Universidad de las Ciencias Informáticas

Facultad 3

“Herramienta para la definición de criterios de factibilidad para evaluar proyectos informáticos.”

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

Autor:

José Luis Guisao Jorge

Tutor(es):

DrC. Marieta Peña Abreu

Ing. Liannet Baez Fernández

Co-tutor:

Ing. Yoslenys Roque Hernández

La Habana, julio de 2018.

“Año 60 de la Revolución”.

Sabemos quiénes somos, cuánto nos costó llegar hasta aquí y todo lo que seguiremos haciendo a nuestro modo, como el mar…

Rosalina Ibarra González

DECLARACIÓN DE AUTORÍA

Declaro ser autor del presente trabajo y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____	_____
José Luis Guisao Jorge	DrC. Marieta Peña Abreu
Firma del autor	Firma del tutor
_____	_____
Ing. Liannet Baez Fernández	Ing. Yoslenys Roque Hernández
Firma del tutor	Firma del tutor

DATOS DE CONTACTO

Autor: José Luis Guisao Jorge

Correo electrónico: jlguisao@estudiantes.uci.cu

Tutor: DrC. Marieta Peña Abreu

Correo electrónico: mpabreu@uci.cu

Tutor: Ing. Liannet Baez Fernández

Correo electrónico: lbaez@uci.cu

Co-tutor: Ing. Yoslenys Roque Hernández

Correo electrónico: yrhdez@uci.cu

AGRADECIMIENTOS

A mis padres por todo el amor y cariño que me han dado, por respetar mis decisiones y opiniones y por siempre apoyarme en cada paso que doy, por haberme educado y enseñado los valores que tengo, por ser los mejores del mundo. A mi mamá por siempre pensar en mí primero y por darme tanto amor y cariño, te quiero inmensamente. A mi padre por ser mi ejemplo de superación y por todo su cariño, te quiero mucho.

A mis hermanas por ser las niñas de mis ojos, las luces de mi vida.

A mi familia por el apoyo, amor y cariño que me han dado toda mi vida. A mis abuelos Francisca, Noris y Crescencio por cuidarme como si fuera su propio hijo. A mi abuelo Onelio que nunca conocí pero que desde el cielo siempre cuida de mí. A mis tías Odalis, Oilda, Onelda, Osaida, Laura y Niurka, las quiero. A mis primos Yordania, Kenia, Carlos, Kirenia, Yordan, Yanisleidis, Karenia, Javier, Leonardo, Juan Pablo, Pablo Enrique y Lennis por haberme hecho vivir la mejor etapa de mi vida.

A mi gran familia de amigos, pues a lo largo de este tiempo se han convertido en parte de mi familia, más que amigos, hermanos. Gracias por sacarme de los vacíos emocionales. A Dayan, Luis Miguel, Daylin, Yoslenys, Rosalina, Addiel, Nailee y Claudia por los buenos y malos momentos vividos y por su amistad. A Julio, Ively, Yeisel, Jeiser, Evelyn y todo Espacio Abierto por haberme hecho un mejor artista y por los momentos inolvidables que vivimos juntos.

A mis tutores por ser los faros que alumbraron mi sendero. A Marieta y Liannet por toda la ayuda y el apoyo brindado. A Yoslenys por la inmensa ayuda y por la amistad, estaré siempre en deuda contigo.

A mi gente del secretariado de la FEU de la Facultad, por aguantar mis locuras y malhumor por tal de que las cosas salieran bien. A Julio, Rosalia, Juan Gabriel, Eddy, Hean Carlos, Doris, Claudia, gracias por caminar por ese camino juntos.

A mis compañeros de apartamento que mejores no pude tener. A Osniel, Oslen, Omar, Luis Ángel y Alejandro gracias por respetarme.

A mis compañeros de año por haberme acogido como uno más de ustedes.

A mis profesores por haberme formado para ser el profesional en quien hoy me convierto. En especial a la profe Reina por haber confiado en mí y haberme dado la oportunidad de enderezar el camino a tiempo, por mostrarme que podía llegar al final y que podía hacer mucho más antes de alcanzar la meta.

A todos los que de una manera u otra compartieron conmigo o tan siquiera preguntaron ¿y la tesis?

A la Revolución Cubana por haberme dado la oportunidad de estudiar en esta maravillosa universidad que me formó como un profesional integral y por convertirme en heredero del sueño de quien la fundara: Fidel Castro Ruz.

A mi Dios por haberme dado la fuerza y la fe en los momentos más difíciles.

DEDICATORIA

A las personas que hicieron posible que este sueño se hiciera realidad.

A mis padres, mi familia y mis amigos por ser los pilares que sostienen mi vida.

RESUMEN

La exigencia de desarrollar un producto con eficiencia y calidad conlleva a la necesidad de realizar estudios de factibilidad que garanticen una correcta selección de proyectos. Tradicionalmente, estos estudios, han sido realizados por expertos en el tema, mediante métodos tradicionales de evaluación que por lo general solo se evalúan criterios económicos deterministas. La mayoría son realizados en entornos de incertidumbre, donde la información disponible de las diferentes alternativas a evaluar puede ser incompleta, vaga o imprecisa, por lo que debe ser valorada cualitativamente. El análisis de factibilidad es una tarea incluida en la ruta crítica de cualquier proyecto. Diversos autores han propuesto criterios económicos, técnicos, comerciales, sociales y medioambientales para diferentes tipos de proyectos. Sin embargo, para los proyectos de software, se observa una vaga integración entre los criterios y poca formalidad en su definición; lo que limita la utilidad de los resultados. El presente trabajo tiene como objetivo desarrollar una herramienta informática que emplee el tratamiento de la incertidumbre presente en las valoraciones de los expertos durante el análisis de la factibilidad de proyectos de software usando técnicas de soft computing. El sistema fue desarrollado utilizando herramientas y tecnología libres, como metodología de desarrollo se empleó AUP-UCI y el marco de trabajo PHP Symfony. El alcance de este trabajo está acotado a la implementación del Componente 1 del Modelo para el análisis de factibilidad de proyectos de software en entornos de incertidumbre. La solución desarrollada contribuye al análisis de factibilidad de proyectos de software con tratamiento de la incertidumbre.

Palabras clave: estudios de factibilidad, factibilidad, incertidumbre.

ABSTRACT

The requirement to develop a product with efficiency and quality leads to the need to carry out feasibility studies that guarantee a correct selection of projects. Traditionally, these studies have been carried out by experts on the subject, using traditional methods of evaluation that generally only deterministic economic criteria are evaluated. Most of them are carried out in uncertainty environments, where the available information of the different alternatives to be evaluated may be incomplete, vague or imprecise, so it must be assessed qualitatively. Feasibility analysis is a task included in the critical path of any project. Several authors have proposed economic, technical, commercial, social and environmental criteria for different types of projects. However, for software projects, there is a vague integration between the criteria and little formality in their definition; which limits the usefulness of the results. The objective of this work is to develop a computer tool that uses the treatment of the uncertainty present in the evaluations of the experts during the feasibility analysis of software projects using soft computing techniques. The system was developed using free tools and technology, as a development methodology was used AUP-UCI and the PHP Symfony framework. The scope of this work is limited to the implementation of Component 1 of the Model for the feasibility analysis of software projects in uncertainty environments. The solution developed contributes to the feasibility analysis of software projects with uncertainty treatment.

Key words: *feasibility studies, feasibility, uncertainty.*

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La creciente utilización de las Tecnologías de la Información y las Comunicaciones y su impacto en la sociedad conllevan a que las organizaciones se enfrenten a mayores retos en la obtención de proyectos exitosos. La selección adecuada de un proyecto a desarrollar puede impulsar la economía y la sociedad (Serra, y otros, 2015), pero es una tarea compleja, ya que está dada por múltiples factores que se deben considerar para tomar tal decisión. Los análisis de factibilidad contribuyen en la toma decisiones en las organizaciones al momento de decidir la ejecución o no del proyecto, ya que evalúan la disponibilidad de recursos para enfrentar el desarrollo, así como los beneficios que puede retornar (Villagrán Andrade, 2015).

Un componente que ha impulsado el desarrollo social, es el despliegue de productos de software, que inciden en la productividad, la planificación, el control, la calidad de los procesos entre otros disímiles aspectos. Los proyectos informáticos imponen retos adicionales en el proceso de análisis de factibilidad dado por el carácter intangible del software y su asociación al desarrollo de conocimientos, siendo engorrosa la toma de decisiones sobre el desarrollo o no de dichos proyectos. Estudios publicados por *Standish Group*, en el Chaos Manifiesto 2015 a alrededor de 50 000 proyectos, arrojaron que el 52% de estos fue renegociado y el 19% fallido, siendo una de las principales causas el aumento de los costos durante su ejecución, a pesar de los esfuerzos realizados por las organizaciones. La evaluación de los criterios correctos ha sido unos de los elementos más importantes desde su surgimiento, el cual también ha repercutido en el desarrollo computacional de los proyectos de software. Las principales escuelas de Gestión de Proyectos (ISO 21500, PMBOK, CMMI, IPMA, PRINCE2) suponen que se haya realizado con anterioridad el análisis de factibilidad, pero no proponen criterios para su realización.

Para resolver estas limitaciones los análisis de la factibilidad pudieran tratarse como un problema de toma de decisiones. A partir de la analogía con este tipo de problemas, se deben considerar elementos como los criterios a evaluar y sus métodos de cálculo, entre otros. Dentro de los criterios a evaluar, desde sus inicios los análisis de factibilidad estuvieron muy relacionados con el cálculo de criterios económicos completamente deterministas, sin tener en cuenta la incertidumbre inherente en el ambiente de desarrollo. La incertidumbre en el proceso de solución de problemas puede estar presente tanto en los datos de la instancia del problema que se resuelve, como en el conocimiento que se utiliza para resolverlo, en este caso está presente en ambos. Esto implica que en este contexto sea necesario modificar la manera de concebir los criterios tradicionales, lo cual se viene infiriendo desde hace años. Diversos autores proponen evaluar criterios tales como: económicos, técnicos, comerciales, sociales y medioambientales; pero existe vaga integración entre los criterios, provocando baja interpretabilidad[1] de los resultados y poca formalidad en la definición para proyectos informáticos.

[1] Capacidad de interpretación de datos o información.

La industria de desarrollo de software en Cuba no se encuentra ajena a la necesidad de realizar análisis de factibilidad al inicio de sus proyectos, razones enunciadas por la dirección del país: evitar futuros riesgos de ejecución y garantizar sostenibilidad en los proyectos. A raíz de la actualización del modelo económico cubano se emite la Resolución 327/2015 (MINJUS, 2015), que sienta bases regulatorias, pero no establece criterios para el análisis de proyectos de software ni cómo poder integrar el cálculo de éstos, para brindar al decisor información integral para la toma de decisiones.
En la Universidad de las Ciencias Informáticas (UCI) cada año se ejecutan un número considerable de proyectos nacionales y de exportación, los cuales necesariamente deben ser evaluados antes de iniciarse. La UCI se encuentra aún como una institución en proceso de madurez en algunos temas, lo cual trae consigo la ejecución de algunos procesos de manera ineficiente, existiendo en ocasiones déficit de personal calificado y de recursos materiales ante el crecimiento del número de proyectos a realizar. Por otra parte, los estándares internacionales para los análisis de factibilidad no se encuentran completamente implantados ni generalizados y dentro de la universidad se carece de un área especializada en el tema. Los modelos de evaluación existentes no se ajustan al modelo de producción de la UCI convergiendo la mayoría a temas de factibilidad económica engorrosos (Peña Abreu, 2017).

A partir de la situación descrita se plantea el siguiente **problema a resolver:** Las insuficiencias de herramientas informáticas que definan criterios para realizar análisis de factibilidad de proyectos informáticos en entornos de incertidumbre, está afectando la predicción de la factibilidad en el desarrollo de software.

Objeto estudio: El proceso de desarrollo de software en la definición de criterios de factibilidad de proyectos informáticos.

Campo de acción: Herramienta informática para la definición de criterios de factibilidad de proyectos informáticos en entornos de incertidumbre.

Objetivo general: Desarrollar una herramienta informática para la definición de criterios de factibilidad basado en técnicas de soft computing para el tratamiento de la incertidumbre que contribuya en la predicción de la factibilidad de proyectos informáticos.

Objetivos específicos:

- Construir el marco teórico de la investigación para apoyar el proceso de desarrollo de software en los estudios de factibilidad de proyectos informáticos.
- Desarrollar los artefactos ingenieriles para la obtención de la herramienta informática.
- Implementar las funcionalidades para obtener la herramienta informática, mediante las tecnologías, herramientas y lenguajes seleccionados.

- Evaluar la herramienta a partir de la validación a través de las técnicas seleccionadas y el correcto funcionamiento del software para obtener un producto con calidad.

Idea a defender: Si se desarrolla una herramienta informática para la definición de criterios de factibilidad basado en técnicas de soft computing para el tratamiento de la incertidumbre, se contribuye positivamente en la predicción de la factibilidad de proyectos informáticos.

Con el objetivo de resolver el problema y lograr el objetivo planteado, se hizo necesaria la utilización de los siguientes **métodos de investigación**:

Métodos teóricos:

- ✓ Histórico-lógico: se empleó para analizar la trayectoria y evolución de los modelos y herramientas existente para la evaluación de proyectos.
- ✓ Analítico-sintético: permitió, fundamentalmente realizar el estudio teórico de la investigación, haciendo posible la selección de los elementos más importantes para el proceso de desarrollo de la herramienta.
- ✓ Inductivo-deductivo: se utilizó para el razonamiento de la información consultada y llegar así a la obtención de un grupo de conocimientos particulares y generales.

Métodos empíricos:

- ✓ Entrevista: se utilizó en el intercambio con el cliente para adquirir información sobre el negocio y los requisitos que se deben cumplir en el desarrollo del software, siendo estas de tipo abierta.
- ✓ Tormenta de ideas: fue utilizado en varias reuniones con la participación del equipo de desarrollo y el cliente. Como resultado se logró generar opiniones sobre los requisitos a satisfacer por el software.

El presente trabajo de diploma está compuesto por tres capítulos, estructurados de la siguiente manera:

Capítulo I: Fundamentación teórica.

En este capítulo se realiza una revisión y análisis de los modelos, métodos, metodologías y herramientas existentes para evaluar proyectos de software. Se plantea el método para establecer criterios de evaluación para el análisis de factibilidad de proyectos de software a utilizar y se describen los pasos para la instrumentación del mismo. Se enuncian los principales conceptos

relacionados con el tema, así como un estudio de las metodologías, lenguajes, herramientas y patrones a usar como apoyo para dar solución al problema planteado. Por último, se describen las pruebas a realizar para la validación y verificación del sistema.

Capítulo II: Propuesta de solución.

En este capítulo se propone la solución de la investigación. Se exponen los requisitos funcionales y no funcionales. Se generan los artefactos del análisis y el diseño. Se describen los patrones de diseño y arquitectónicos empleados y se definen los estándares de codificación para la implementación de la herramienta.

Capítulo III: Evaluación de la propuesta de solución.

En este capítulo se evalúa el cumplimiento de los objetivos planteados, se valida el diseño aplicando las métricas Tamaño operacional de clases y Relaciones entre clases, se realizan pruebas a través de la aplicación del método de caja negra al sistema, pruebas de aceptación por parte del cliente y la validación de la solución propuesta a través de la aplicación de estas pruebas.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo son abordados los principales conceptos utilizados en la investigación. Se hace un estudio de diferentes modelos, métodos, metodologías y herramientas existentes, para la evaluación de proyectos de software con tratamiento de la incertidumbre en las valoraciones de los expertos durante el análisis de la factibilidad. Se describen las principales tecnologías, lenguajes de programación y herramientas definidas para el desarrollo de la solución, así como la metodología de desarrollo.

1.1 Principales conceptos

Para comprender los conceptos de factibilidad y criterio de factibilidad se presentan las siguientes definiciones:

Factibilidad: se realiza al mismo tiempo que se ejecuta el proyecto de acuerdo con los resultados que se obtienen y ayuda a decidir si se realiza la inversión para introducir los resultados (Hernández, 2002).

Criterio de factibilidad: Indicador medible a tener en cuenta para el estudio de factibilidad. Puede tener diversas clasificaciones (económica, técnica, comercial, medioambiental, social, entre otras) (Hernández, 2002).

Modelo: representación simplificada de la realidad que permite un entendimiento del objeto de estudio para resolver un problema determinado y representarlo como un enfoque sistémico (Denning, 2012).

Incertidumbre: Falta de certidumbre (RAE, 2018), de confianza o de certeza sobre algo.

1.2 Herramientas informáticas para estudio de factibilidad

Algunas organizaciones han optados por la utilización de herramientas informáticas para el estudio de la factibilidad de proyectos de software. En este epígrafe se relacionan y comparan las más utilizadas.

Herramienta computacional para la gestión y evaluación de proyectos software enmarcados en actividades de investigación (GEPSw):

El colectivo de autores (Martínez León, y otros, 2011) de la Universidad de Pereira propone una herramienta computacional para la evaluación de proyectos de software sobre las bases teóricas del PMBOK. Esta abarca todas las áreas del conocimiento de la gestión de proyectos, pero no tiene en cuenta los análisis de factibilidad, solo evalúa criterios durante el ciclo de vida del proyecto.

