre de los módulos estará precedido por la palabra *module*.

Uso de símbolos: Ruby on Rails hace extensivo el uso de símbolos. El nombre de un símbolo parece el nombre de una variable pero este está precedido por los dos puntos. Un ejemplo de esto es *:id*, *:evaluación_id* o *:conditions*. Rails hace uso los símbolos generalmente para nombrar los parámetros de un método o para buscar llaves en las tablas hash, ejemplo de esto es *redirect_to :action => "editar" , :id => params[:id]*

Líneas y comentarios de códigos: Ruby no necesita punto y coma final para decir que una línea de código terminó si estas están en líneas separadas. Las líneas de código tendrán siempre un máximo de 85 caracteres para facilitar comprensión de esta. Para comentar el código o realizar algún comentario del mismo, se utiliza el símbolo de número *(#)* por ejemplo:

```
#Vamos a imprimir Hola Mundo
puts “Hola mundo”
puts “en Ruby on Rails”
```

Estructuras de control: Ruby utiliza todas las estructuras de control usuales como *if, while, for, foreach.* Todas estas estructuras terminan con la palabra end para indicar el final del bloque de instrucciones. Por ejemplo:

```
If cont > 10
   puts "Intente nuevamente"
elsif cont < 5
   while cont < 9
      cont+= 1
   end
end
```

2.7. Conclusiones del capítulo

En este capítulo se arribaron a las siguientes conclusiones:

- Se definió la propuesta de solución del problema, detallándola con la ayuda de los artefactos propuestos por la metodología programación extrema, lo cual posibilitó organizar todo el proceso de desarrollo.

- Se describió el flujo de procesos para extraer las funcionalidades que debe poseer el sistema y se describieron las mismas mediante historias de usuarios, de las cuales se elaboró el plan de iteraciones para definir el momento en el que serán implementadas y cuánto durará su desarrollo.

- Se realizó una descripción de la fase de diseño donde se elaboraron las tarjetas CRC, se realizó un análisis de los patrones de diseño y arquitectónicos de los cuales hace uso el marco de trabajo Ruby on Rails, lo cual facilitó el desarrollo de la herramienta.

- Finalmente durante la fase de codificación se elaboraron las tareas de programación, derivadas de las historias de usuario y para estandarizar el código generado por el equipo de desarrollo en esta fase, se definieron los estándares de código a utilizar, obteniéndose finalmente la herramienta deseada acorde a los requisitos iniciales.

Capítulo 3: Validación y prueba de la solución

3.1 Introducción

En el presente capítulo se describe la validación de la herramienta desarrollada, esto se hará mediante las pruebas de software, con el objetivo de lograr la aceptación del cliente y comprobar que se alcanzaron los objetivos enunciados inicialmente.

3.2. Fase de Pruebas

Una de las principales fortalezas de la metodología de desarrollo XP es el proceso de prueba, el cual se realiza de manera continua para asegurar durante todo el proceso de desarrollo, el éxito del producto que se está elaborando. Esto permite elaborar un software de más calidad ya que los errores son detectados en un plazo de tiempo corto y se corrigen de una manera más sencilla.

XP divide las pruebas de software o de sistema en dos grupos, las pruebas unitarias y las pruebas de aceptación. Las pruebas unitarias son las encargadas de verificar el código y estas son diseñadas por los programadores. Mientras que las pruebas de aceptación o pruebas funcionales están destinadas a evaluar si al final de una iteración se consiguió la funcionalidad que se esperaba y que esta esté en función de los requisitos establecidos inicialmente, estas pruebas usualmente son diseñadas por el usuario o cliente final.

El objetivo fundamental que tienen las pruebas de software, es verificar los requisitos del sistema, por lo que son los propios requisitos la principal fuente de información a la hora de construir las pruebas del sistema(33).

