TRABAJO DE DIPLOMA

PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

“SoftQuality, sistema informático de ayuda a la toma de decisiones para la evaluación y selección de productos de software”

Autor: Ramón Guillermo Álvarez Hernández

Tutores: Dra. Yamilis Fernández Pérez

Ing. Naivy Pujol Méndez

Año 60 de la Revolución

La Habana, julio de 2018

DECLARACIÓN DE AUTORÍA

Declaro ser único autor de la presente tesis y concedo a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ___ días del mes de _____ del año _____.

_____

Ramón Guillermo Álvarez Hernández

(Autor)

_____ _____

MSc. Yamilis Fernández Pérez

(Tutor)

Ing. Naivy Pujol Méndez

(Tutor)

AGRADECIMIENTOS

Agradezco a mis padres, al resto de mi familia, a mis amigos y a todas las personas que de una forma u otra me ayudaron durante la carrera.

DEDICATORIA

Dedico este trabajo a mis padres.

Resumen

La calidad del software es una preocupación a la que se dedican muchos esfuerzos. Sin embargo, el software casi nunca es perfecto. Todo proyecto tiene como objetivo producir software de la mejor calidad posible, que cumpla, y si puede supere las expectativas de los usuarios. La presente investigación tiene como precedente la necesidad de CALISOFT de realizar las evaluaciones de calidad de los productos de software usando el modelo Evalprod de manera informatizada, ya que la aplicación manual del mismo es engorrosa, compleja y extensa. Para ello es esencial el uso de un sistema automatizado que guie el proceso de evaluación y apoye en la toma de decisiones. En la investigación se obtiene una aplicación informática que implementa el modelo para la evaluación Evalprod. El sistema se desarrolló a partir de tecnologías libres, con XP como metodología de desarrollo. Para la validación del software se usaron las pruebas unitarias y de aceptación.

Palabras clave: Evaluación de la calidad de software

Índice de Contenido

Introducción 1

CAPÍTULO 1. Fundamentación teórica 6

Introducción 6

1.1 Soft Computing 6

1.2 Calidad de *Software* 9

1.3 Modelos de calidad y estándares 10

1.4 Evaluación y selección de productos de *software*. 12

1.5 Herramientas informáticas para evaluar la calidad de productos de *software* que usan modelos de calidad. 14

1.5.1 Modelo de selección y evaluación de productos de *software* (Evalprod). 16

1.5.2 Descripción de los componentes del modelo 17

1.6 Herramientas, Metodologías y Lenguajes de Desarrollo 20

Capítulo 2. Análisis y diseño de la aplicación 26

Introducción 26

2.1 Recopilación de información para el sistema a desarrollar 26

2.2 Características del sistema 26

2.2.1 Requisitos del sistema 27

2.2.2 Requisitos no funcionales 29

2.3 Fase de Planeación 30

2.3.1 Historias de usuarios 30

2.3.2 Plan de entrega 33

2.3.3 Plan de iteraciones 33

2.4 Fase de Diseño 34

2.4.1 Tarjetas CRC 35

2.5 Arquitectura 35

2.5.1 Patrón de arquitectura 36

2.6 Patrones de diseño 37

2.6.1 Patrones GRASP (Patrones de *Software* para la Asignación General de Responsabilidad) 37

2.6.2 Otros patrones importantes 39

Capítulo 3 Implementación y pruebas al sistema 40

Introducción 40

3.1 Tareas de Ingeniería 40

3.2 Estilos de programación 42

3.2.1 Pruebas de *software* 43

3.2.2 Pruebas Unitarias 43

3.2.3 Pruebas de Aceptación 44

Conclusiones generales 48

Recomendaciones 49

Referencias bibliográficas 50

Introducción

El *software*, se ha convertido en un producto vital, tanto para empresas, organismos, servicios y tareas cotidianas de los ciudadanos como para la toma de decisiones, el intercambio de información y la gestión del conocimiento. Su utilización, promueve el aumento de la producción, amplía las posibilidades de negociación, supervisa y controla activamente a los pacientes de manera no invasiva y facilita la comunicación al disminuir las barreras geográficas. Se está convirtiendo, cada vez más, en una tecnología penetrante, omnipresente y habilitadora proporcionando a los seres humanos una serie de servicios y aplicaciones inteligentes sin precedentes (Pérez, 2018).

Esta situación induce al incremento de la complejidad del *software*, surgiendo un ambiente de competencia y especialización entre las entidades productoras y comercializadoras, lo que provoca un especial interés por la calidad del mismo. La calidad de los productos de *software* se ha convertido en uno de los elementos diferenciadores entre las diferentes compañías a nivel mundial por las ventajas competitivas que puede aportar (Pérez, 2018).

La búsqueda de la calidad de los sistemas, ha propiciado la creación de modelos, estándares y metodologías para evaluarla, asegurarla y controlarla (Pérez, 2018). Por tanto, resulta pertinente el desarrollo de herramientas informáticas para evaluar la calidad y seleccionar productos de *software*, aspecto fundamental de este trabajo de diploma.