DECIDE:

Software especializado que facilita la elaboración de planes de negocios, formulación y evaluación de proyectos de inversión, proporciona metodologías y herramientas para una toma de decisiones sustentada. DECIDE ha sido desarrollado por la empresa mexicana Soluciones Informáticas y

Aplicaciones Crediticias S. A. de C.V. Permite realizar evaluación de un proyecto de inversión, elaborar estudios de mercado, análisis técnicos, económicos, financieros y de riesgos. Se basa en la evaluación de criterios económicos, de mercado y técnicos de proyectos con variación en el alcance y está dirigido fundamentalmente a consultoras y bancos, no define criterios, sino que evalúa los introducidos por la organización que la utiliza. Es una herramienta privativa, por lo que tiene costo de licencia para su uso (DECIDE, 2013).

Expertchoise:

Software para la toma de decisiones, basado en el Proceso Jerárquico Analítico (AHP, *Analytic Hierarchy Process*). Asiste a los decisores organizando la información relacionada con la complejidad del problema en un modelo jerárquico que consta de un objetivo, escenarios posibles, criterios y alternativas. Ha sido usado exitosamente en una variedad de aplicaciones incluyendo, priorización y evaluación de proyectos, planeamiento estratégico y análisis de costo/beneficio. *ExpertChoise* es una herramienta privativa, por lo que tiene costo de licencia para su uso (Choise, 2015).

Easyplanex:

Software que provee una solución integral para evaluar y optimizar la formulación de proyectos de inversión desarrollado por BoraSystems. Realiza análisis de riesgo, utiliza la técnica de Montecarlo[2], emplea diferentes distribuciones de probabilidades. Permite calcular la probabilidad de ocurrencia de un resultado. *EasyPlanex* realiza análisis de sensibilidad en forma automática, genera los resultados para todos los escenarios posibles. Además, permite optimizar un proyecto, encontrando la mejor alternativa. Es una herramienta privativa por lo que tiene costo de licencia para su uso (EasyPlanex, 2014).

Gespro:

Paquete de Gestión de Proyectos desarrollado por la UCI, registrado en el Centro Nacional de Derecho de Autor (CENDA) con el No. de Registro 1540-2010. Gespro abarca varias áreas de la Gestión de Proyectos, las cuales están presentes en el sistema mediante funcionalidades y módulos. Entre estas se encuentran: 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. Permite realizar estudios de factibilidad económica, a una cartera de proyectos, y obtener finalmente un listado de proyectos ordenados según su factibilidad (Piñeiro Pérez, y otros, 2013).

[2] El método de MonteCarlo es una técnica numérica para calcular probabilidades y otras cantidades relacionadas, utilizando secuencias de números aleatorios.

1.2.1 Valoración de las herramientas

Luego de realizado el estudio de las herramientas de evaluación de proyectos mencionadas, se muestra una tabla comparativa entre estas herramientas y los indicadores definidos.

Tabla 1. Comparación de las herramientas de evaluación de proyectos estudiadas.

Herramientas	Definen criterios de factibilidad	Trabajan la incertidumbre	Licencia
GEPSw	No	No	Privativa
DECIDE	No	No	Privativa
ExpertChoise	No	No	Privativa
EasyPlanex	No	Sí	Privativa
Gespro	No	No	Libre

Las herramientas analizadas facilitan la toma de decisiones en el momento de evaluar un proyecto, lo cual constituye una de sus ventajas. Como se puede evidenciar en la tabla comparativa, en su mayoría son propietarias por lo que es necesario pagar una licencia para su uso, ninguna de estas propone criterios de evaluación de factibilidad, elementos de gran importancia cuando se evalúa la factibilidad de un proyecto como tampoco tratan la incertidumbre de la información, siendo estas sus principales desventajas. Estas herramientas no satisfacen completamente las necesidades actuales para la evaluación de proyectos de software en entornos de incertidumbre. Por lo anteriormente expuesto, se propone el desarrollo de una herramienta informática para seleccionar criterios de factibilidad para evaluar proyectos de software bajo condiciones de incertidumbre, basándose en el Modelo para el análisis de factibilidad de proyectos de software en entornos de incertidumbre.

1.3 Modelos y métodos

A continuación, se expone el estudio realizado al modelo escogido y los métodos utilizados para la instrumentación de este:

1.3.1 Modelo para el análisis de factibilidad de proyectos de software en entornos de incertidumbre (MFac-PS):

Este modelo propone el análisis de factibilidad de proyectos de software en entornos de incertidumbre, a partir de la evaluación de criterios utilizando una combinación de métodos tradicionales (método DELPHI, método de análisis de síntesis curricular, método de evaluación de competencias) y técnicas de *soft computing* (modelo lingüístico 2-tuplas), que son expuestos posteriormente en este trabajo para una mejor comprensión.

Características:

- Define criterios de evaluación para realizar el análisis de factibilidad en proyectos de software en las áreas: económica, técnica, comercial y social.

- Propone el análisis de criterios con información heterogénea posibilitando la evaluación en diferentes dominios de expresión.
- Realiza tratamiento de la incertidumbre del entorno, utilizando técnicas de *soft computing* para el cálculo de los criterios de análisis de factibilidad (Peña Abreu, 2017).

El modelo propone cuatro componentes:

- Componente 1 “Definición de criterios de evaluación”: define los criterios de evaluación a considerar en los análisis de factibilidad de proyectos de software. En el componente se definen los criterios de factibilidad para evaluar proyectos de software, éstos constituyen la cantera para la selección de criterios de evaluación cada vez que se realice un análisis de factibilidad de proyectos de software. Para realizar la propuesta se aplican combinaciones de técnicas tradicionales y el método de modelado lingüístico 2-tuplas con el propósito de disminuir la incertidumbre. Es importante la descripción del contexto objeto de análisis por su influencia en la evaluación de los criterios.
- Componente 2 “Recopilación de datos para la evaluación”: recopila los datos para realizar el análisis de factibilidad.
- Componente 3 “Evaluación de criterios”: establece la evaluación de los criterios a partir de técnicas de *soft computing,* brindando una salida parcial al decisor con los resultados de cada criterio de manera independiente.
- Componente 4 “Respuesta integral”: integra el resultado de la evaluación de cada criterio a partir de técnicas de *soft computing,* brindando una mejor capacidad de predicción de la factibilidad al decisor, así como un análisis más integral para la toma de decisiones (Peña Abreu, 2017).

Se escoge este modelo por las características antes mencionadas que propician la definición de criterios de evaluación de factibilidad para proyectos de software en entornos de incertidumbre. El alcance de la presente investigación se centrará en la implementación del Componente 1 “Definición de criterios de evaluación” que para su instrumentación, (Peña Abreu, 2017), propone los siguientes pasos:

Paso 1. Definición de requisitos para la selección de expertos.
Paso 2. Selección de expertos participantes.
Paso 3. Definición de criterios de factibilidad para evaluar proyectos de software.
Paso 4. Conceptualización de criterios.

Descripción detallada del Paso 1, definición de requisitos de expertos.

En este paso se obtienen los requisitos que debe tener un trabajador de la industria del software para considerarse experto en análisis de factibilidad en proyectos de software. Se aplica la técnica Delphi. Se sugieren las siguientes actividades:

Actividad 1: Consultar a los participantes para que enuncien los requisitos que en su opinión deben evaluarse para catalogar un experto en el área.
Actividad 2: Consultar a los participantes para que evalúen los requisitos obtenidos en la actividad anterior. Seguir Algoritmo 1.
Actividad 3: Obtener el listado de requisitos para que un trabajador vinculado al desarrollo de software pueda ser seleccionado como experto.

Algoritmo 1 Evaluar requisitos de trabajador de software usando el modelo 2-tuplas:

Entradas:

R - Requisitos candidatos para seleccionar a los expertos $\mathrm{r_q} \in R : q \in [1..z]$

V - Conjunto de participantes evaluadores. $\mathrm{v_l} \in \mathrm{V} : t \in [1..L]$

Salidas: Listado de requisitos Re seleccionados para la evaluación de los expertos.

1. Inicio
2. Definir como conjunto básico de términos lingüísticos para la evaluación de los expertos (CBTL) $S_T = \{MPC, PC, MC, C, MC\}$[3]. (Figura 1).
3. Para cada requisito $r_q \in R$ hacer
4. Para cada participante evaluador $v_l \in V$ hacer
5. Emitir preferencias de participante v_l sobre el requisito r_q usando una escala lingüística de cinco etiquetas (Figura 1). Se construye la matriz de evaluación, donde cada elemento M_{ql} representa la evaluación del riesgo q por parte del participante l.
6. Fin del ciclo paso 4.
7. Calcular el q-ésimo elemento R_q de evaluación de requisitos tal que R_q representa la evaluación agregada de los participantes del requisito r_q, una vez aplicado el método 2-tuplas.
8. Fin del ciclo paso 3.
9. A partir del análisis del vector V_R, ordenar los requisitos según su evaluación y seleccionar los requisitos Re que finalmente se considerarán para evaluar a los expertos.
10. Devolver el listado de requisitos Re seleccionados.
11. Fin del algoritmo.

[3] Etiquetas lingüísticas:
MPC: muy poco considerable, PC: poco considerable, MC: medianamente considerable, C: considerable, MC: muy considerable

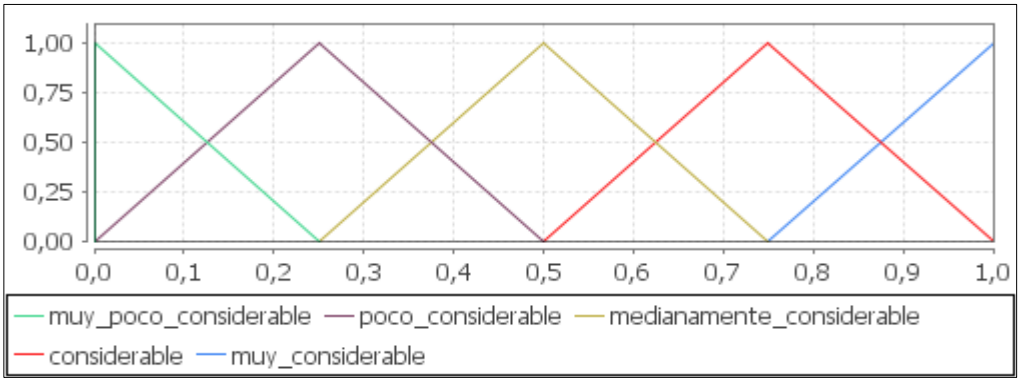

Figura 1. Etiquetas lingüísticas utilizadas para evaluar criterios para seleccionar expertos.

Descripción detallada del Paso 2, selección de expertos.

El objetivo de este paso es seleccionar a los expertos que se emplearán en el análisis de factibilidad considerando los requisitos R_e identificados en el paso anterior. Para ello se pueden aplicar métodos como “Análisis de síntesis curricular” o el “Método de evaluación de competencias”. En este paso se sugieren las siguientes actividades:

Actividad 1: Revisar síntesis curricular de candidatos y las competencias
Actividad 2: Validar evidencias de competencias y síntesis curricular de candidatos.
Actividad 3: Seleccionar a los expertos evaluarán la factibilidad.

Descripción detallada del Paso 3, definición de criterios de factibilidad.

Para instrumentar el paso se combina el uso de los métodos: análisis bibliográfico, el método Delphi y el método 2-tuplas. Las actividades propuestas se describen a continuación:

Actividad 1: Identificar un conjunto de criterios candidatos para el análisis.
Actividad 2: Aplicar ronda del método Delphi, una encuesta abierta donde los expertos pueden proponer nuevos criterios.
Actividad 3: Evaluar los criterios candidatos por parte de los expertos empleando el Algoritmo 1 con un CTL de cinco etiquetas lingüísticas definidas previamente (Figura 1).
Actividad 4: Eliminar criterios que hayan sido evaluados con la etiqueta no_considerar.
Actividad 5: Obtener listado criterios base, a aplicar en el contexto en cuestión.

Descripción detallada del Paso 4, conceptualización de criterios

El objetivo de este paso es lograr una definición clara de cada criterio: su descripción, dominio. Para ello se aplica el método del Grupo Focal en un solo equipo. La secuencia de actividades para el trabajo en equipo son las siguientes:

Actividad 1: Realizar primera sesión para explicar objetivo de trabajo y presentar los criterios base obtenidos en el Paso 3.

Actividad 2: Presentar propuesta de dominio para cada criterio.
Actividad 3: Realizar sesión final donde el grupo focal perfecciona lo presentado en la Actividad 2 y realiza recomendaciones para el trabajo con los criterios.
Actividad 4: Realizar ronda de valoraciones.
Actividad 5: Obtener listado final de criterios base para realizar análisis de factibilidad de proyectos de software en el contexto en análisis.

Para una mejor comprensión se describen seguidamente los métodos DELPHI y 2-tuplas.

1.3.2 Método DELPHI

Creado en 1948 por la *Research and Development Corporation* (*RAND Corporation,* en inglés) en Santa Mónica, Estados Unidos, para investigar el impacto de la tecnología en la guerra. En esta primera aplicación realizada en 1951 y desclasificada 10 años después se preguntó a 7 expertos sobre el futuro del arsenal norteamericano (Jones, y otros, 1995) (Cruz, y otros, 2008).
En el año 1958 sale la publicación de un artículo que expone su fundamentación científica, (Helmer, y otros, 1959) y en 1975 *Linstone* y *Turoff* publican el compendio *The Delphi Method. Techniques and Application,* donde aparecen los resultados de 489 estudios que utilizaron el Delphi en 20 años (García Valdés, y otros, 2013).
El método DELPHI es una técnica prospectiva para obtener información esencialmente cualitativa, pero relativamente precisa, acerca del futuro (Cáceres Navarro, 2012) y tiene como objetivo alcanzar un consenso sobre una estimación, a través de reuniones, cuestionarios y encuestas. Muchos proyectos poseen un considerable elemento de especialización y no se puede esperar que sus gestores sean expertos en cada uno de los aspectos. Permite mediante la utilización de herramientas estadísticas lograr consensos entre especialistas sin la necesidad de que estos se encuentren en un mismo sitio simultáneamente (Córdoba Padilla, 2011).
Una de las características principales de este método son los principios básicos que lo rigen:

- Es un proceso iterativo: consistente en la realización de rondas sucesivas de consultas para que los participantes revisen sus opiniones. Después de cada una se presentan los resultados, de tal manera que para la siguiente repetición los expertos conozcan los distintos puntos de vista y puedan ir modificando su opinión, si los argumentos presentados les parecen más apropiados que los suyos.
- Requiere retroalimentación: los expertos reciben las valoraciones de todos los participantes antes de cada ronda, para contrastar sus criterios con los del resto del grupo y ofrecer nuevamente su juicio.
- Requiere del anonimato para las respuestas individuales. Esto permite que un miembro del grupo no sea influenciado por la reputación de otro o por el peso que supone oponerse a la mayoría.

- Tiene como propósito la construcción de un consenso: este es un acuerdo general de grupo a partir del procesamiento estadístico de las diferencias y coincidencias entre las apreciaciones individuales y sus modificaciones a través de las rondas (Llorens Fábregas , 2005).

Ventajas:

- El principio de anonimato elimina el efecto del líder sobre los encuestados y garantiza la libertad de opiniones.
- Flexible ante el cambio de las opiniones en cada iteración.
- Para su realización no es necesario que todos los expertos estén simultáneamente en el mismo lugar.

Desventajas:

- Implementación es muy laboriosa y costosa.
- Requiere de una buena comunicación para la búsqueda y recepción de las respuestas.

1.3.3 Método lingüístico 2-tuplas:

Las 2-tuplas lingüísticas surgen de la necesidad de modelar información lingüística y que no haya, o al menos, en el menor grado posible, una pérdida de información en las operaciones y una falta de exactitud. El modelado lingüístico que representa la información basándose en el Enfoque Lingüístico Difuso pierde información porque parte de valores discretos en un universo continuo. Las 2-tuplas, se construyen a raíz de las variables lingüísticas que vienen a su vez de la teoría de números difusos, heredando sus operaciones. Esta teoría está claramente formalizada y validada. Además, las 2-tuplas son una forma matemática de representar la información y que se basa en la existencia de los números reales, partiendo además de que buscar modelar conocimiento lingüístico de un Universo Continuo (de Castro García, 2013).

Los pasos fundamentales que se realizan para la fusión de los valores lingüísticos son:

1. Selección del Conjunto Básico de Términos Lingüísticos (CBTL)

El operar sobre este tipo de información heterogénea, implica su modelado en un marco común de expresión para poder operar sobre ella. Este proceso consiste en unificar la información de entrada en un único dominio de expresión. Se lleva a cabo utilizando conjuntos difusos sobre el dominio lingüístico CBTL, y simbolizado por S_T. En este dominio se representa toda la información de entrada suministrada por los expertos y se transforma en información homogénea mediante conjuntos difusos. La selección del CBTL no es aleatoria, sino que tiene en cuenta los dominios utilizados en el problema.