3.2.1. Pruebas unitarias

Las pruebas unitarias o pruebas de caja blanca se basa en realizar pruebas al código del sistema. Para llevar a cabo esta tarea se comprueban los caminos lógicos de la aplicación mediante casos de prueba, que pongan a prueba los algoritmos implementados. Las pruebas unitarias no se le puede realizar a todo el código de la aplicación, ya que el número de caminos lógicos puede llegar a crecer de manera exponencial lo cual imposibilita realizar casos de pruebas para todo estos caminos y muchos menos se

podrían procesar todos. Por este motivo las pruebas de caja blanca se realizan a los principales algoritmos o procedimientos.(4)

Uno de los métodos o técnicas de prueba unitarias, es *la prueba del camino básico*. Esta técnica permite obtener una medida de la complejidad de un procedimiento o algoritmo y un conjunto básico de caminos de ejecución de este, los cuales luego se utilizan para obtener los casos de prueba. Esta técnica asegura que durante la prueba se ejecute al menos una vez cada sentencia del código que se está probado (4). Esta será la técnica utilizada para desarrollar los casos de pruebas unitarias a la herramienta desarrollada.

De igual manera existen varias métricas de software para realizar pruebas unitarias, entre estas se encuentra la *complejidad ciclomática*, la cual será utilizada junto a la técnica explicada anteriormente. Esta métrica proporciona una medición cuantitativa de la complejidad lógica de un procedimiento. La complejidad ciclomática cuando se utiliza en el contexto del método de prueba del camino básico, el valor que se calcula como complejidad ciclomática define el número de caminos independientes[13] del conjunto básico[14] de un programa y nos da un límite superior para el número de casos de prueba que se deben realizar para asegurar que cada sentencia de código se ejecuta al menos una vez.(4)

Para realizar las pruebas unitarias o de caja blanca al código de la herramienta se seleccionaron varios algoritmos que son los fundamentales y los que más comunes se encuentran. A continuación se explica todo el procedimiento de obtener los casos de prueba y poner a pruebas estos a través de un ejemplo, utilizando la técnica prueba del camino básico junto con la métrica complejidad ciclomática. Para esto se hará uso de una

[13] **camino independiente:** es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición.

[14] **conjunto básico**: Es el conjunto de caminos independientes.

notación para representación de flujo de control, denominada grafo de flujo, en la figura se muestra como se utiliza.

Figura 8. Construcciones estructurales en forma de grafo de flujo

El algoritmo seleccionado lleva como nombre *salvar_expertos_agregados* tiene como finalidad agregar expertos a un proceso de evaluación, para esto recibe como parámetro los identificadores (id) de los usuarios que seleccionan para agregar. En caso de que este usuario ya sea un usuario experto en la herramienta este solo se agrega el proceso de evaluación en caso contrario se agrega además como usuario experto a la herramienta. En caso de estar agregado como experto a este proceso de evaluación este no se volverá a adicionar.

```
def salvar_expertos_agregados(params)

  evaluacion = Evaluacion.find(params[:evaluacion]) #1

  if params[:marcados] #2
    agregar_ids = params[:marcados].collect {|id| id.to_i}#3
    agregar_ids.each do |id|#4

      if Experto.exists?(id)#5
        experto = Experto.find(id)#6
      else
        experto =  Experto.new#7

        user = User.find(id)#7

        experto.id = user.id#7
        experto.login = user.login#7
        experto.firstname = user.firstname#7
        experto.lastname = user.lastname#7
        experto.mail = user.mail#7

        experto.save#7
      end#8

      unless evaluacion.expertos.exists?(experto)#8
        evaluacion.expertos.push( experto )#9
      end#10
    end#11
  end#11
end#11
```

Figura 9. Prueba unitaria método salvar_expertos_agregados

Para obtener los casos de prueba a partir de este algoritmo se debe construir inicialmente el grafo de flujo correspondiente al código.

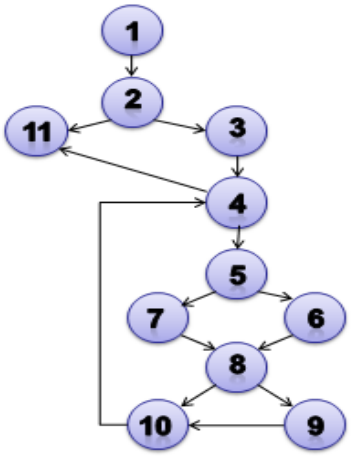


Figura 10. Prueba unitaria grafo de flujo

Luego se determina la complejidad ciclomática **V(G)** del grafo resultante **G**, para esto se utiliza las siguientes formulas:

El número de regiones del grafo de flujo coincide con la complejidad ciclomática.

V(G) = A – N + 2

Donde A es el número de aristas del grafo de flujo y N es el número de nodos del mismo.