Según la NC ISO/IEC 25000, la calidad del *software* es el grado en que este satisface las necesidades expresadas o implícitas, cuando se usa bajo condiciones determinadas (Oficina Nacional de Normalización(NC),“NC ISO/IEC 25000, 2011). Pero el término calidad de producto de *software* es ambiguo dado su carácter intangible y eso dificulta el proceso de evaluación, en el que intervienen diversos factores.

La evaluación es un proceso costoso y complejo por la cantidad de recursos involucrados, resultando de vital importancia para la toma de decisiones, ya que una evaluación equivoca puede provocar cuantiosos gastos monetarios innecesarios a la entidad donde se implanta o incluso la pérdida de vidas humanas, según el dominio de aplicación. De ahí la relevancia de lograr una adecuada correlación entre los resultados obtenidos al evaluar un producto *software* con la calidad que esta muestra en la realidad (Pérez, 2018).

Resulta primordial también, lograr que la evaluación de la calidad de un *software* se traduzca en valores que puedan ser comparados e integrados con criterios de costo y beneficio.

La evaluación de la calidad del producto de *software*, trae disimiles beneficios como controlar y mejorar las características del producto, asegurar a los clientes un nivel de calidad, comparar con productos de la competencia, posicionar sus productos en el mercado, conocer la calidad del producto que compra, minimizar los fallos en producción, entre otros (Pérez, 2018).

A pesar de la existencia de modelos, metodologías y métodos de evaluación de *software* basados en *Soft Computing*, enfocados a considerar el tratamiento de la incertidumbre, persisten deficiencias en los mismos, tales como la incapacidad para ofrecer un marco conceptual capaz de manipular información heterogénea. No existe tampoco una normalización de la escala de valoración; el tratamiento de las relaciones y restricciones entre las características y medidas de calidad, es prácticamente inexistente y, por último, se dificulta el manejo de la información de los usuarios finales al no tener en cuenta que estos pueden proceder de diferentes áreas de conocimiento, poseer diversa formación y dominio de los atributos y necesitar, por tanto, escalas con desigual granularidad (Pérez, 2018). Todo esto provoca pérdida de información, limita el proceso de toma de decisiones y provoca insatisfacción en los clientes de la evaluación.

Cuba crea en agosto de 2012 el Centro Nacional de Calidad del *Software* (CALISOFT), que tiene como principales objetivos ofrecer servicios de evaluación de la calidad de los procesos de desarrollo de *software*, los procesos de prestación de servicios de tecnología de la información y los productos informáticos. Además, tiene la tarea de establecer estándares técnicos y procedimientos que normalicen la industria informática, dirigiéndola hacia niveles de calidad superiores. Esta empresa brinda, también, servicios de consultorías asociadas a la ingeniería y calidad de las aplicaciones informáticas y servicios de formación en los temas de calidad e ingeniería (Pérez, 2018).

Hay que mencionar que, a pesar del esfuerzo desarrollado en la Industria cubana del *software*, por velar por la calidad de los productos de *software*, existe un nivel bajo de implementación del proceso de evaluación del producto, evidenciando la no existencia de sistemas que implementen métodos para agregar la información heterogénea e interdependiente que se genera en este proceso. Unido a eso, la poca flexibilidad de las

plataformas existentes al adicionar nuevos métodos de agregación de información para la evaluación de la calidad de *software* y comparar los resultados de los mismos.

Ante la problemática planteada, conjuntamente la UCI y CALISOFT han desarrollado un modelo para la evaluación y selección de productos de *software*, según su calidad. Este modelo palía las dificultades halladas y citadas anteriormente, ya que incorpora elementos como son: 1) la manipulación de información heterogénea, ambigua, imprecisa, y de diferentes fuentes; 2) el análisis de la interdependencia entre los criterios presentes en el modelo. Actualmente este modelo se ha generalizado a otras entidades como XETID.

Para garantizar la aplicabilidad del modelo al ser este proceso tan complejo, se hace necesaria la implementación de una herramienta informática que guíe el proceso de evaluación y ayude en la toma de decisión, basado en el modelo propuesto. La ejecución del modelo de manera manual, es engorroso debido a su complejidad, principalmente cuando se hace extensivo a centros nacionales como CALISOFT o aumenta la cantidad de elementos a evaluar. Además, es importante comparar los resultados con otros métodos de agregación que permita dar una visión general de la evaluación.

Desde el punto de vista de evaluación como servicio, es importante la satisfacción de los clientes, entendidos no solo como los usuarios finales, sino también los desarrolladores o quienes desean diseñar nuevas versiones. Una herramienta informática de este tipo permitiría agilizar la evaluación, al poder medir la calidad del producto con precisión, prontitud e interpretar fácilmente los resultados, elementos esenciales para la satisfacción del usuario (Pérez, 2018).