2. Transformación en un Conjunto Difuso

Existen diferentes funciones de transformación que convierten la información heterogénea de entrada en conjuntos difusos, por lo que estas transformaciones se realizarán utilizando procesos de

comparación entre conjuntos difusos. Para transformar la información de entrada se utilizan medidas de semejanza. Las funciones de transformación son:

3. Transformación de Conjuntos Difusos sobre el CBTL en 2-tuplas Lingüísticas del CBTL

La función de transformación $x: F(S_T) \rightarrow S_T x\ [-0.5,0.5) = \check{S}$ que transforma los conjuntos difusos sobre el CBTL en 2-tuplas lingüísticas pertenecientes al CBTL. Una vez que todos los valores han sido unificados en 2-tuplas, la agregación puede realizarse utilizando los operadores de agregación. Seguidamente, los valores de preferencia colectiva son ordenados según los operadores de comparación (Peña Abreu, 2017).

1.4 Metodología de desarrollo de software

Dentro del desarrollo de software y con la necesidad de que los proyectos lleguen al éxito y obtener un producto de gran valor para los clientes, se hace necesario el empleo de una metodología que se ajuste a las características del equipo de desarrollo y las exigencias de los usuarios finales.

Las metodologías de desarrollo se pueden enmarcar en dos grandes grupos: tradicionales y ágiles. Las primeras enfatizan en el uso exhaustivo de documentación durante todo el ciclo de vida del proyecto. Son recomendadas para proyectos de grandes dimensiones y con grandes equipos de desarrollo. Las segundas dan importancia a la capacidad de respuesta a los cambios, se enfatiza en la satisfacción del cliente y promueven el trabajo en equipo (Figueroa, y otros, 2010).

Para guiar el proceso de desarrollo se escoge la Metodología para la Actividad Productiva de la UCI (AUP-UCI), variación del Proceso Unificado Ágil (AUP, por sus siglas en inglés) para la institución, orientada a guiar el proceso productivo de la UCI, motivo por el cual se escoge la misma. AUP-UCI tiene 3 fases: Inicio, Ejecución y Cierre; propone 7 disciplinas: Modelado de negocio, Requisitos, Análisis y diseño, Implementación, Pruebas internas y Pruebas de liberación, Pruebas de aceptación. Para la disciplina Requisitos, AUP-UCI define cuatro escenarios para modelar el sistema en los proyectos; de estos se selecciona el Escenario 3, que se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio con procesos muy complejos, independientes de las personas que los manejan y ejecutan, proporcionando objetividad, solidez, y su continuidad. Se debe tener presente que este escenario es muy conveniente si se desea representar una gran cantidad de niveles de detalles y la relaciones entre los procesos identificados (Rodríguez Sánchez, 2015).

1.5 Herramientas, tecnologías y lenguajes para el desarrollo

A continuación, se describen las herramientas, tecnologías y lenguajes a utilizar en el desarrollo de la propuesta de solución.

1.5.1 Herramienta de modelado

De acuerdo con (Kendall, y otros, 2005) la ingeniería de software asistida por computadora (*Computer Aided Software Engineering, CASE*) es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo, su objetivo es acelerar el proceso para el que han sido diseñadas, para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas.

Visual Paradigm for UML es una herramienta *CASE* que soporta el modelado mediante UML. Proporciona asistencia a los analistas, ingenieros de software y desarrolladores, durante todo el ciclo de vida de desarrollo de un software. Puede integrarse con otras aplicaciones, como herramientas ofimáticas. Permite generar código de forma automática, reduciendo los tiempos de desarrollo y evitando errores en la codificación del software. Además de generar diversos informes a partir de la información introducida en la herramienta (Paradigm, 2015).
Se selecciona *Visual Paradigm* en su versión 8.0 para la creación de los diagramas necesarios para el desarrollo de la solución.

1.5.2 Lenguaje de modelado

El Lenguaje Unificado de Modelado (*Unified Modeling Language*, UML) es un lenguaje de modelado visual usado para especificar, visualizar, construir y documentar artefactos de un software. Capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Brinda apoyo a la mayoría de los procesos de desarrollo orientados a objetos (Rumbaugh, y otros, 2000). Se selecciona el lenguaje de modelado UML en su versión 2.0 para la confección de varios productos de trabajo de la solución.

1.5.3 Lenguajes de programación

Del lado del cliente se selecciona el *HyperText Markup Lenguage 5* (HTML5, Lenguaje de marcado de hipertexto). Este lenguaje se utiliza para describir la estructura de las páginas web. Según (W3C, 2016) HTML da a los desarrolladores los medios para:

- Publicar documentos en línea con encabezados, texto, tablas, listas, fotos, entre otros.
- Recuperar información en línea a través de enlaces de hipertexto, con solo presionar un botón.
- Diseñar formularios para realizar transacciones con servicios remotos, para usar en la búsqueda de información, realizar reservas, solicitar productos, etc.
- Incluya hojas de cálculo, videoclips, clips de sonido y otras aplicaciones directamente en sus documentos.

Es un estándar a cargo del *World Wide Web Consortium*[4] (W3C) o Consorcio WWW. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la *World Wide Web* (WWW). Es el estándar que se ha impuesto en la visualización de páginas web y es el que todos los navegadores actuales han adoptado.

JavaScript es el lenguaje interpretado orientado a objetos desarrollado por *Netscape* que se utiliza en millones de páginas web y aplicaciones de servidor en todo el mundo. JavaScript de Netscape es un superconjunto del lenguaje de scripts estándar de la edición de ECMA-262 3 (*ECMAScript*) que presenta sólo leves diferencias respecto a la norma publicada. Es un lenguaje de programación dinámico que soporta construcción de objetos basado en prototipos. *JavaScript* puede funcionar como lenguaje procedimental y como lenguaje orientado a objetos. Los objetos se crean programáticamente añadiendo métodos y propiedades a lo que de otra forma serían objetos vacíos en tiempo de ejecución, en contraposición a las definiciones sintácticas de clases comunes en los lenguajes compilados como C++ y *Java*. Una vez se ha construido un objeto, puede usarse como modelo (o prototipo) para crear objetos similares (MDN, 2018). Se escoge J*avaScript* en su versión v1.8.

jQuery es una biblioteca multiplataforma de *JavaScript,* creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM[5], manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. *jQuery* es la biblioteca de *JavaScript* más utilizada (W3Techs, 2018). *jQuery* es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT[6] y la Licencia Pública General de GNU[7] v2, permitiendo su uso en proyectos libres y privados (JQuery, 2018). jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio. Se utiliza jQuery en su versión v3.2.1.

Hypertext Preprocessor (PHP, Preprocesador de hipertexto) es un lenguaje de código abierto del lado del servidor, especialmente adecuado para el desarrollo web de contenido dinámico. Puede ser utilizado en cualquier sistema operativo y servidor web. Posibilita la utilización de programación por procedimientos o programación orientada a objetos (POO), o una mezcla de ambas (PHP, 2015). Se selecciona el lenguaje PHP en su versión v5.3, el cual permite:

- Soporte para bases de datos: MySQL, PostgreSQL, Oracle, entre otras.

[4] Organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación.
[5] *Document Object Model* (Modelo de Objetos del Documento) es esencialmente una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos HTML, XHTML y XML. A través del DOM, los programas pueden acceder y modificar el contenido, estructura y estilo de los documentos HTML y XML.
[6] Es una de las tantas licencias de software que se originan en el Instituto Tecnológico de Massachusetts (MIT, Massachusetts Institute of Technology).
[7] Es una licencia de derecho de autor ampliamente usada en el mundo del software libre y código abierto.

- Integración con varias bibliotecas externas, permite generar documentos en PDF (documentos de Acrobat Reader).
- Admite servidores web como Apache.

Se escogen estos lenguajes por sus características y las facilidades que brindan para la implementación, además de que el equipo de desarrollo posee conocimientos sobre ellos y tiene experiencia en el trabajo con estos.

1.5.4 Marco de trabajo

Un marco de trabajo o *framework* es un conjunto de componentes físicos y lógicos estructurados de manera que permiten ser reutilizados en el diseño y desarrollo de nuevos sistemas de información. Permiten la utilización de modelos arquitectónicos como Cuatro Capas y Modelo Vista Controlador (*Model View Controller*, MVC) (Recaman, 2012).

Bootstrap es un *framework* web o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web del lado del cliente. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de *JavaScript* adicionales. A diferencia de muchos *frameworks* web, solo se ocupa del desarrollo *front-end*. *Bootstrap* es uno de los proyectos más destacado en *GitHub* y es usado por varias organizaciones (GitHub, 2010). Se utiliza Bootstrap en su versión v3.3.7.

AngularJS es un proyecto de código abierto, escrito en *Javascript* que contiene un conjunto de librerías útiles para el desarrollo de aplicaciones SPA (*Single Page Applications*), y propone una serie de patrones de diseño para llevarlas a cabo. En pocas palabras, es lo que se conoce como un *framework* para el desarrollo, en este caso con programación del lado del cliente. *AngularJS* además promueve y usa patrones de diseño de software, como el conocido MVC (Modelo-Vista-Controlador), aunque en una variante muy extendida en el mundo de *Javascript*. Básicamente estos patrones marcan la separación del código de los programas dependiendo de su responsabilidad. Eso permite repartir la lógica de la aplicación por capas, lo que resulta muy adecuado para aplicaciones de negocio y para las aplicaciones SPA, sobre todo si las mismas cuentan con una API REST en el servidor (Basalo, y otros, 2014). Para la implementación se utiliza la versión v5.0 de *AngularJS*.

Symfony es un marco de trabajo diseñado para optimizar el desarrollo de aplicaciones web. Utiliza el patrón arquitectónico MVC que separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Automatiza tareas comunes, permite al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. Está desarrollado completamente con PHP. Es

compatible con la mayoría de gestores de bases de datos, como *MySQL*, *PostgreSQL* y *Oracle* (Eguiluz, 2012).

Características:

- Extensible: desarrollado en su totalidad alrededor de *Bundles*, los cuales son directorios que contienen todo tipo de archivos (clases php y archivos web como *JavaScript*, css e imágenes), dentro de una estructura jerarquizada de directorios.
- Flexible: cuenta con un *micro-kernel* basado en contenedores de inyección de dependencia y en disparadores de eventos.
- Pensado para desarrolladores: provee herramientas que mejora la productividad de los desarrolladores, como por ejemplo la barra de *debug*, el soporte nativo para diferentes entornos, los errores y excepciones en páginas detalladas.
- Usabilidad avanzada: posee una simple API (Interfaz de Programación de Aplicaciones) que brinda la posibilidad de usar el marco de trabajo con gran facilidad.

Se selecciona como marco de trabajo *Symfony* en su versión v3.4 por las características y facilidades que brinda, además de contar con abundante documentación. Es sencillo y fácil de entender por parte de los programadores.

1.5.5 Sistema Gestor de Base de Datos (SGBD)

Un sistema gestor de base de datos es un conjunto de programas que administran y gestionan la información contenida en una base de datos. Entre las acciones que realiza se encuentran: definición de los datos, mantenimiento de la integridad de los datos dentro de la base de datos control de la seguridad y privacidad de los datos y manipulación de los datos (Alvarez, 2007).

PostgreSQL es un poderoso sistema de gestión de base de datos objeto-relacional de código abierto. Tiene más de 15 años de desarrollo activo y una arquitectura comprobada que le ha valido una sólida reputación de fiabilidad, integridad de datos y corrección (PostgreSQL, 2018). Utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afecta al resto y el sistema continua funcionando. Funciona bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema (García García, y otros, 2015).

Características:

- Documentación completa y excepcional.
- Soporta diferentes tipos de datos, incluyendo *INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL y TIMESTAMP*, entre otros.
- Se ejecuta en los principales sistemas operativos incluidos *Linux, UNIX* (*AIX, BSD, HP-UX, macOS, Solaris*) y *Windows* (PostgreSQL, 2018).

Por las características antes mencionadas se escoge *PostgreSQL* en su versión v5.3 como sistema gestor de base datos, además de que el marco de trabajo Symfony3 trae definido.

1.5.6 Herramienta de mapeo objeto relacional

El mapeo objeto-relacional (más conocido por su nombre en inglés, *Object-Relational Mapping*, o sus siglas ORM) es una técnica de programación que posibilita el acceso a una base de datos relacional desde una aplicación desarrollada según los principios de la programación orientada a objetos, haciendo uso de una interfaz que traduce los objetos en registros de las tablas y viceversa. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo) (Fernández Díaz, 2012).

Doctrine v2.0 es un asignador relacional de objetos (ORM) para PHP v5.3.0 o superior que proporciona persistencia transparente de objetos PHP situado en la parte superior de una poderosa capa de abstracción de base de datos (DBAL por *DataBase Abstraction Layer*) (Team, 2011). Una de las características clave de *Doctrine* es la opción de escribir las consultas de base de datos en un dialecto SQL propio orientado a objetos llamado Lenguaje de Consulta *Doctrine* (DQL por *Doctrine Query Language*), inspirado en *Hibernates* HQL. Además, DQL difiere ligeramente de SQL en que abstrae considerablemente la asignación entre las filas de la base de datos y objetos, permitiendo a los desarrolladores escribir poderosas consultas de una manera sencilla y flexible (Team, 2011).

Dentro de las principales ventajas que aporta el ORM es la reutilización, permitiendo llamar a los métodos de un objeto de datos desde varias partes de la aplicación e incluso desde diferentes aplicaciones. Además, la utilización de objetos en vez de registros y de clases en vez de tablas, tiene otra ventaja: permite añadir métodos de acceso en los objetos que no tienen relación directa con una tabla (LIBROSWEB, 2010).

1.5.7 Servidor web

Un servidor web es un “programa que atiende y responde a las diversas peticiones de los navegadores, proporcionándoles los recursos que solicitan mediante el protocolo HTTP o HTTPS4” (Mateu, 2004).

Apache es un servidor web de código abierto para la creación de páginas y servicios web. Es un servidor multiplataforma, gratuito, robusto y seguro (Apache, 2014).
Se selecciona como servidor web *Apache* en su versión 2.4.4 por ser multiplataforma, cuenta con abundante documentación y soporta varios lenguajes de programación, dentro de los cuales se encuentra el seleccionado para la solución.

1.5.8 Entorno de Desarrollo Integrado

Un IDE (*Integrated Development Environment*) es un programa compuesto por un conjunto de herramientas de programación. Puede dedicarse a un solo lenguaje de programación o a varios. Consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (Rouse, 2007).

NetBeans es un IDE hecho principalmente para Java, pero puede ser utilizado para cualquier lenguaje de programación. Es una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Es un producto libre y gratuito sin restricciones de uso. Facilita el trabajo mediante auto completamiento de código, visor de clases, métodos y componentes (NetBeans, 2015).
Se selecciona el IDE *NetBeans* en su versión v8.2 por ser un producto libre, además de brindar facilidades para la implementación de la solución.

1.5.9 Arquitectura de software

La arquitectura representa un modelo relativamente pequeño, intelectualmente tratable, de la forma en que un sistema se estructura y sus componentes se entienden entre sí; este modelo es transferible a través de sistemas; en particular, se puede aplicar a otros sistemas que exhiben requerimientos parecidos y puede promover reutilización en gran escala. El diseño arquitectónico soporta reutilización de grandes componentes o incluso de framework en los que se pueden integrar componentes (Reynoso, 2004).

Según Pressman, la arquitectura de software es la representación que capacita al ingeniero del software para: analizar la efectividad del diseño para la consecución de los requisitos fijados, considerar las alternativas arquitectónicas en una etapa en la cual hacer cambios en el diseño es relativamente fácil, y reducir los riesgos asociados a la construcción del software (Pressman, 2010).

Sin embargo, para este trabajo se utilizará la definición oficial establecida por la IEEE: “La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.”

1.5.10 Patrón arquitectónico

Los patrones arquitectónicos, o patrones de arquitectura, también llamados arquetipos ofrecen soluciones a problemas de arquitectura de software en ingeniería de software. Dan una descripción de los elementos y el tipo de relación que tienen junto con un conjunto de restricciones sobre cómo pueden ser usados. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. En

comparación con los patrones de diseño, los patrones arquitectónicos tienen un nivel de abstracción mayor (Pressman, 2010).

Patrón Modelo-Vista-Controlador (MVC):

El patrón arquitectónico Modelo Vista Controlador (Model View Controller, MVC por sus siglas en inglés) es un estilo arquitectónico que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos. El estilo arquitectónico MVC se ve frecuentemente en aplicaciones web, donde la vista es la interface de usuario y el código es el que provee de datos dinámicos a la página; el modelo es el Sistema de Gestión de Base de Datos y la lógica de negocio; y el controlador es el responsable de recibir los eventos de entrada desde la vista (Patrones, 2010).

El Modelo es el responsable de:

- Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.
- Define las reglas de negocio (la funcionalidad del sistema).
- Lleva un registro de las vistas y controladores del sistema.
- Si estamos ante un modelo activo, notificará a las vistas los cambios que en los datos puedan producir un agente externo.

El Controlador es responsable de:

- Recibir los eventos de entrada.
- Contiene reglas de gestión de eventos, estas acciones pueden suponer
- peticiones al modelo o a las vistas.

Las Vistas son responsables de:

- Recibir datos del modelo mediante el controlador y las muestras al usuario.
- Tienen un registro de su controlador asociado.
- Pueden dar el servicio de "Actualización ()", para que sea invocado por el controlador o por el modelo cuando es un modelo activo.