V(G) = P + 1

Donde P es el número de nodos predicados contenidos en el grafo de flujo G.

Del grafo obtenido se tiene que:

Número de regiones del grafo de flujo es 5.	**A = 14**	**P = 4**
V(G) = 5	**N = 11**	**V(G) = P + 1**
	V(G) = A – N + 2	**V(G) = 4 + 1**
	V(G) = 14 – 11 + 2	**V(G) = 5**
	V(G) = 5	

El número de caminos independientes de la estructura del programa es igual a 5 por lo que se obtiene el conjunto básico.

Camino 1: 1-2-11
Camino 2: 1-2-3-4-11
Camino 3: 1-2-3-4-5-6-8-9-10
Camino 4: 1-2-3-4-5-6-8-10
Camino 5: 1-2-3-4-5-7-8-9-10

Luego se definen 5 casos de prueba para el código del algoritmo. Estos se realizaron en Ruby on Rails el cual posibilita realizar pruebas unitarias, mediante un completo marco de trabajo de prueba que viene pre instalado con Ruby. Este marco de trabajo de prueba, permite insertar datos de prueba mediante la configuración de los ficheros fixtures[15] correspondiente al modelo de los objetos que se desean insertar. Antes de comenzar las pruebas se insertan 3 evaluaciones, los usuarios de prueba de GESPRO y tres usuarios expertos.

```
class EvaluacionTest < ActiveSupport::TestCase
  fixtures :evaluacions
  fixtures :users
  fixtures :expertos
```

Figura 11. Prueba unitaria fixtures

Caso 1: Para la evaluación con id = 1, no se pasa en los parámetros los id de los expertos marcados es decir la variable params[:marcados] no existe.

[15] **fixtures:** Son elementos que utilizan las pruebas, fundamentalmente almacena datos de prueba en la base de datos.

```
test "camino_1" do

  evaluacion = evaluacions(:evaluacion_uno)

  params = {:evaluacion=>evaluacion}
  salvar_expertos_agregados(params)

  assert_equal(0,evaluacion.expertos.size)

end
```

Figura 12. Prueba unitaria código camino 1

Resultado: No se inserta ningún experto a la evaluación.

```
test$ ruby -I test unit/evaluacion_test.rb
RESQUE, loaded!
Loaded suite unit/evaluacion_test
Started
..
Finished in 0.39395 seconds.

2 tests, 1 assertions, 0 failures, 0 errors
```

Figura 13. Prueba unitaria resultado ejecutar código camino 1

Caso 2: Para la evaluación con id = 1, no se pasa en los parámetros los id de los expertos marcados es decir la variable params[:marcados] existe pero no contiene ningún elemento.

```
test "camino_2" do

  evaluacion = evaluacions(:evaluacion_uno)

  params = {:evaluacion=>evaluacion,:marcados=>[]}
  salvar_expertos_agregados(params)

  assert_equal(0,evaluacion.expertos.size)


end
```

Figura 14. Prueba unitaria código camino 2

Resultado: No se inserta ningún experto a la evaluación.

```
test$ ruby -I test unit/evaluacion_test.rb
RESQUE, loaded!
Loaded suite unit/evaluacion_test
Started
..
Finished in 0.404783 seconds.

2 tests, 1 assertions, 0 failures, 0 errors
```

Figura 15. Prueba unitaria resultado ejecutar código camino 2

Caso 3: Para la evaluación con id = 1 se pasa en los parámetros los id de los expertos marcados es decir la variable params[:marcados] existe con los id 1,2 y 3 pero estos son ya expertos del sistema, solo deben ser agregados como expertos a la evaluación y no al sistema.

```
test "camino_3" do

  evaluacion = evaluacions(:evaluacion_uno)

  #Existen 3 expertos en el sistema
  assert_equal(3,Experto.count)

  params = {:evaluacion=>evaluacion,:marcados=>[1,2,3]}
  salvar_expertos_agregados(params)

  #Ya existen esto 3 expertos en el sistema por lo que no se agrega ninguno
  assert_equal(3,Experto.count)

  assert_equal(3,evaluacion.expertos.size)

end
```

Figura 16. Prueba unitaria código camino 3

Resultado: Son adicionados los tres expertos a la evaluación pero no al sistema.

```
test$ ruby -I test unit/evaluacion_test.rb
RESQUE, loaded!
Loaded suite unit/evaluacion_test
Started
..
Finished in 0.486099 seconds.

2 tests, 3 assertions, 0 failures, 0 errors
```

Figura 17. Prueba unitaria resultado ejecutar código camino 3

Caso 4: Para la evaluación con id = 1 se pasa en los parámetros los id de los expertos marcados es decir la variable params[:marcados] existe con los id 1,2 y 3 pero estos son ya expertos del sistema y en el caso del experto 3 ya ha sido agregado con anterioridad a la evaluación, solo deben ser agregados a la evaluación los expertos 1 y 2 y ningún experto nuevo al sistema.

```
test "camino_4" do

  evaluacion = evaluacions(:evaluacion_uno)

  #Existen 3 expertos en el sistema
  assert_equal(3,Experto.count)

  #Se asegura que exista el experto con id 3 en la evaluacion
  params = {:evaluacion=>evaluacion,:marcados=>[3]}
  salvar_expertos_agregados(params)

  #Se trata de insertar nuevamente el experto con id 3
  params = {:evaluacion=>evaluacion,:marcados=>[1,2,3]}
  salvar_expertos_agregados(params)

  #Ya existen esto 3 expertos en el sistema por lo que no se agrega ninguno
  assert_equal(3,Experto.count)

  assert_equal(3,evaluacion.expertos.size)

end
```

Figura 18. Prueba unitaria código camino 4

Resultado: Solo se agregan los expertos 1 y 2 a la evaluación ya que el experto 3 estaba agregado con anterioridad a esta y no se agregan los expertos nuevamente al sistema.

```
test$ ruby -I test unit/evaluacion_test.rb
RESQUE, loaded!
Loaded suite unit/evaluacion_test
Started
..
Finished in 0.497934 seconds.

2 tests, 3 assertions, 0 failures, 0 errors
```

Figura 19. Prueba unitaria resultado ejecutar código camino 3

Caso 5: Para la evaluación con id = 1 se pasa en los parámetros los id de los expertos marcados es decir la variable params[:marcados] existe con los id 1,2, 3 y 4 pero de estos los expertos 1,2 y 3 son ya expertos del sistema. El usuario 4 debe ser adicionado como

experto al sistema ya que no existe, deben ser agregados a la evaluación todos los expertos.

```
test "camino_5" do

  evaluacion = evaluacions(:evaluacion_uno)
  # De los 4 a insertar 3 ya son expertos del sistema
  assert_equal(3,Experto.count)

  # El usuario con id 4 se agrega como experto del sistema
  params = {:evaluacion=>evaluacion,:marcados=>[1,2,3,4]}
  salvar_expertos_agregados(params)

  assert_equal(4,Experto.count)
  assert_equal(4,evaluacion.expertos.size)

end
```

Figura 20. Prueba unitaria código camino 5

Resultado: El usuario 4 es agregado como experto del sistema y se agregan los expertos 1, 2, 3 y 4 a la evaluación.

```
test$ ruby -I test unit/evaluacion_test.rb
RESQUE, loaded!
Loaded suite unit/evaluacion_test
Started
..
Finished in 0.575865 seconds.

2 tests, 3 assertions, 0 failures, 0 errors
```

Figura 21. Prueba unitaria resultado ejecutar código camino 3

Se ejecutaron los casos de pruebas descritos utilizando el marco de trabajo para pruebas unitarias de Ruby, obteniéndose para todas resultados satisfactorios. El algoritmo probado funciona correctamente para todos los escenarios para el cual fue implementado, lo cual quedo demostrado con las pruebas que se le realizaron al mismo.

3.2.2. Pruebas de aceptación

A continuación se muestran algunos Casos de Prueba de Aceptación (CPA en lo adelante) que se realizaron para validar el sistema a partir de las historias de usuarios y los requisitos del sistema, de un total de 37, el resto de los CPA se encuentran en los

documentos anexos a este trabajo. Estos CPA se aplican a la herramienta utilizando un caso de estudio de factibilidad, lo cual posibilita probar el funcionamiento completo de la aplicación por parte del usuario o cliente final ya que mediante este se puede ejecutar todo el proceso de análisis de factibilidad.