1.5.11 Patrones de diseño

Un patrón es un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlos en nuevas situaciones (Larman, 1999).

Los patrones de diseño (del inglés *design patterns*) son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño es una solución a un problema del diseño (Gamma). Los patrones de diseño pretenden:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas de software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Patrones *GRASP*:

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos. El nombre se eligió para indicar la importancia de captar (*grasping*) estos principios, si se quiere diseñar eficazmente el software orientado a objetos (Larman, 2003). A continuación, se mencionan los patrones que se utilizaron durante el desarrollo de la investigación:

Experto: El patrón experto en información es el principio básico de asignación de responsabilidades. Indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtiene un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento).

Creador: El patrón creador nos ayuda a identificar quién debe ser el responsable de la creación o instanciación de nuevos objetos o clases. La nueva instancia deberá ser creada por la clase que tiene la información necesaria para realizar la creación del objeto, o usa directamente las instancias creadas del objeto.

Controlador: El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control. Se recomienda dividir los eventos del sistema en el mayor número de controladores para poder aumentar la cohesión y disminuir el acoplamiento.

Alta Cohesión: Este patrón determina, que la información almacenada en una clase debe ser coherente y estar relacionada con esta, en mayor medida y enfocada en sus responsabilidades. Al realizar un diseño donde las clases del componente mantengan una alta cohesión como por ejemplo las clases controladoras, es posible ganar en claridad y facilidad a la hora de entender el diseño, además de simplificar el mantenimiento y soportar mayor capacidad de reutilización.

Bajo acoplamiento: Consiste en tener las clases lo menos relacionadas entre sí, para que, en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión en las demás, potenciando la reutilización y disminuyendo la dependencia entre las clases.

Patrones GoF:

Son patrones de diseño publicados en el libro *Design Patterns: Elements of Reusable Object-Oriented Software* por Gamma, Helm, Jonson y Vlissides conocidos mundialmente por *Gang of Four* o Pandilla de los cuatro. Se clasifican en creacionales, estructurales y de comportamiento (Gamma, y otros).

- Los patrones creacionales se encargan de la creación de los objetos ayudando a que el sistema sea independiente de la creación, composición y representación de los objetos.
- Los patrones estructurales son los encargados de cómo las clases y objetos están compuestos para formar estructuras más grandes. Los patrones estructurales usan la herencia para componer interfaces u objetos en tiempo de ejecución.
- Los patrones de comportamiento plantean algoritmos y la asignación de responsabilidades entre objetos. Estos patrones no sólo describen clases y objetos sino también describen la comunicación entre ellos.

Patrón Factory Method: Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar; su objetivo es devolver una instancia de múltiples tipos de objetos, que normalmente provienen de una misma clase padre y sólo se diferencian entre ellos por algún aspecto de comportamiento (Gamma, y otros).

Patrón Decorador: El principal objetivo de este patrón es añadir responsabilidades a objetos concretos de manera dinámica y transparente sin afectar a otros objetos. Este patrón brinda más flexibilidad que la herencia estática y evita que las clases más altas en la jerarquía estén demasiado cargadas de funcionalidad y sean complejas (Gamma, y otros).

Patrón Inyección de dependencias: Este patrón es utilizado en los contenedores de servicios (o contenedores de inyección de dependencias), los cuales son objetos PHP que gestionan la creación de instancias de servicios, es decir, objetos (Gamma, y otros).

Conclusiones parciales

El estudio realizado en el presente capítulo permitió abordar sobre las herramientas, métodos y modelos para el análisis de factibilidad en proyectos informáticos, lo cual permitió la selección del modelo MFac-PS, para el desarrollo de la herramienta ya que este tiene la ventaja de tratar la incertidumbre. A partir de este estudio se enunciaron los principales términos asociados al dominio de la investigación, lo que permitió adquirir un mayor conocimiento sobre el objeto de estudio. Se ha realizado el análisis del

lenguaje de modelado empleado y de la herramienta CASE seleccionada para la aplicación durante todo el desarrollo del software. Realizada una revisión de los principales elementos teóricos, se decide desarrollar una aplicación web mediante el uso de las herramientas, tecnologías y lenguajes mencionados.

CAPÍTULO II: PROPUESTA DE SOLUCIÓN

Introducción

En el presente capítulo se toman los conceptos que se encuentran en el modelo de dominio para a partir de los mismos identificar y confeccionar los prototipos de interfaz. Se presentan los requisitos funcionales obtenidos mediante técnicas de tormenta de ideas y entrevista, así como los prototipos empleados para su validación. Se muestran las propiedades necesarias para el correcto desarrollo del sistema a través de los requisitos no funcionales. Resultado del análisis, se presenta el diagrama del proceso de negocio y las descripciones de requisitos por proceso. Resultado del diseño, se describe la arquitectura, el modelo de datos y los patrones aplicados. También se obtienen los artefactos generados a partir de la aplicación de la metodología de desarrollo utilizada y una descripción de las entidades que tienen persistencia en la base de datos.

2.1 Descripción general de la propuesta de solución

Para darle cumplimiento al objetivo general previamente planteado, se describe la Herramienta para la selección de criterios de factibilidad para evaluar proyectos informáticos (XiSCriF-PS), el cual contribuirá en la predicción de la factibilidad influyendo en el éxito de estos proyectos. En el proceso de selección recibirá como entrada un conjunto de participantes, los cuales propondrán los requisitos para la selección de expertos mediante una ronda del método Delphi y estos serán evaluados con el modelo lingüístico 2-tuplas; luego de obtenidos los requisitos se seleccionan los expertos que cumplen con estos. Los expertos proponen los criterios candidatos para el análisis de factibilidad mediante una nueva ronda del método Delphi y para evaluar estos se emplea el modelo lingüístico 2-tuplas; luego de obtenerse el listado de criterios base se procede a conceptualizar estos criterios para obtener el listado final de criterios para el análisis de la factibilidad. En la Figura 1 se muestra el proceso antes mencionado.

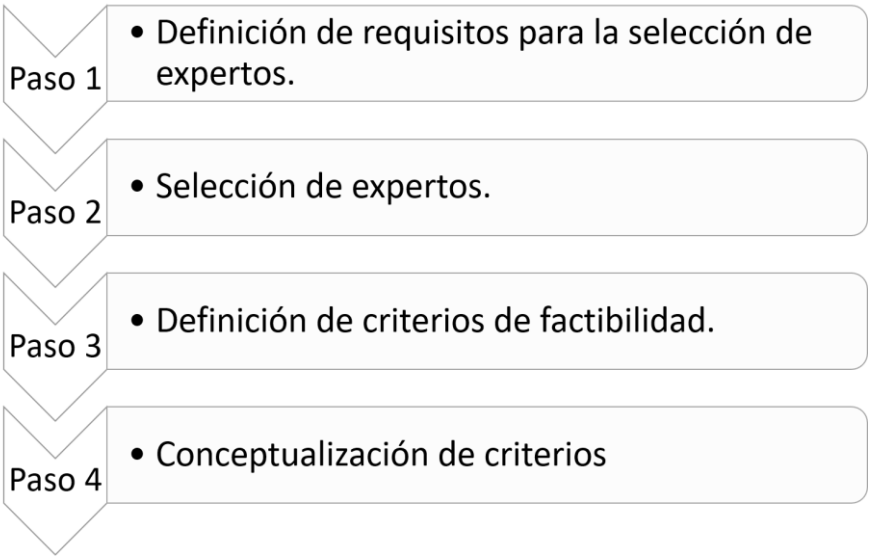

Figura 2. Pasos para la definición de criterios de factibilidad.

2.2 Modelo conceptual

Un modelo de dominio o modelo conceptual, muestra clases conceptuales significativas en un dominio del problema. Es una representación de dichas clases del mundo real, no de componentes de software (Larman, 2003). En UML se ilustra como un grupo de diagramas de estructura estática donde no se define ninguna operación.

La Figura 1 muestra los conceptos del dominio y las relaciones entre ellos.

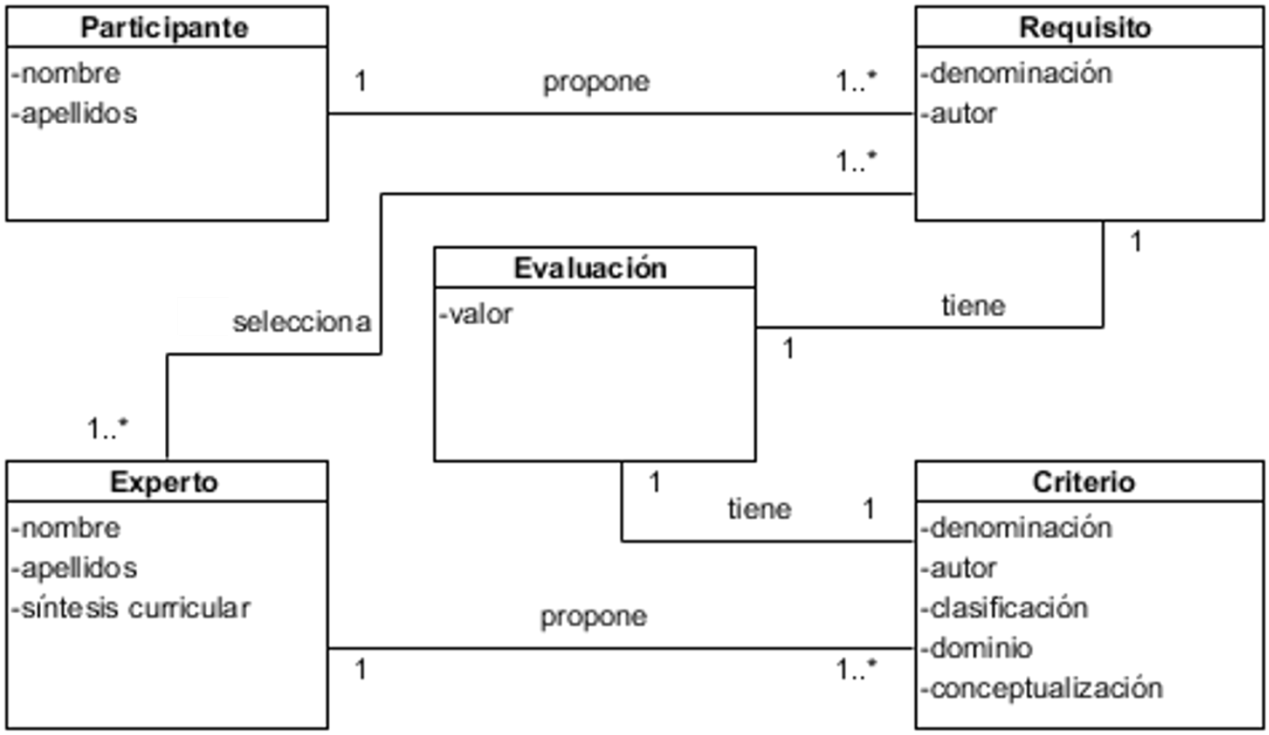

Figura 3. Modelo conceptual.

Descripción de los conceptos

Participante: Persona capacitada para enunciar los requisitos que en su opinión deben evaluarse para catalogar un experto en el área.

Evaluación: Representa la evaluación de los requisitos y los criterios de factibilidad por los participantes y los expertos respectivamente.

Experto: Persona capacitada en conocimientos sobre los estudios de factibilidad.

Requisito: Parámetro necesario que se debe cumplir para ser considerado experto en el área.

Criterio: Parámetro que se debe tener en cuenta para el análisis de la factibilidad en el proyecto a estudiar.

2.3 Descripción del proceso del negocio

Un diagrama es una herramienta visual muy intuitiva para la gestión del trabajo. Funciona muy bien para detectar y comunicar los pasos a seguir para lograr un propósito, así como los momentos críticos en donde el equipo debe prestar una especial atención (SINNAPS, 2018).

Un proceso se puede definir como "un conjunto de actividades, acciones o toma de decisiones interrelacionadas, caracterizadas por entradas y salidas, orientadas a obtener un resultado específico como consecuencia del valor añadido aportado por cada una de las actividades que se llevan a cabo en las diferentes etapas de dicho proceso" (García-Morales, 2002).

Los diagramas de procesos son la representación gráfica de los procesos y son una herramienta de gran valor para analizar los mismos y ver en qué aspectos se pueden introducir mejoras (García-Morales, 2002).

En la siguiente figura se muestra el diagrama del proceso de negocio.

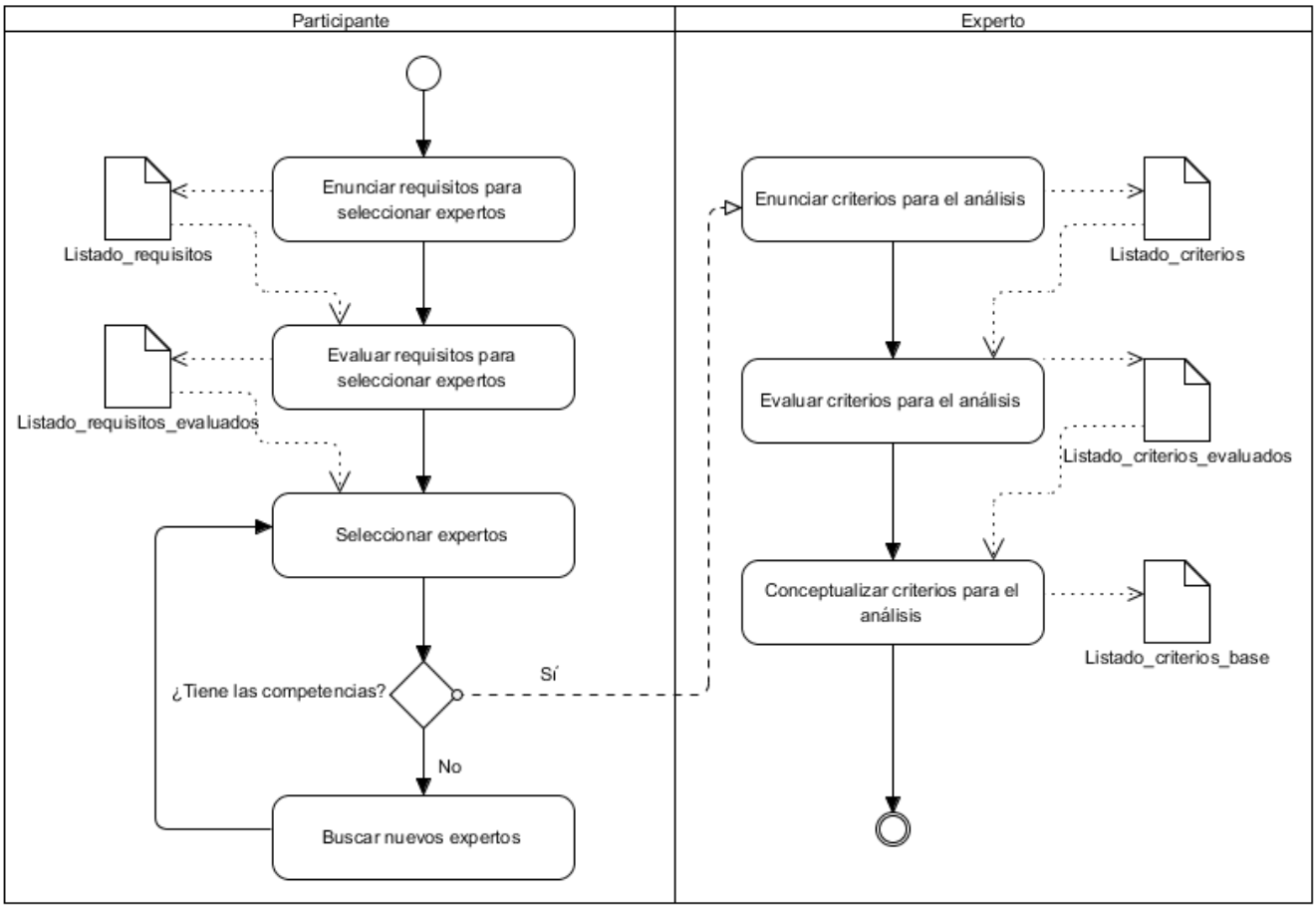

Figura 4. Diagrama de proceso del negocio.

2.4 Requisitos del software

Los requisitos de software expresan las necesidades y apremios colocados en un producto de software que contribuyen a la solución de un cierto problema del mundo real (Olazabal Arteaga, 2014). Como

parte del análisis previo al diseño de la solución se emplearon las técnicas: entrevista y tormenta de ideas para la obtención o captura de requisitos.

La entrevista es un método clásico que posibilita tomar conocimiento del problema y comprender los objetivos de la solución propuesta. Mediante esta técnica el equipo de desarrollo se acerca al problema de una forma natural consultando al cliente (IEEE, 2004), siendo estas de tipo abierta.

La tormenta de ideas es una técnica de reuniones en grupo cuyo objetivo es que los participantes muestren sus ideas de forma libre. Consiste en la acumulación de ideas y/o información sin ser evaluadas (Escalona, y otros, 2002).

Se enuncian seguidamente los requisitos no funcionales y funcionales del sistema, así como la descripción de requisitos por proceso.

2.4.1 Requisitos funcionales

Los requisitos funcionales (RF) describen el funcionamiento del sistema, la forma en que debe reaccionar ante ciertas entradas y cómo se debe comportar en situaciones específicas. Detallan la función del producto de software, entrada, salidas, excepciones y usuarios (Sommerville, 2007).