Caso de Prueba de Aceptación		
Código: 9	**Historia de Usuario (Nro. y nombre):** HU #3 Gestionar fichas de proyectos	
Funcionalidad que se prueba: Mostrar fichas de proyectos existentes.		
Condiciones de Ejecución: El usuario que está autenticado debe tener privilegios de especialista o administrador. Debe existir al menos una ficha de proyecto para que sea mostrado el listado.		
Acción	**Datos de entradas**	**Resultado esperado**
Se accede a la funcionalidad para inscribir las fichas de proyecto.		El sistema debe mostrar un listado con todas las fichas de proyecto existentes actualmente.
Resultado de la prueba: Satisfactoria		

Tabla#18 Descripción del Caso de Prueba de Aceptación HU#3

Caso de prueba de Aceptación		
Código: 10	**Historia de Usuario (Nro. y nombre):** HU #3 Gestionar fichas de proyectos	
Funcionalidad que se prueba: Crear una ficha de proyecto.		
Condiciones de Ejecución: El usuario que está autenticado debe tener privilegios de especialista o administrador.		
Acción	**Datos de entradas**	**Resultado esperado**
Se escoge la opción de crear una nueva ficha de proyecto.		El sistema debe mostrar un formulario para entrar los datos necesarios para crear una ficha de proyecto.

Se introducen los datos necesarios para crear una ficha de proyecto.	Datos obligatorios y datos opcionales.	
Se guardan los valores introducidos en cada uno de los campos.	Nueva ficha de proyecto.	El sistema debe mostrar un mensaje indicando que la acción se ha realizado de manera satisfactoria. Mostrar el listado con todas las fichas de proyecto y mostrando la última que se acaba de crear.
Resultado de la prueba: Satisfactoria		

Tabla#19. Descripción del Caso de Prueba de Aceptación HU#3

Caso de prueba de Aceptación		
Código: 11	**Historia de Usuario (Nro. y nombre):** HU #3 Gestionar fichas de proyectos	
Funcionalidad que se prueba: Modificar una ficha de proyecto existente.		
Condiciones de Ejecución: El usuario que está autenticado debe tener privilegios de especialista o administrador. Debe existir al menos una ficha de proyecto.		
Acción	**Datos de entradas**	**Resultado esperado**
Se selecciona la ficha de proyecto que se desea modificar, se escoge la opción modificar.	La ficha de proyecto a modificar.	El sistema debe mostrar un formulario con los valores actuales y permitir que estos sean editados.
Se introducen los nuevos valores de los campos que se desean modificar.	Los valores de los campos que se modifican.	
Se guardan los cambios	La ficha de proyecto modificada.	El sistema debe mostrar

		un mensaje indicando que la acción se ha realizado de manera satisfactoria. Mostrar el listado con todas las fichas de proyecto que existen y mostrando la última que se acaba de modificar con los cambios realizados.
Resultado de la prueba: Satisfactoria		

Tabla#20. Descripción del Caso de Prueba de Aceptación HU#3

Caso de prueba de Aceptación		
Código: 12	**Historia de Usuario (Nro. y nombre):** HU #3 Gestionar fichas de proyectos	
Funcionalidad que se prueba: Mostrar todos los detalles de una ficha de proyecto existente.		
Condiciones de Ejecución: El usuario que está autenticado debe tener privilegios de especialista o administrador. Debe existir al menos una ficha de proyecto.		
Acción	**Datos de entradas**	**Resultado esperado**
Se selecciona la ficha de proyecto que se desea visualizar y se escoge la opción mostrar.	La ficha de proyecto que se desea mostrar los detalles.	El sistema debe mostrar un formulario con los valores actuales.
Resultado de la prueba: Satisfactoria		

Tabla#21. Descripción del Caso de Prueba de Aceptación HU#3

Caso de prueba de Aceptación	
Código: 13	**Historia de Usuario (Nro. y nombre):** HU #3 Gestionar fichas de proyectos
Funcionalidad que se prueba: Eliminar una ficha de proyecto.	
Condiciones de Ejecución: El usuario que está autenticado debe tener privilegios de	

especialista o administrador. Debe existir al menos una ficha de proyecto.		
Acción	**Datos de entradas**	**Resultado esperado**
Se selecciona la ficha de proyecto que se desea eliminar, se escoge la opción eliminar.	La ficha de proyecto que se desea eliminar	El sistema debe mostrar un mensaje indicando que la acción se ha realizado de manera satisfactoria. Mostrar el listado con todas las fichas de proyecto restantes.
Resultado de la prueba: Satisfactoria		