Luego de aplicadas las técnicas antes mencionadas se identificaron un total de 28 RF expuestos en la Tabla 2. La prioridad de cada requisito fue definida por el cliente en función de la importancia para el negocio.

Tabla 2. Requisitos funcionales.

No.	Requisito	Prioridad
RF_1	Autenticar usuario.	Media
RF_2	Adicionar usuario.	Media
RF_3	Modificar usuario.	Media
RF_4	Eliminar usuario.	Media
RF_5	Listar usuarios.	Baja
RF_6	Adicionar requisito.	Media
RF_7	Modificar requisito.	Media
RF_8	Eliminar requisito.	Media
RF_9	Listar requisitos.	Baja
RF_10	Evaluar requisito.	Alta
RF_11	Adicionar criterio.	Media
RF_12	Modificar criterio.	Media

RF_13	Eliminar criterio	Media
RF_14	Listar criterios.	Baja
RF_15	Evaluar criterio.	Alta
RF_16	Adicionar experto.	Media
RF_17	Modificar experto.	Media
RF_18	Eliminar experto.	Media
RF_19	Listar expertos.	Baja
RF_20	Conceptualizar criterio.	Alta
RF_21	Adicionar tipo de requisito.	Media
RF_22	Modificar tipo de requisito.	Media
RF_23	Eliminar tipo de requisito.	Media
RF_24	Adicionar tipo de criterio.	Media
RF_25	Modificar tipo de criterio.	Media
RF_26	Eliminar tipo de criterio.	Media
RF_27	Adicionar tipo de dominio.	Media
RF_28	Modificar tipo de dominio.	Media
RF_29	Eliminar tipo de dominio.	Media

Descripción de requisitos por proceso

La descripción de requisitos por proceso (DRP) es un instrumento para el levantamiento de requerimientos para el desarrollo de un software, también conducen el proceso de las pruebas de aceptación, que son empleados para verificar que las DRP han sido implementadas correctamente. Teniendo en cuenta el escenario escogido en la disciplina Requisito se muestran a continuación la DRP correspondiente al requisito “Conceptualizar criterio” (Ver **Tabla 3**), mientras que el resto se puede consultar en el expediente del proyecto.

Tabla 3. Descripción de requisito por proceso. Requisito "Conceptualizar criterio".

Precondiciones	El usuario debe estar autenticado en el sistema. El usuario debe poseer los permisos de experto para conceptualizar criterios. Debe haber sido adicionado al menos un criterio con anterioridad.
Flujo de eventos	
Flujo básico Conceptualizar criterio	
1.	El sistema muestra una lista de los criterios existentes. Muestra los datos de Criterio y Estado.
2.	El usuario da clic sobre el criterio que desea conceptualizar.

3.	El sistema muestra un formulario con los datos del criterio: - Id - Criterio - Clasificación - (*) Dominio: Tipo de dominio. Lista desplegable con los distintos nomencladores para el dominio de un criterio. - (*) Conceptualización. Un campo de texto para introducir la conceptualización del criterio y las opciones
4.	Se introducen los datos para conceptualizar criterio
5.	El sistema brinda las siguientes opciones: - Aceptar - Cancelar
6.	El usuario selecciona la opción Aceptar.
7.	El sistema valida los datos.
8.	El sistema guarda los datos del criterio.
9.	Concluye así el requisito.
Pos-condiciones	
1.	Se listaron los criterios existentes.
Flujos alternativos	
Flujo alternativo 7.a Información incompleta	
1.	El sistema señala el o los campos obligatorios que no hayan sido introducidos y muestra el mensaje de información: Este campo es obligatorio; indicando el o los campos en cuestión.
2.	El usuario introduce los datos.
3.	Volver al paso 7 del flujo básico.
Pos-condiciones	
1.	N/A
Flujo alternativo 5.a Cancelar cuando se hayan introducido datos	
1.	El usuario selecciona la opción Cancelar.
2.	El sistema muestra un mensaje de información: ¿Está seguro que desea cancelar la operación?
3.	El sistema brinda las siguientes opciones: - Aceptar - Cancelar
4.	El usuario selecciona la opción Aceptar.
5.	El sistema no guarda los datos introducidos y regresa a la lista de criterios.
6.	Concluye así el requisito.

Pos-condiciones		
1.	No se modifica la entidad criterio.	
Flujo alternativo 5.b Descartar cuando se hayan introducido datos. Opción *Cancelar*		
1.	El usuario selecciona la opción cancelar.	
2.	El sistema mantiene los cambios realizados.	
3.	Volver al paso 5 del flujo básico.	
Validaciones		
1.	Cliente	
Conceptos	N/A	N/A
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

Prototipo elemental de interfaz de usuario:

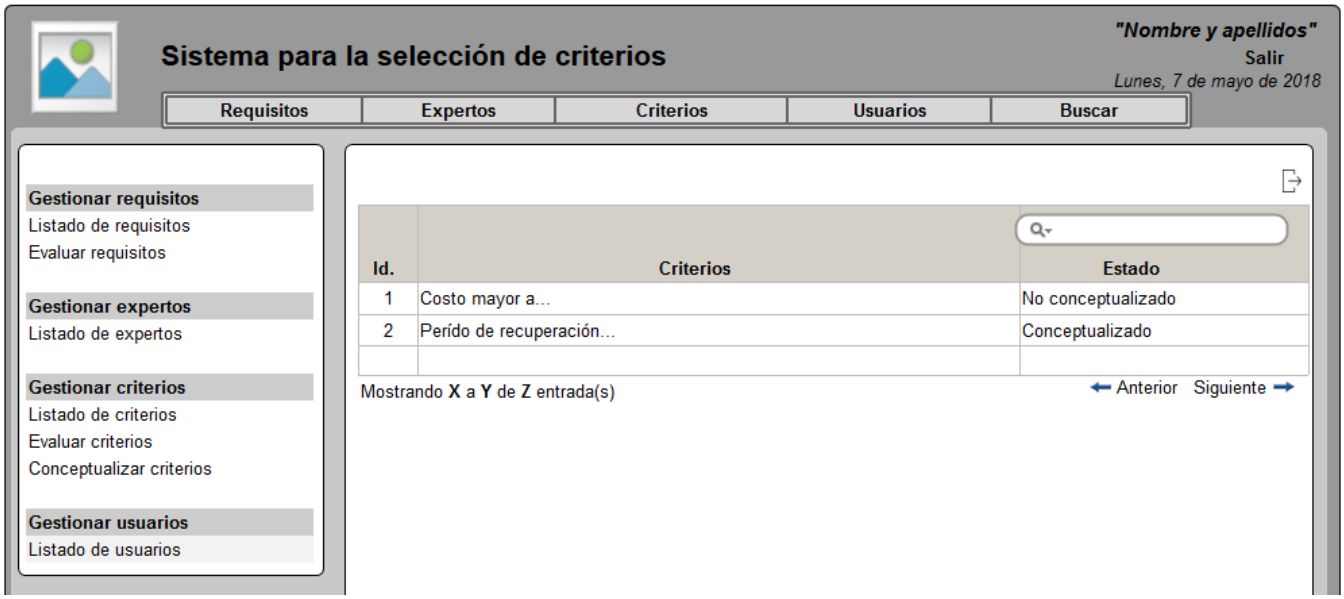

Figura 5. Prototipo elemental de interfaz de usuario "Conceptualizar criterio" (Listado de criterios base).

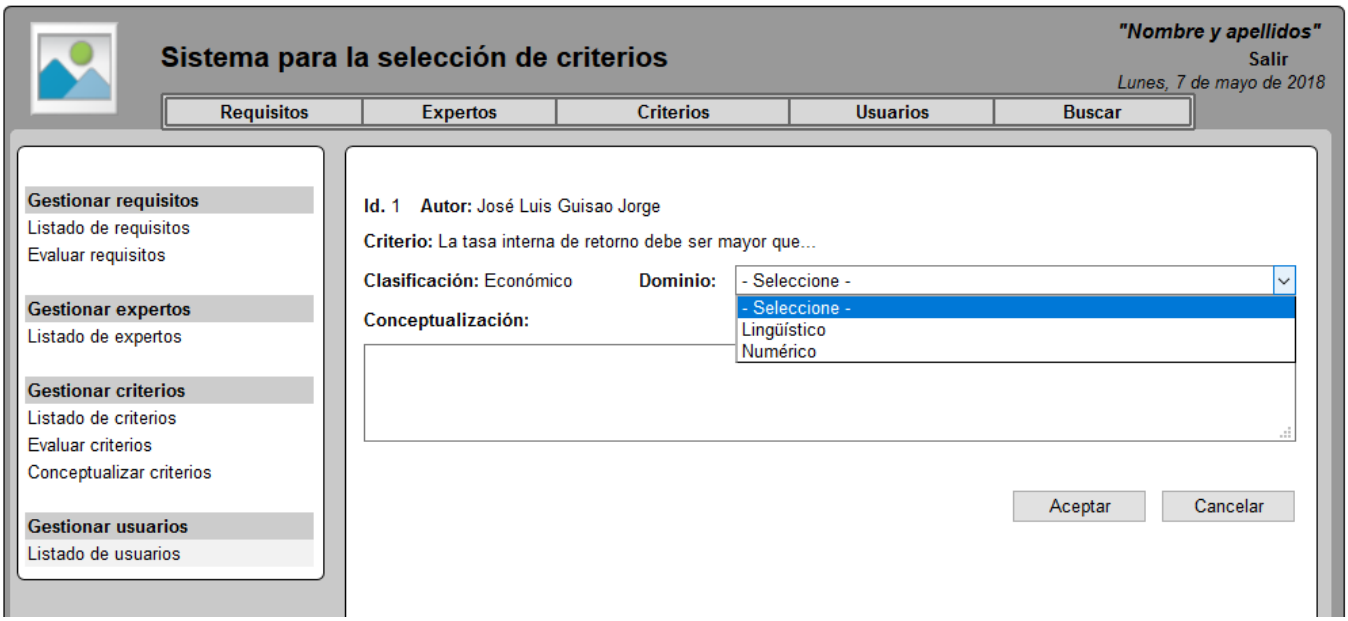

Figura 6. Prototipo elemental de interfaz de usuario "Conceptualizar criterio" (Conceptualizar).

2.4.2 Requisitos no funcionales

Los requisitos no funcionales definen propiedades del sistema, tales como: tiempo de respuesta, necesidades de almacenamiento, fiabilidad, usabilidad y seguridad (Sommerville, 2007). A continuación, se listan los requisitos no funcionales (RNF) de la aplicación a desarrollar.

Tabla 4. Requisitos no funcionales.

No.	Requisito
Usabilidad	
RNF_1	La aplicación debe presentar una vista sencilla, descriptiva y fácil de usar, con el objetivo de proporcionar a los usuarios un mejor entendimiento con la aplicación.
Portabilidad	
RNF_2	La aplicación debe poder instalarse en los sistemas operativos Windows y Linux/Unix.
Fiabilidad	
RNF_3	La aplicación restringirá el acceso por roles a los usuarios autorizados.
Disponibilidad	
RNF_4	La aplicación deberá estar disponible siempre, asegurando el acceso desde cualquier lugar de red a los usuarios autorizados.
Tiempo de respuesta	
RNF_5	El tiempo de respuesta en el procesamiento de los datos y solicitud de estos no debe sobrepasar los 5 segundos.
Interfaz	

RNF_6	La aplicación debe ofrecer una interfaz amigable y un diseño sencillo que le permita al usuario interactuar fácilmente.
RNF_7	La aplicación, al ocurrir un error con los datos de entrada muestra mensajes al usuario informando este.
Software	
RNF_8	El servidor de aplicación debe tener instalado el servidor web Apache y el marco de trabajo Symfony3.
RNF_9	El servidor de base de datos debe tener como Gestor de Base de Datos PostgreSQL en su versión 9.2 o superior.
RNF_10	Las PC clientes deben tener instalado un navegador web (Mozilla Firefox, Google Chrome); se recomienda para estas Mozilla Firefox en su versión 54.0 o superior.
Hardware	
RNF_11	La PC servidor debe tener 2 GB de memoria RAM o superior, un disco duro de 320 GB de almacenamiento o superior, un procesador Pentium IV 2.4 GHz o superior.
RNF_12	Las PC clientes deben tener 512 MB, como mínimo de memoria RAM, un procesador Pentium IV 1.7 GHz o superior.
Seguridad	
RNF_13	Existirán diferentes tipos de usuarios según las acciones que puedan realizar. Los permisos serán limitados, basados en los roles definidos y el usuario no podrá gestionarlos.
RNF_14	La aplicación debe permitir que la contraseña se almacene de manera encriptada en la base de datos.
RNF_15	La aplicación debe permitir la comprobación de credenciales en la autenticación del usuario en el servidor de aplicaciones y no en el servidor de base de datos, para evitar inyecciones SQL que falseen la autenticación y provoquen la suplantación de identidad.

2.5 Arquitectura de software

La arquitectura de software se define, a grandes rasgos como las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos. La arquitectura de software es de especial importancia ya que es la manera en que se estructura un sistema; tiene un impacto directo sobre la capacidad de este para satisfacer lo que se conoce como los atributos de calidad del sistema (Cervantes, 2018).

2.5.1 Patrón arquitectónico

Un patrón de arquitectura de software describe un problema particular y recurrente del diseño, que surge en un contexto específico, y presenta un esquema genérico y probado de su solución (Oktaba, 2002). Symfony 2 basa su funcionamiento interno en el estilo Modelo-Vista-Controlador (MVC), uno de los más usados por los frameworks web (Eguiluz, 2011). Este se aplicará para el diseño de las clases y propone tres capas fundamentales como lo indica su nombre, las cuales se describen a continuación.

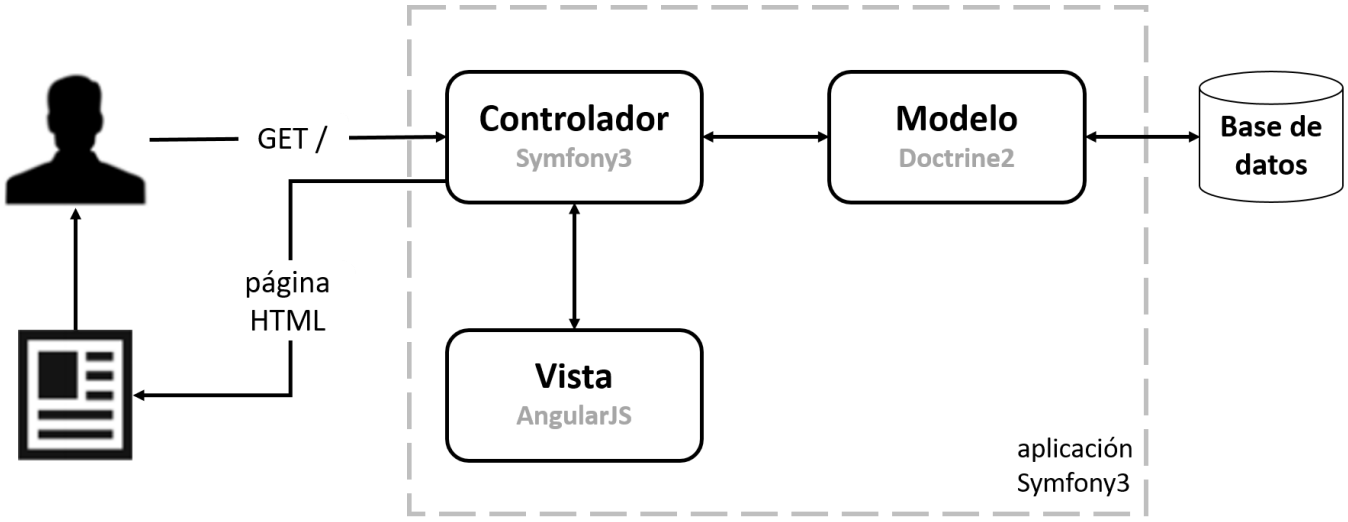

Figura 7. Funcionamiento del patrón arquitectónico Modelo-Vista-Controlador.

Vista

La capa de presentación o vista es la encargada de proveer la interacción con el usuario y facilitar la visualización de las funcionalidades del sistema, tales como:

- Mostrar datos, eliminarlos, ordenarlos, solicitarlos, validarlos.
- Informar de los errores lógicos y de ejecución.
- Controlar la navegación entre pantallas.

Esta capa en XiSCriF-PS sería la encargada de visualizar al cliente los datos referentes al proceso de selección de criterios de factibilidad registrados en la base de datos.

Controlador

Esta capa contiene la funcionalidad que implementa la aplicación. Involucra cálculos basados en la información dada por el usuario y datos almacenados y validaciones. Controla la ejecución de la capa de acceso a datos y servicios externos. Se puede diseñar la lógica de la capa de negocios para uso directo por parte de componentes de presentación o su encapsulamiento como servicio y llamada a través de una interfaz de servicios que coordina la conversación con los clientes del servicio o invoca cualquier flujo o componente de negocio.

Esta capa en XiSCriF-PS sería la encargada de ajustar las funcionalidades para capturar los eventos y registrar los datos del proceso de selección de criterios. Se encontrarían también las clases controladoras encargadas de manejar la comunicación entre la vista y el modelo.

Modelo

La capa de datos o modelo no es más que un grupo de clases responsables del almacenamiento de los datos, esta incluye necesariamente un modelo de las entidades del dominio del negocio. El modelo, es la representación real de los datos y representa además la persistencia del estado del sistema.

Esta capa representaría todas las clases entidades. Cubriría también aquellas clases que almacenan las consultas para acceder a los registros de la base de datos.

2.6 Modelo de diseño

El modelo de diseño describe la realización física de los procesos, centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de programación, tienen impacto en el sistema a considerar, siendo una entrada fundamental para las actividades de implementación (Olazabal Arteaga, 2014).

2.6.1 Patrones de diseño utilizados

Un patrón de diseño es una abstracción de una solución en un nivel alto. Los patrones solucionan problemas que existen en muchos niveles de abstracción. Hay patrones que abarcan las distintas etapas del desarrollo, entre ellos los patrones GRASP (*General Responsability Assignment Software Patterns* o Patrones Generales para Asignar Responsabilidades) y GoF (*Gang of Four* o Banda de los Cuatro) (Giraldo, et al., 2011).

Patrones GRASP

Symfony 2 evidencia el uso de varios patrones de diseño que este trae incluidos por defecto en su arquitectura, además de estar concebidos de tal manera que obliga al programador a aplicarlos. Entre ellos tenemos:

- Experto: Es uno de los patrones que más se utiliza cuando se trabaja con *Symfony* 2, con la inclusión de *Doctrine* para mapear la base de datos. Este marco de trabajo utiliza este ORM para implementar su capa de abstracción en el modelo, encapsular toda la lógica de los datos y generar las clases con todas las funcionalidades comunes de las entidades, las clases de abstracción de datos poseen un grupo de funcionalidades que están relacionadas directamente con la entidad y contienen la información necesaria de la tabla que representan. Por ejemplo, la clase controladora “CriterioController.php”.
- Creador: En las clases controladoras se encuentran las funcionalidades con el sufijo *action*, aquí se encuentran las acciones definidas para el sistema y se ejecutan en cada una de ellas. En

dichas acciones se crean los objetos de las clases que representan las entidades, lo que evidencia cuáles funcionalidades son creadoras de dichas entidades. La clase controladora "CriterioGtr.php" es un ejemplo donde se evidencia este patrón.

- Controlador: Todas las peticiones web son manipuladas por un controlador frontal, ejemplo de esto son las clases "app.php" y "app_dev.php", que es el punto de entrada único de toda la aplicación en un entorno determinado. Este patrón se evidencia en las clases controladoras del sistema. En *Symfony* 2 hay una capa específicamente para los controladores, que son el núcleo de este marco de trabajo; aquí cada clase tiene su responsabilidad y es única.
- Alta Cohesión: Una de las características principales del marco de trabajo *Symfony* 2 es la organización del trabajo en cuanto a la estructura del proyecto. Realizando un diseño donde las clases mantengan una alta cohesión, permite que el software sea flexible a cambios sustanciales con efecto mínimo. Se emplea en las clases controladoras ya que fueron asignadas responsabilidades a las clases de forma tal que la cohesión siguiera siendo alta, o sea, cada clase se encargará de realizar solamente las funciones que estén en correspondencia con la responsabilidad que posea. Por ejemplo, la clase controladora "RequisitoController.php".
- Bajo acoplamiento: Este patrón se evidencia dentro del marco de trabajo, en las clases que implementan la lógica del negocio y de acceso a datos que se encuentran en el modelo, las cuales no tienen asociaciones con las de la vista, lo que proporciona que la dependencia en este caso sea baja. Como existe poca dependencia entre esas clases se permite una mayor reutilización. Un ejemplo donde se emplea este patrón es la clase entidad "Criterio.php".

Patrones GoF

Patrón Factory Method: En la implementación de la herramienta este patrón es utilizado cuando se realiza una petición al contenedor de servicios, en la que se le pasa por parámetros el servicio que tiene definido el repositorio que se desea instanciar; internamente el contenedor de servicios determina cuál es la entidad que corresponde a dicho repositorio y realiza una llamada a la clase "EntityManager.php", pasándole por parámetro la entidad calculada. Dicha clase es la encargada de implementar el patrón Factory Method, esto se realiza específicamente en el método getRepository(entityName).

Patrón Decorador: En la implementación de la herramienta este patrón se evidencia en el uso de una plantilla global "app.component.html" para las vistas que decorará las demás páginas de la aplicación.

2.6.2 Diagrama de clases del diseño

En el siguiente diagrama de clases del diseño se muestran las clases involucradas en el requisito "Conceptualizar criterio".

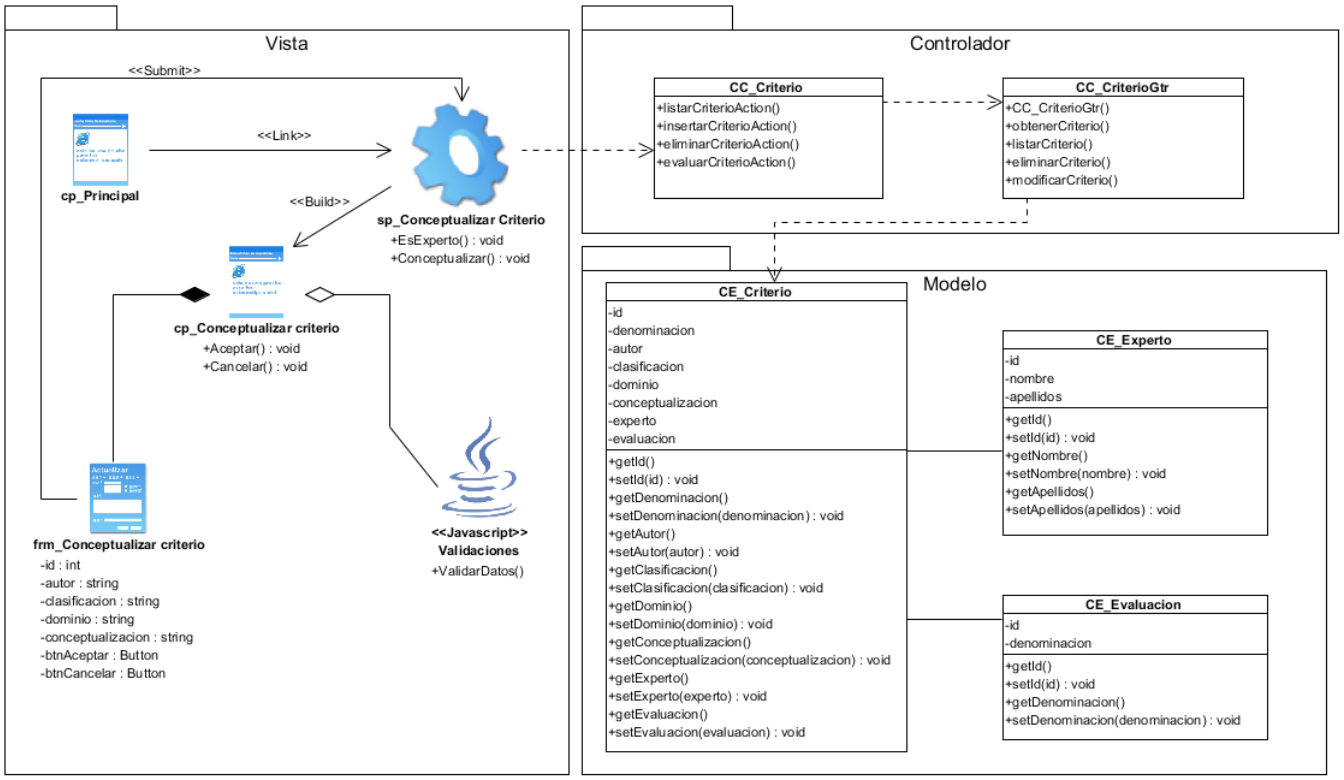

Figura 8. Diagrama de clases del diseño.

2.6.3 Modelo de datos

Un modelo de datos es una serie de conceptos que puede utilizarse para describir los datos de acuerdo con reglas y convenios predefinidos y luego ser manipularlos. Se utiliza para describir la estructura lógica y física de la información persistente gestionada por el sistema, así como la correlación entre las clases de diseño y las estructuras de datos persistentes. En otras palabras, permite describir las estructuras de datos de la base de datos, su tipo, descripción y la forma en que se relacionan (Ferrin González, 2016). A continuación, se presenta el modelo de base de datos de la herramienta.

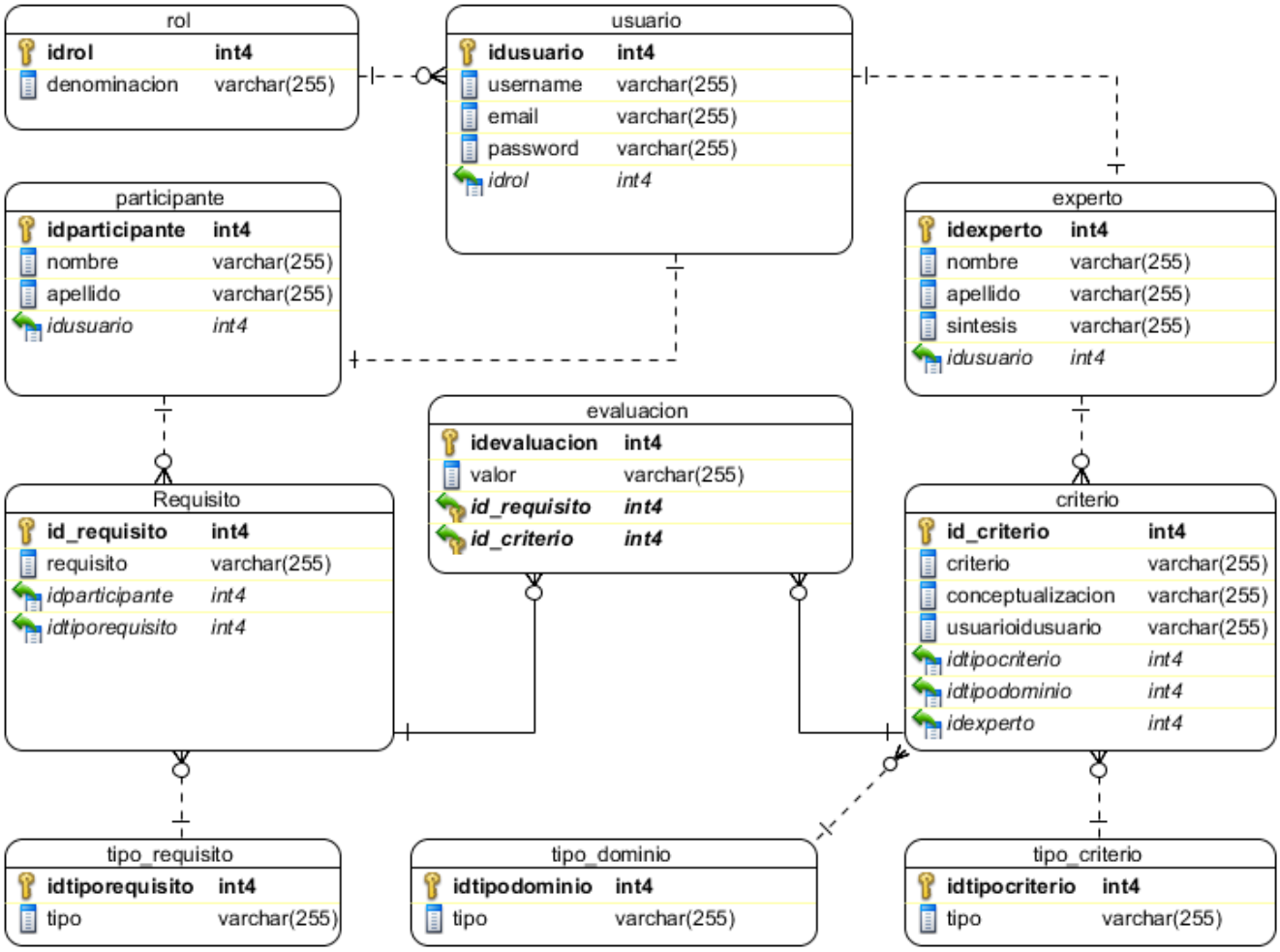

Figura 9. Modelo de la base de datos.

2.6.4 Diagrama de despliegue

Un diagrama de despliegue modela la arquitectura en tiempo de ejecución de un sistema**.** Esto muestra la configuración de los elementos de hardware (nodos) y muestra cómo los elementos y artefactos del software se trazan en esos nodos (SparxSystems, 2018).

A continuación, se presenta el diagrama de despliegue para el sistema:

Figura 10. Diagrama de despliegue para el sistema.

2.6.5 Estándares de codificación

A continuación, se describen los estándares de codificación utilizados para la implementación de la propuesta de solución.

Identación

El contenido siempre se identará con tabs, nunca utilizando espacios en blanco.

Cabecera del archivo

Es importante que todos los archivos *.php inicien con una cabecera específica que indique información de la versión, autor de los últimos cambios, etc. Es decisión de cada equipo decidir si se quieren o no agregar más datos.

```
/**
 * Clase Controladora "RequisitoController.php"
 * @autor: jlguisao
 * @modificado: 2/05/18
 */

namespace AppBundle\Controller\Api;


use ...

class RequisitoController extends BaseApiController
{
```

Figura 11. Cabecera del archivo.

Comentarios en las funciones

Todas las funciones deben tener un comentario, antes de su declaración, explicando que hacen. Ningún programador debería tener que analizar el código de una función para conocer su utilidad. Tanto el nombre como el comentario que acompañe a la función deben ser suficiente para entender el código.

```
//Esta función obtiene la información de los requisitos
    obtenerRequisito(): Observable<any> {
        return this.apiService.get(this.urlResource + 'obtenerRequisito');
    }
```

Figura 12. Comentarios en las funciones.

Ubicación y denominación de archivos.

Se ubicarán los archivos según las convenciones establecidas por *AngularJS* y *Symfony3*.
Para la denominación de los archivos se seguirán las convenciones establecidas por *AngularJS*, *Symfony3* y por el equipo de desarrollo. Ejemplo:

- Para las clases de gestión del negocio se usará el sufijo Gtr: RequisitoGtr.
- Para las clases controladoras se usará el sufijo Controller: RequisitoController.

Para la denominación de los servicios a crear se tendrán en cuenta los siguientes elementos:

- Para los servicios se utilizará la siguiente nomenclatura: módulo.service.ts (todo en minúscula utilizando como separador el carácter punto “.”).
    - Ejemplo:
      requisito.service.ts
- Para los componentes se usará la nomenclatura: nombre.component.ts.
- Para los modelos se usará la nomenclatura: nombre.ts.

Clases

Las clases serán colocadas en un archivo .php aparte, donde sólo se colocará el código de la clase. El nombre del archivo será el mismo del de la clase. Las clases siguen las mismas reglas de las funciones, por tanto, debe colocarse un comentario antes de la declaración de la clase explicando su utilidad. Los nombres de las clases deben iniciar con letra mayúscula. Si un nombre de clase se comprende de más de una palabra, la primera letra de cada nueva palabra debe comenzar con letra mayúscula. No se permiten las letras mayúsculas sucesivas; por ejemplo, una clase "ListadoPDF" no se permite, mientras " ListadoPdf" es aceptable.

Siempre utilizar las etiquetas <?php ?> para abrir un bloque de código. No utilizar el método de etiquetas cortas.

Estilo y reglas de escritura de código PHP.

- Nombres de variables:

Los nombres deben ser descriptivos y concisos. No usar grandes frases ni pequeñas abreviaciones para las variables. Siempre es mejor saber qué hace una variable con solo conocer su nombre. Esto se aplica para los nombres de variables, funciones, argumentos de funciones y clases. Los nombres de las variables y de las funciones pueden iniciar con letra minúscula, pero si estas tienen más de una palabra, cada nueva palabra debe iniciar con letra mayúscula. Las constantes deben de escribirse siempre en mayúsculas y tanto estas como las variables globales deben de tener como prefijo el nombre de la clase a la que pertenecen.

- Las definiciones de la función:

Los nombres de las funciones pueden contener solo caracteres alfanuméricos y siempre deben empezar en letras minúsculas. Cuando el nombre de una función conste de más de una palabra, la primera letra de cada nueva palabra debe comenzar con mayúscula.

```
public function obtenerRequisitosAction(RequisitoGtr $requisitoGtr)
{
    $listadoRequisitos = $requisitoGtr->listarRequisitos();
    return $this->view($listadoRequisitos,  statusCode: StatusCodes::HTTP_OK);
}
```

Figura 13. Definición de la función.

- Llamadas a funciones:

Deben llamarse las funciones sin los espacios entre el nombre de la función, el paréntesis de la apertura, y el primer parámetro; los espacios entre las comas y cada parámetro, y ningún espacial entre el último parámetro, el paréntesis del cierre, y el punto y coma.

```
public function insertarUsuarioAction(Request $request, UsuarioGtr $usuarioGtr)
{
    $usuarioGtr->insertar($request->request->all());
    return $this->viewSuccess('Operación realizada con éxito', StatusCodes::HTTP_CREATED);
}
```

Figura 14. Llamadas a funciones.