Tabla#22. Descripción del Caso de Prueba de Aceptación HU#3

3.3. Conclusiones del capítulo

Del presente capítulo se concluye que:

- Se realizó la validación y prueba de la herramienta desarrollada asegurando la adecuada correspondencia entre las funcionalidades desarrolladas con los requisitos iniciales.
- Se ejecutaron las pruebas unitarias y de aceptación que son las que propone la metodología XP, para comprobar en cada una de las iteraciones del desarrollo, que se han alcanzado los objetivos propuestos al inicio de estas.
- Las pruebas unitarias realizadas a varios algoritmos del sistema posibilitaron la detección de errores, arrojando finalmente resultados satisfactorios luego de corregidos estos.
- Finalmente se realizaron pruebas de aceptación o funcionales a la herramienta, las cuales mediante un caso de estudio introducido en esta, posibilitaron la corrección de errores detectados durante este proceso obteniéndose finalmente luego de una nueva iteración, el correcto funcionamiento de la aplicación desarrollada para realizar los estudios de factibilidad en proyectos de software.

Conclusiones Generales

Mediante el desarrollo de este trabajo se llego a las siguientes conclusiones:

- Luego de realizar un análisis de las herramientas informáticas y modelos utilizados actualmente para realizar estudios de factibilidad de proyecto, se llegó a la conclusión de que no satisfacen completamente las necesidades existentes en la UCI.

- Se hace necesario, la elaboración de una herramienta para realizar análisis de factibilidad en proyectos de software que se integre a la herramienta GESPRO, seleccionando como modelo a implementar el propuesto por la Ing. Marieta Peña.

- Para el desarrollo de la herramienta se analizaron un conjunto de posibles herramientas y tecnologías a utilizar, seleccionando finalmente las que satisfacían las necesidades del sistema a desarrollar y todas son tecnologías libres.

- Durante el desarrollo de la herramienta se obtuvieron un conjunto de artefactos propuestos por la metodología de desarrollo seleccionada y finalmente el código ejecutable de la misma.

- Se validó la herramienta desarrollada utilizando pruebas unitarias y de aceptación con la introducción de un caso de estudio de factibilidad de proyecto en la herramienta, arrojando finalmente los resultados deseados.

- Con el desarrollo de este sistema se obtuvo una herramienta para evaluar factibilidad en proyectos de software la cual se puede integrar como un módulo de la plataforma GESPRO, acorde con las necesidades existente en la UCI, la cual facilita y contribuye a la toma de decisiones a la hora de seleccionar proyecto a desarrollar.

Recomendaciones

1. Realizar un estudio más profundo que permita incorporar nuevas funcionalidades a la herramienta.

2. Incorporar nuevos reportes del estudio de factibilidad realizado.

3. Luego de poner el sistema en explotación por parte de los especialistas que la utilicen, tomar experiencias y retroalimentarse en aras de mejorar el herramienta.

Referencias Bibliográficas

1. Yaima Puig Meneses. Asumir la planificación como herramienta de trabajo. Granma. 2011 Jul 29;:16.

2. Ing. Maylé Díaz Castro. Método de evaluación de proyectos para decidir su aceptación. [La Habana]: Universidad de las Ciencias Informáticas; 2010. 101 p.

3. Project Management Institute, Inc. PMI,Guía de los Fundamentos de la Dirección de Proyectos. 2004.

4. Presman R. Ingenieria de Software un enfoque práctico. Quinta Edición. 2002. 601 p.

5. Pérez Serrano G. Elaboración de proyectos sociales. Casos prácticos. Madrid, España: 1999.

6. Academia de Ciencias de Cuba. Requisitos para tener en cuenta para la elaboración de los estudios de prefactibilidad económica y científico-técnica de los resultados científicos. La Habana: 1991.

7. Karen Acevedo E, E. A. B. Estudio de Factibilidad de un proyecto. U. d. Atlántico, Slideshare: 2010.

8. Ramírez Almaguer, V. M. y. D. R. «ETAPAS DEL ANÁLISIS DE FACTIBILIDAD. COMPENDIO BIBLIOGRÁFICO » Contribuciones a la Economía. 2009.