- Llaves

Las llaves de apertura irán al final de la sentencia que delimitan, y las de cierre alineadas con el inicio de la sentencia en una nueva línea.

```
public function insertarReqAction(Request $request, RequisitoGtr $requisitoGtr)
{

    return $this->viewSuccess( msg: 'Operación realizada con éxito',  code: StatusCodes::HTTP_CREATED);
}
```

Figura 15. Estándar de las funciones y las llaves.

Conclusiones parciales

En este capítulo fueron mostrados los productos de trabajo generados durante el diseño de la solución propuesta que permitieron describir y diseñar todos los artefactos ingenieriles necesarios para la posterior implementación. En el mismo se identificaron y describieron los requisitos funcionales y no funcionales, obtenidos a través de las técnicas antes descritas, lo cual permitió identificar las funcionalidades a implementar y los requerimientos necesarios para el correcto funcionamiento de la aplicación. Se conformó la arquitectura del sistema que rige el diseño, el modelo de datos, el diagrama de clases del diseño y el de despliegue, lo cual permitió tener una estructura bien organizada de las clases para el paso a la siguiente disciplina de la metodología seleccionada. El uso de estándares de codificación permitió proyectar una calidad en el código generado.

CAPÍTULO III: EVALUACIÓN DE LA PROPUESTA DE SOLUCIÓN

Introducción

En este capítulo se pretende evaluar el grado de calidad y fiabilidad de los resultados obtenidos en el desarrollo de este trabajo luego de la implementación de la herramienta informática. Se muestran los resultados obtenidos de la aplicación de las técnicas utilizadas para la validación del diseño y de la implementación. Además, se muestran los resultados de las pruebas de aceptación por parte del cliente para la verificación de la eficacia del sistema.

3.1 Validación del diseño

Las métricas de software constituyen los elementos que permiten evaluar la calidad de una determinada característica o artefacto que se genere en un proyecto de software. Para la validación del diseño de la solución propuesta, se estudiaron las diferentes métricas de diseño y sus características.
Con el objetivo de medir el nivel de relaciones entre las clases del sistema y la complejidad de cada clase por separado se seleccionaron dos métricas que permiten realizar mediciones de este tipo: Relaciones entre clases y Tamaño operacional de las clases. A continuación, se muestran los resultados de la aplicación de estas métricas.

3.1.1 Métricas de diseño

Una métrica permite medir de forma cuantitativa la calidad de los atributos internos del software y de esta forma permite al ingeniero evaluar la calidad del diseño durante el desarrollo del sistema. Las métricas se centran en cuantificar tanto la complejidad, como la funcionalidad y eficiencia inmersa en el desarrollo de software. Inclina sus objetivos a mejorar la comprensión de la calidad del producto, a estimar la efectividad del proceso y mejorar la calidad del trabajo (Pressman, 2010).

Métrica relaciones entre clases (RC):

Esta métrica está dada por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad:

- ✓ Acoplamiento: Un aumento de RC implica un aumento del Acoplamiento de la clase.
- ✓ Reutilización: Un aumento de RC implica una disminución en el grado de reutilización de la clase.

En la siguiente tabla se muestran las medidas utilizadas para evaluar cada uno de estos parámetros de calidad.

Tabla 5. Criterios de evaluación para la métrica RC.

Parámetros	Categoría	Criterios
Acoplamiento	Ninguno	0

	Bajo	1
	Medio	2
	Alto	>2
Reutilización	Bajo	>2*Prom.
	Medio	Entre Prom. y 2*Prom
	Alto	<=Prom.

A continuación, se muestran las figuras con los resultados obtenidos:

Figura 16. Representación de la incidencia de los resultados de la evaluación de la métrica RC en los atributos Acoplamiento y Reutilización.

Métrica tamaño operacional de las clases (TOC):

Consiste en medir el tamaño general de una clase tomando los siguientes valores:

- ✓ El total de operaciones (operaciones tanto heredadas como privadas de la instancia), que se encapsulan dentro de la clase.
- ✓ El número de atributos (atributos tanto heredados como privados de la instancia), encapsulados por la clase.

En la siguiente tabla se muestran las medidas utilizadas para evaluar cada uno de estos parámetros de calidad.

Tabla 6. Criterios de evaluación para la métrica TOC.

Parámetros	Categoría	Criterios
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom. y 2* Prom.
	Alta	> 2* Prom.

Complejidad implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Prom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom. y 2* Prom.
	Alta	<= Prom.

Si el resultado obtenido indica valores grandes, significa que la clase posee un alto grado de responsabilidad, debido a esto se reducirá la reutilización, se hará mucho más difícil la implementación y la realización de pruebas de dicha clase. Por tanto, mientras menor sea el valor para el TOC se hará mucho más fácil la reutilización de dicha clase dentro del sistema. Una vez aplicada dicha métrica a las clases de la solución, la misma arrojó los siguientes resultados:

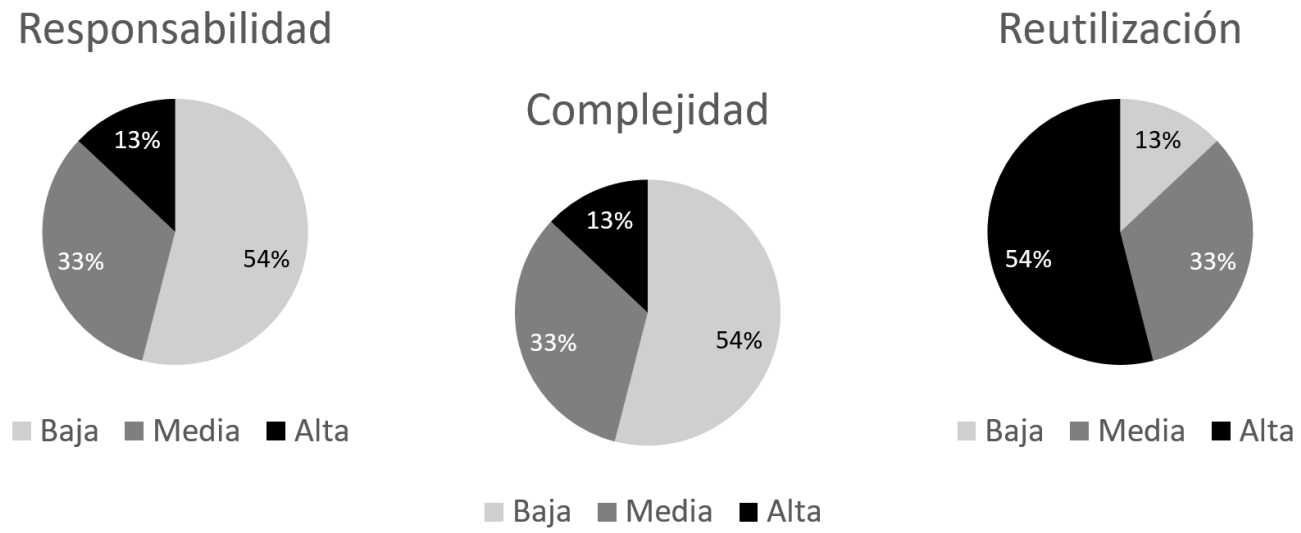

Figura 17. Representación de la incidencia de los resultados de la evaluación de la métrica TOC en los atributos responsabilidad, complejidad y reutilización.

Después de aplicadas las métricas seleccionadas se puede concluir que el diseño realizado tiene una buena calidad pues se aprecian resultados satisfactorios en los atributos de calidad medidos. Los datos obtenidos demuestran que las clases están concebidas de forma tal que se facilite su implementación, debido que presentan un acoplamiento bajo (68%). Se obtuvieron bajos niveles de responsabilidad (54%) y de complejidad de implementación (54%), evidenciándose la correcta asignación de responsabilidades a las clases involucradas en la solución. Además, se puede asegurar que el diseño de la solución tiene altos niveles de reutilización comprobado en los resultados arrojados por ambas métricas, (54%) en TOC y (68%) en RC, ya que se busca el poder reutilizar las clases en caso de ser necesario. Con los resultados anteriores queda validado el diseño de la herramienta a desarrollar, atendiendo a los parámetros definidos.

3.2 Pruebas de software

Las pruebas y validaciones de software son de vital importancia para corroborar que la aplicación desarrollada no presente problemas de ejecución y se encuentre libre de errores, ya que durante el proceso de desarrollo de software es frecuente que los desarrolladores, al programar las diferentes funcionalidades, obvien algunas posibilidades que pueden variar el resultado esperado durante la ejecución del código; por tanto, estas pruebas y validaciones constituyen la vía a través de la cual se asegura que el producto desarrollado está listo para ser entregado a los usuarios finales. Los errores más frecuentes pueden suceder a causa del mal uso de estructuras de datos, errores lógicos, entre otras (Pressman, 2010).

Teniendo en cuenta la metodología utilizada para el desarrollo de esta herramienta, se decide validar la propuesta de solución a través de las disciplinas pruebas internas y de aceptación.

3.2.1 Pruebas internas

En esta disciplina se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas (Rodríguez Sánchez, 2015).

Para la realización de estas pruebas se utilizó el método caja negra usando la técnica partición equivalente.

Método de caja negra

Las pruebas de caja negra, también denominada prueba de comportamiento, se centran en los requisitos funcionales del software, o sea, la prueba de caja negra permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Evalúa las funcionalidades del software (lo que hace) (Pressman, 2010).

Este método se aplicó haciendo uso de la técnica partición equivalente. A continuación, se describe los resultados de esta técnica.

Técnica partición equivalente:

La partición equivalente es un método de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores (por ejemplo, proceso incorrecto de todos los datos de caracter) que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada (Pressman, 2010).

En el **Anexo 1** se muestra la descripción de las variables para el requisito “Conceptualizar criterio”. Luego de definir el tipo de variables y las descripciones se procede a realizar el diseño de caso de pruebas para este requisito, en el **Anexo 2** se muestra el diseño de caso de pruebas para dicho requisito.
Durante la ejecución de estas pruebas se realizaron tres iteraciones, donde los errores detectados fueron solucionados conllevando a que en la 3ra iteración no se detectaron errores. Entre los errores detectados se encuentran: errores ortográficos, de validación, de interfaz. A continuación, se presentan los resultados obtenidos al aplicar esta prueba al producto obtenido.

Tabla 7. Resultado de las iteraciones al aplicar el método de caja negra.

Iteraciones	No conformidades
Iteración 1	19
Iteración 2	8
Iteración 3	0

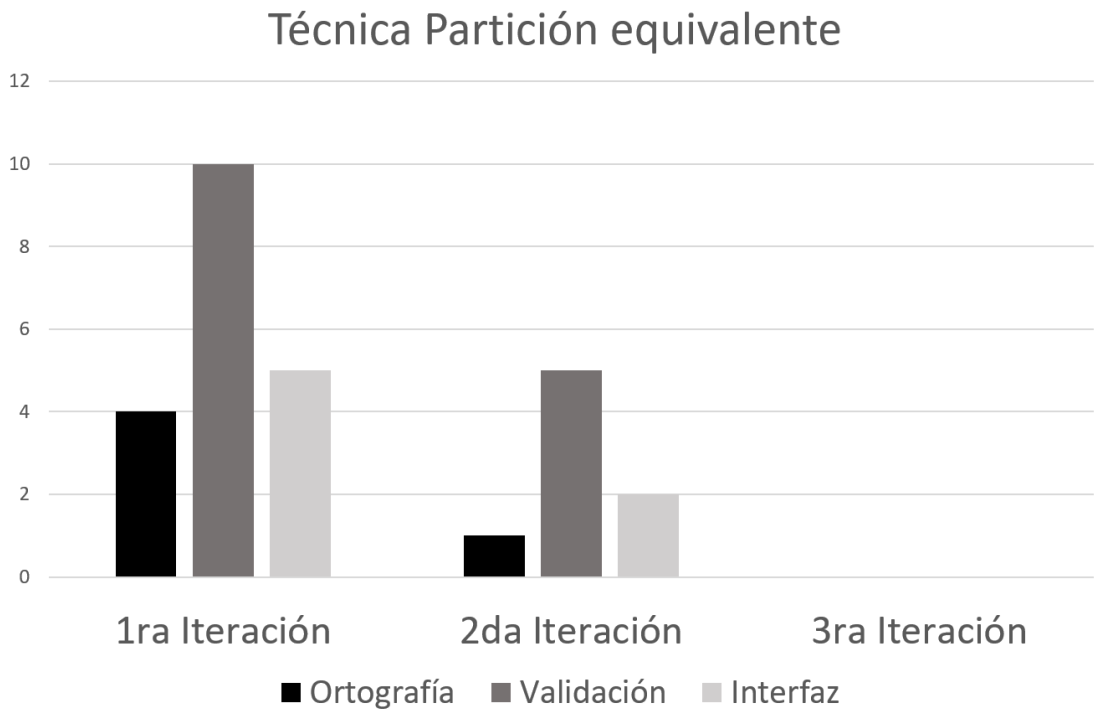

Figura 18. Resultados de la aplicación del método de caja negra.

3.2.2 Pruebas de aceptación

Es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido (Rodríguez Sánchez, 2015).

Para realizar estas pruebas se utilizó el mismo principio de las pruebas internas, permitiendo que sea el propio cliente el que realice las mismas. Para esto, se realizaron 2 iteraciones, en la 1ra se detectaron solo 4 no conformidades, 2 de Validación, 1 de Ortografía y 1 de Interfaz. Ya en la 2ª iteración estos errores fueron resueltos satisfaciendo las necesidades del cliente. Estos resultados quedan reflejados en la siguiente imagen:

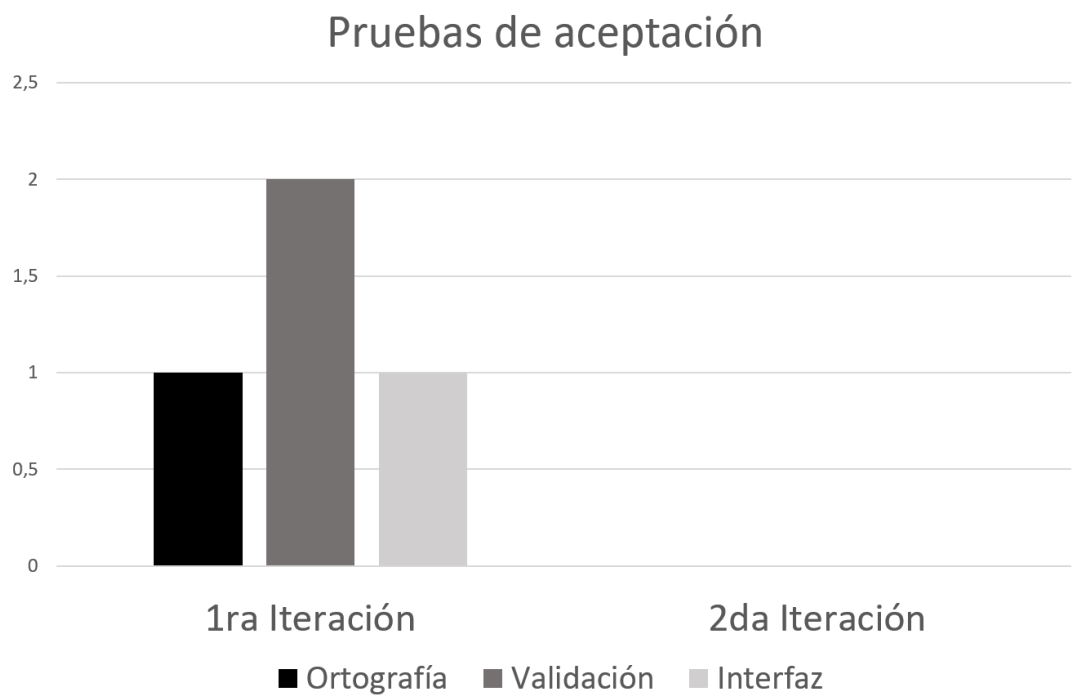

Figura 19. Resultados de la prueba de aceptación.

Una vez realizadas las pruebas de Aceptación, se generó el Acta de aceptación del producto como constancia de la conformidad por parte del cliente de la solución propuesta (ver **Anexo 3**).

3.3 Validación de la solución

Para la validación de la solución se consideraron los análisis de factibilidad de los proyectos de software que se describen a continuación:

Proyecto 1: Este proyecto tuvo como objetivo mejorar el servicio de despertador automático brindado por ETECSA en Cuba, para poder así reutilizar el personal subutilizado que tenían brindando este servicio, mejorando la eficiencia y productividad del trabajo además de introducir en el país la tecnología de *Asterisk* que es una de las tecnologías líderes en el mundo como solución de telefonía IP.

Proyecto 2: Este proyecto tuvo como objetivo informatizar el sistema penitenciario cubano, aprovechando el desarrollo de las tecnologías que se llevaban a cabo en el país, para apoyar los

procesos de trabajo que se ejecutan para el control, tratamiento y atención a los internos en los establecimientos penitenciarios y los correspondientes a los tres niveles de mando.

Proyecto 3: Tiene como objetivo informatizar completamente los procesos aduanales del país, aprovechando el desarrollo de las tecnologías que se están llevando a cabo en estos momentos como apoyo para el control de los mismos. Es una solución Web desarrollada sobre tecnologías libres.

Proyecto 4: Tiene como objetivo garantizar la comunicación entre la UCI y los graduados de la misma, para lo cual es esta versión se incorpora una nuevo diseño y arquitectura de la información que permite mejorar la gestión del conocimiento. Debe implementar un módulo para los cursos a distancia, comunidades virtuales, así como mensajería en línea web para mantener la comunidad de los graduados activa fuera del centro.

Proyecto 5: Esta aplicación tiene que brindar al cliente el valor monetario que este deberá abonar de forma mensual, como resultado de las llamadas telefónicas efectuadas a través de la pizarra a su empresa. Debe además registrar toda la información de las llamadas entrantes y salientes, así como brindar un grupo de reportes.

Para realizar el análisis de factibilidad de los proyectos se utilizó el MFac-PS teniendo en cuenta los criterios obtenidos a partir de la herramienta desarrollada en la presente investigación. Se consideraron para la selección de los mismos dos expertos obteniéndose un total de 6 criterios considerándose fundamentalmente las áreas técnica y comercial:

- grado de comercialización
- demanda actual del producto
- impacto entre los productos existentes
- hardware existente en la organización
- herramientas de software existentes
- soporte de software

Para analizar la factibilidad de los proyectos se considera la medida propuesta por (Peña 2018), que se describe a continuación considerando la siguiente ecuación:

$$Di = |Ri - Si|$$

Donde:

Di: Diferencia

Ri: Factibilidad real de los proyectos

Si: Factibilidad arrojada por el cálculo realizado teniendo en cuenta los criterios definidos por la propuesta de solución

Si $D <= 2$ se considera un grado aceptable de acierto entre lo real y lo arrojado por el método. Se puede variar el umbral en función de la opinión de los expertos (Peña Abreu, 2017).

Se determina el grado de aceptación de los proyectos con las siguientes medidas propuestas por (Peña Abreu, 2017):

- Casos positivos: Son aquellos donde los resultados de factibilidad del método aplicado y la realidad son similares con una $D <= 2$.
- Casos falsos positivos: Son aquellos donde el cálculo utilizando los criterios obtenidos por la herramienta arroja que el proyecto es factible y en la realidad tuvo menor grado de factibilidad o fue cancelado.
- Casos falsos negativos: Son aquellos donde el cálculo utilizando los criterios obtenidos por la herramienta arroja que el proyecto no es factible y en la realidad sí lo fue.

Análisis de los resultados

A continuación, se presenta la comparación de los resultados obtenidos al calcular la factibilidad teniendo en cuenta los criterios obtenidos por la propuesta de solución y la factibilidad real.

Tabla 8. Comparación entre lo arrojado por la herramienta y lo real.

Proyectos	Lugar teniendo en cuenta los criterios obtenidos	Lugar factibilidad real	Diferencia	Tipo de caso
Proyecto 1	4	4	0	Positivo
Proyecto 2	2	5	3	Falso positivo
Proyecto 3	1	1	0	Positivo
Proyecto 4	5	2	3	Falso negativo
Proyecto 5	3	3	0	Positivo

En el caso del Proyecto 2 arrojó falso positivo dado que este proyecto fue cancelado luego de haber comenzado su desarrollo, el fracaso del mismo está asociado a que no se hizo un análisis profundo del equipo de desarrollo que lo iba a ejecutar, al realizar los estudios previos del proyecto se contó con un equipo preparado en el negocio en el que se iba a desarrollar, posteriormente cuando comenzó la

ejecución el equipo que comenzó el desarrollo era generalmente nuevo y por tanto con poca experticia lo que trajo consigo que no fuesen capaces de realizar el desarrollo en el tiempo pactado. Esta experiencia indica que al realizar los estudios de factibilidad y evaluar los indicadores debe hacerse con sumo cuidado evaluando las condiciones reales de recursos disponibles e incluso previendo situaciones de riesgo, como es el caso.

En el caso del Proyecto 4 los resultados en la realidad fueron mucho mejores de lo que arrojó el sistema y esto está dado fundamentalmente porque la respuesta del sistema está basada en que económicamente este proyecto no era factible para la institución, sin embargo, había un interés social para la misma de que este fuese desarrollado porque los principales beneficiados son los graduados UCI. Es la organización quien finalmente decide cual proyecto ejecutar según sus objetivos estratégicos.

Tabla 9. Porcentaje de tipo de caso respecto a los resultados finales.

Tipo de caso	Porcentaje respecto a la cantidad de proyectos	Cantidad de proyectos
Positivos	60%	3
Falsos negativos	20%	1
Falsos positivos	20%	1

Al analizar los resultados expuestos en la tabla anterior, se puede concluir que los criterios obtenidos por la herramienta se pueden concluir que la solución contribuye positivamente en la predicción de la factibilidad, debido a que el resultado obtenido al evaluar una muestra de proyectos, tuvo una certeza del 60% de acierto en correspondencia con los resultados reales. Solo el 40% de estos proyectos fueron evaluados de forma distinta a consecuencia de ser cancelados por otros motivos ajenos al estudio de factibilidad y no porque estos hubieran sido evaluados erróneamente. El resultado final no se encuentra alejado de la realidad, sino que coincide con esta en la mayoría de los casos, por lo que se demuestra la contribución de la propuesta de solución en la predicción de la factibilidad de proyectos informáticos.

Conclusiones parciales

Al finalizar el presente capítulo se puede concluir que las métricas de diseño aplicadas permitieron medir de forma cuantitativa la calidad de los atributos internos del software y de esta forma permitió evaluar la calidad del diseño durante el desarrollo de la herramienta. Las pruebas de aceptación y de caja negra, permitieron comprobar el correcto funcionamiento de la herramienta luego de realizadas varias iteraciones. La validación de la solución demostró que la herramienta desarrollada contribuye positivamente en la predicción de la factibilidad de proyectos informáticos.

CONCLUSIONES GENERALES

El desarrollo del presente trabajo de diploma permitió que se les diera solución a los objetivos específicos, a fin de dar cumplimiento al objetivo general, logrando arribar a las siguientes conclusiones:

- La elaboración del marco teórico referencial de la investigación permitió realizar el estado del arte de la investigación, así como las herramientas, lenguajes y la metodología de desarrollo a emplear en la implementación.
- El estudio del estado del arte realizado, de herramientas informáticas para análisis de factibilidad, permitió al autor establecer un punto de partida de la posición actual de los estudios de factibilidad.
- Durante el modelado de negocio, análisis y diseño de la solución se obtuvieron los productos de trabajo propuestos por la metodología de desarrollo seleccionada, lo que facilitó la implementación de las funcionalidades definidas.
- Las pruebas realizadas permitieron garantizar un correcto funcionamiento de la herramienta y el cumplimiento de las necesidades y requisitos del cliente, avalado mediante la carta de aceptación.
- La herramienta informática obtenida contribuye positivamente en la predicción de la factibilidad de proyectos informáticos.

RECOMENDACIONES

Luego de la experiencia adquirida durante la realización de la investigación, se recomienda:

- Incorporar a la aplicación el servicio LDAP de la Universidad de las Ciencias Informáticas para que la aplicación pueda ser utilizada en los centros productivos.
- Realizar el paginado de las tablas para controlar la estadística de la información visualizada.

REFERENCIAS BIBLIOGRÁFICAS

1. **Alvarez, Sara. 2007.** Sistemas gestores de bases de datos. *Desarrollo Web.* [En línea] 31 de Julio de 2007. https://desarrolloweb.com/articulos/sistemas-gestores-bases-datos.html.
2. **Apache. 2014.** Ibrugor. [En línea] 2014. http://www.ibrugor.com/blog/apache-http-serverque-es-como-funciona-y-para-que-sirve/.
3. **Basalo, Alberto y Álvarez, Miguel Angel. 2014.** DesarrolloWeb.com. [En línea] 2014. http://www.desarrolloweb.com/articulos/que-es-angularjs-descripcion-framework-javascript-conceptos.html.
4. **Cáceres Navarro, Analaicy . 2012.** *Guía para la estimación de la factibilidad económica de la migración a Software Libre y de Código Abierto en PYMES.* 2012.
5. **Cervantes, Humberto. 2018.** Software Guru. [En línea] 2018. http://sg.com.mx/revista/27/arquitectura-software.
6. **Choise, Expert. 2015.** IOSA: Investigación de Operaciones SA. [En línea] 2015. http://www.iosa.com.pe/productos/expert-choice.
7. **Córdoba Padilla, Marcial. 2011.** *Formulación y evaluación de proyectos.* 2011.
8. **Cruz, M y Campano, A. 2008.** *El procesamiento de la información en investigaciones educacionales.* La Habana : s.n., 2008.
9. **de Castro García, Noemí. 2013.** *Números realistas Vs. Dos-tuplas.* s.l. : Universidad de León, 2013.
10. **DECIDE. 2009.** DECIDE - Software para Formulación y Evaluación de Proyectos de Inversión. [En línea] 2009. http://www.decide.com.mx/decide.htm.
11. **—. 2013.** ITESO. [En línea] 2013. http://www.iteso.mx/web/general/detalle?group_id=57482..
12. **Denning, P. J. 2012.** Closing Statement: What Have We Said About Computation? s.l. : The Computer Journal, 2012. Vol. 77, 7, págs. 863-865.
13. **EasyPlanex. 2014.** BoraSystem. [En línea] 2014. http://www.borasystems.com/es/productossoftware/easyplanex/.
14. **Eguiluz, J. 2011.** *Desarrollo web ágil con Symfony 2-Primera edición.* 2011.
15. **Eguiluz, Javier. 2012.** *Desarrollo web ágil con Symfony2.* 2012.

16. **Escalona, María José y Koch, Nora. 2002.** *Ingeniería de Requisitos en Aplicaciones para la Web. Un estudio comparativo.* España : s.n., 2002.

17. **Fernández Díaz, M. 2012.** Serie Científica de la Universidad de Ciencias Informáticas. [En línea] 2012. http://publicaciones.uci.cu/index.php/SC/article/viewFile/1049/603.

18. **Ferrin González, Raciel. 2016.** *Migración de los módulos Persona y Combustible del Sistema Orbita a la arquitectura de referencia en PHP Bosón.* La Habana : Universidad de las Ciencias Informáticas, 2016.

19. **Figueroa, Roberth G., Solís, Camilo J y Cabrera, Armando A. 2010.** *Metodologías tradicionales vs. metodologías ágiles.* 2010.

20. **Gamma, Erich, y otros.** *Design Patterns: Elements of Reusable Object -Oriented Software.*

21. **García García, Yadira y Álvarez Sosa, Alejandro David. 2015.** *Herramienta para evaluar la factibilidad técnica y comercial de proyectos de software mediante la computación con palabras.* La Habana : UCI, 2015.

22. **García Valdés, Margarita y Suárez Marín, Mario. 2013.** *El método Delphi para la consulta a expertos en la investigación científica.* La Habana : Revista Cubana de Salud Pública, 2013.

23. **García-Morales, Elisa. 2002.** SEDIC | Sociedad Española de Documentación e Información Científica. [En línea] 2002. http://www.sedic.es/autoformacion/seccion6_DProcesos.htm.

24. **Giraldo, G., y otros. 2011.** Revista Avances en Sistemas e Informática. [En línea] 2011. http://www.redalyc.org/articulo.oa?id=133122679013.. ISSN: 1657-7663..

25. **GitHub. 2010.** GitHub: Popular Watched Repositories. [En línea] 2010. https://web.archive.org/web/20100419140506/http://github.com/popular/watched.

26. **Helmer, O y Rescher, N. 1959.** *On the Epistemology of the Inexact Sciences.* California : s.n., 1959.

27. **Hernández, Rolando Alfredo. 2002.** *Gestión de Proyectos para Informáticos.* La Habana : Universidad de las Ciencias Informáticas, 2002.

28. **IEEE. 2004.** *Guía al cuerpo de conocimiento de la Ingeniería de Software.* 2004.

29. **Jones, J y Hunter, D. 1995.** *Qualitative Research: Consensus methods for medical and health services research. BMJ.* 1995. págs. 311-376.

30. **JQuery. 2018.** JQuery foundation. [En línea] 2018. https://jquery.org/license/.

31. **Kendall, Kenneth E. y Kendall, Julie E. 2005.** *Análisis y diseño de sistemas. Sexta Edición. [Trad.] Antonio Núfiez Ramos.* México : s.n., 2005. ISBN: 970-26-0577-6.

32. **Larman, Craig. 2003.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Madrid, España. : PEARSON EDUCACIÓN, S.A., 2003. ISBN: 84-205-3438-2.

33. **LIBROSWEB. 2010.** LIBROSWEB.es. [En línea] 2010. http://librosweb.es/symfony/capitulo_8/por_que_utilizar_un_orm_y_una_capa_de_abstraccion.html.

34. **Llorens Fábregas , Juan. 2005.** *Gerencia de proyectos de tecnología de información.* 2005.

35. **Martínez León, Nelson Enrique, Gómez Flórez, Luis Carlos y Pimentel Ravelo, Jorge Iván. 2011.** *herramienta computacional para la gestión y evaluación de proyectos software enmarcados en actividades de investigación.* s.l. : Scientia et Technica, 2011.

36. **Mateu, Carles. 2004.** *Desarrollo de aplicaciones web.* 2004.

37. **MDN. 2018.** MDN web docs. [En línea] 2018. https://developer.mozilla.org/es/docs/Web/JavaScript.

38. **MINJUS. 2015.** *Resolución 327.* 2015.

39. **NetBeans. 2015.** NetBeans. [En línea] 2015. https://netbeans.org/index_es.html.

40. **Oktaba, Hanna. 2002.** *Introducción a patrones.* Facultad de Ciencias, Universidad Nacional Autónoma de México. 2002.

41. **Olazabal Arteaga, René. 2014.** *Módulo de gestión de trazas para el marco de trabajo Symfony 2.* La Habana : Universidad de las Ciencias Informáticas, 2014.

42. **Paradigm, Visual. 2015.** Knowledge Reuse Group. [En línea] 2015. http://www.ie.inf.uc3m.es/grupo/docencia/reglada/Is1y2/PracticaVP.pdf..

43. **Patrones. 2010.** Patrones de diseño. [En línea] 2010. https://patronesdediseno.net16.net.

44. **Peña Abreu, Marieta. 2017.** *Modelo para el análisis de factibilidad de proyectos de software en entornos de incertidumbre.* La Habana : s.n., 2017.

45. **PHP. 2015.** PHP. [En línea] 2015. http://www.php.net/manual/es/intro-whatis.php.

46. **Piñeiro Pérez, P. Y. y colectivo de autores. 2013.** *GESPRO Paquete para la gestión de proyectos.* La Habana : Nueva Empresa, 2013. ISSN: 1682-2455..

47. **PostgreSQL. 2018.** PostgreSQL [OFICIAL SITE]. [En línea] 2018. https://www.postgresql.org/about/.

48. **Pressman, Roger S. 2010.** *Ingeniería de Software un enfoque Práctico.* New York : s.n., 2010.

49. **RAE. 2018.** Diccionario de la lengua española. Real Academia Española. [En línea] 2018. http://dle.rae.es/?id=LDxc1BE.

50. **Recaman, Hernando Chaux. 2012.** *Marco de trabajo para aplicaciones web de código abierto en instituciones universitarias.* 2012.

51. **Reynoso, C. A. 2004.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004.

52. **Rodríguez Sánchez, Tamara. 2015.** *Metodología de desarrollo para la Actividad Productiva de la UCI.* La Habana : UCI, 2015.

53. **Rouse, M. 2007.** *What is integrate development environment (IDE).* 2007.

54. **Rumbaugh, James, Jacobson, Ivar y Booch, Grady. 2000.** *El Lenguaje Unificado de Modelado. Manual de referencia. [trad.] Addison Wesley Longman.* Madrid, España : s.n., 2000. ISBN: 84-7829-037-0.

55. **Serra, Martins y Kunc, Martin. 2015.** *Benefits Realisation Management and its influence on project success and on the execution of business strategies.* s.l. : International Journal of Project Management, 2015. págs. 53-66.

56. **SINNAPS. 2018.** SINNAPS, gestor de proyectos. [En línea] 2018. https://www.sinnaps.com/blog-gestion-proyectos/que-es-un-diagrama-de-proceso.

57. **Sommerville, Ian. 2007.** *Ingeniería del Software. Octava edición.* Madrid, España. : s.n., 2007.

58. **SparxSystems. 2018.** SparxSystems. [En línea] 2018. http://www.sparxsystems.com.ar.

59. **Team, Doctrine Project. 2011.** Doctrine Project Team. [En línea] 2011. http://www.lawebdelprogramador.com/cursos/PHP/7033-Doctrine_2_ORM_Documentation.html.

60. **Villagrán Andrade, María José. 2015.** *Estudio de factibilidad para la implementación de un centro de interpretación del patrimonio cultural y natural de Yaharcocha, cantón Ibarra, provincia de Imbabura.* s.l. : Escuela Superior Politécnica de Chimborazo, 2015.

61. **W3C. 2016.** W3C. [En línea] 2016. https://www.w3.org/standards/webdesign/htmlcss.

62. **W3Techs. 2018.** W3Techs. [En línea] 2018. https://w3techs.com/technologies/overview/javascript_library/all.