9. Marieta Peña Abreu, Pedro Yobanis Piñero. Modelo para análisis de factibilidad en la evaluación de Proyectos de Software. 2012 Feb;

10. Alberto Cabrer Rodríguez. Propuesta de un procedimiento para evaluar la factibilidad de las posibles alternativas de desarrollo de software [Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas]. [Ciudad de La Habana,]: Universidad de las Ciencias Informáticas; 2010. 77 p.

11. Oficial Web Site Expert Choice. Collaboration Software to Meet the Needs of Your Organization - Expert Choice [Internet]. 2011 Dic 13 [citado 2011 Dic 13];Available from: http://www.expertchoice.com/products-services/

12. Ishizaka A, Labib A. Analytic hierarchy process and expert choice: Benefits and limitations. OR Insight. 2009;22(4):201–20.

13. Software EasyPlanEx para evaluar y optimizar proyectos de inversión [Internet]. 2012 Ene 9 [citado 2012 Ene 9];Available from: http://www.borasystems.com/homeespanol/softwareeasyplanex.html

14. DECIDE - Proyectos Eficientes [Internet]. 2012 Ene 9 [citado 2012 Ene 9];Available from: http://www.decide.com.mx/decideframeset.htm

15. Laboratorio de Soluciones de Gestión de Proyectos. Resumen_para_la_Toma_de_Decisiones_GESPRO_11.05. 2011 Nov 5;

16. Introducción — Inicio [Internet]. 2012 Ene 17 [citado 2012 Ene 17];Available from: http://comunidades.uci.cu/gespro-help/introduction/

17. Ivar J, Grady B, James R. El Proceso Unificado de Desarrollo e Software. Primera Edición en Español. Pearson Educación S.A; 2000.

18. Juan Palacio. Flexibilidad con SCRUM. 2007.

19. Kent Beck. Extreme Programming Explained. 1999.

20. Rational Rose Enterprise es el producto más completo de la familia Rational Rose, incluye soporte de Unified Modeling Language. [Internet]. [citado 2012 May 23];Available from: http://www.rational.com.ar/herramientas/roseenterprise.html

21. UML, BPMN and Database Tool for Software Development [Internet]. [citado 2012 May 11];Available from: http://www.visual-paradigm.com/

22. Sobre PostgreSQL | www.postgresql.org.es [Internet]. 2012 Ene 13 [citado 2012 Ene 13];Available from: http://www.postgresql.org.es/sobre_postgresql

23. pgAdmin: PostgreSQL administration and management tools [Internet]. 2012 Ene 13 [citado 2012 Ene 13];Available from: http://www.pgadmin.org/

24. Kevin C . Baird. Ruby by example concepts and code. San Francisco, USA: 2007.

25. Peter Cooper. Beginning Ruby From Novice to Professional. USA: 2007.

26. Ruby on Rails [Internet]. 2012 Ene 17 [citado 2012 Ene 17];Available from: http://www.rubyonrails.org.es/

27. Kerrigan M, Mocan A, Tanler M, Fensel D. The web service modeling toolkit-an integrated development environment for semantic web services. The Semantic Web: Research and Applications. 2007;:789–98.

28. NetBeans IDE - Features [Internet]. 2012 Ene 16 [citado 2012 Ene 16];Available from: http://netbeans.org/features/index.html

29. IBM Corporation and others. Welcome to Eclipse [Internet]. 2005 Feb 24 [citado 2012 Ene 20];Available from: http://archive.eclipse.org/eclipse/downloads/drops/R-3.1-200506271435/org.eclipse.platform.doc.isv.3.1.pdf.zip

30. Sommerville I. Software Engineering. Octava Edición. Ed. Addison-Wesley. Harlow: Pearson Education. SA; 2007.

31. Craig Larman. UML y Patrones Introducción al análisis y diseño orientado a objetos. México: Prentice Hall; 1999.

32. Sam Ruby, Dave thomas, David Heinemeier Hansson. Agile Web Development with Rails. 3o ed. United States of American: 2009.

33. Gutiérrez JJ, Escalona MJ, Mejías M, Torres J. Pruebas del Sistema en Programación Extrema. Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla. 2006;