Por todo lo antes descrito se plantea como **problema de investigación**: ¿Cómo facilitar la aplicación del nuevo modelo de evaluación y selección de productos de *software* Evalprod?

Se define como **objeto de estudio** el modelo de evaluación y selección de productos de *software*, enmarcado en el **campo de acción**: la informatización del modelo de evaluación y selección de productos de *software* desarrollado.

Para solucionar el problema de la investigación se identifica como **objetivo general**: Desarrollar una herramienta informática que facilite la aplicación del modelo de evaluación y selección de productos de *software*, del cual se derivan los siguientes **objetivos específicos**:

- Construir el marco teórico conceptual relacionado con el análisis de los elementos fundamentales que sustentan la investigación: Soft Computing, calidad de *software* y modelo de evaluación y selección de productos de *software*.
- Seleccionar las herramientas necesarias para realizar el diseño e implementación del sistema.
- Implementar la solución propuesta a partir del modelo EvalProd.
- Validar la propuesta, a través de los métodos definidos en la investigación.

La investigación está orientada de manera general por los siguientes **métodos científicos:**

Histórico-Lógico: Fue utilizado para el estudio crítico de los trabajos anteriores, estado del arte y para fundamentar la investigación. Es necesario realizar un análisis desde su origen hasta la fecha de los modelos y estándares de calidad incluyendo la gestión de métricas de calidad para un producto de *software*, así como, conocer la evolución y nuevas propuestas surgidas en el período de la investigación.

Analítico-Sintético: Utilizado para estudiar, analizar los trabajos, tendencias similares, que permitieron agrupar los detalles y elementos significativos a tener en cuenta para enriquecer el presente trabajo con un valor agregado, como es el caso de la evaluación de las características de calidad a partir del cálculo de métricas.

Hipotético-Deductivo: Permite la elaboración de la hipótesis de la investigación, así como definir nuevas líneas de trabajo a partir de los resultados parciales. Con los conocimientos adquiridos se logra definir una herramienta para el cálculo de las métricas, le permitirá conocer el grado de implementación de las características de calidad.

Modelación: Permite la representación explícita de la solución propuesta a través de la modelación de los procesos, las ideas y referentes teóricos extraídos de las fuentes bibliográficas consultadas. En este trabajo se utilizó en el análisis, diseño e implementación de la aplicación.

Análisis documental: Se emplea en la consulta de la literatura especializada en las temáticas afines de la investigación.

Estructura del trabajo

Capítulo 1 Fundamentación Teórica

Introducción

En este capítulo se abordan los conceptos principales relacionados con la investigación para el desarrollo de la solución al problema planteado. Se hace un análisis del estado de arte de los modelos de evaluación de la calidad y los sistemas informáticos que los implementan. Además, se explican la metodología y herramientas utilizadas para dar solución a la problemática planteada.

Capítulo 2 Análisis y diseño de la aplicación

Descripción de la solución propuesta al problema de investigación. Se explica el funcionamiento del sistema a desarrollar, así como se definen los principales elementos de la documentación de mayor relevancia tras la aplicación de la metodología de desarrollo.

Capítulo 3 Implementación y validación de la aplicación

Se implementa la solución propuesta, posteriormente se realizan las pruebas para validar el código y detectar posibles errores. También se le realizarán pruebas de aceptación con los clientes del producto.

CAPÍTULO 1. Fundamentación teórica

Introducción

El presente capítulo tiene como objetivo presentar, analizar y detallar los elementos teóricos que sustentan la presente investigación. Se realiza un estudio de los modelos de calidad de *software*, del proceso de evaluación de la calidad de productos existentes, así como los elementos a usar para agregar la información en el proceso de evaluación de la calidad, conformando de esta manera el estudio del estado del arte, definiéndose una solución propia por parte del usuario. Se definió además la metodología de desarrollo de *software*, necesaria para guiar el desarrollo de la herramienta a confeccionar, incluyéndose las tecnologías y herramientas de *software* que serán empleadas para dar solución al problema planteado. Se realiza un análisis de diferentes productos de *software* que apoyan el proceso de evaluación de la calidad.

1.1 Soft Computing

Básicamente, Soft Computing no es un cuerpo homogéneo de conceptos y técnicas, más bien es una mezcla de distintos métodos que de una forma u otra cooperan desde sus fundamentos. En este sentido, el principal objetivo de la Soft Computing es aprovechar la tolerancia que conllevan la imprecisión y la incertidumbre, para conseguir manejabilidad, robustez y soluciones de bajo costo (González, 2008).

Entre los componentes principales de la Soft Computing, se encuentran el razonamiento probabilístico, la lógica y conjuntos difusos, redes neuronales y metaheurísticas. No debe entenderse la Soft Computing como la suma de todos estos elementos, sino como el resultado de la integración, cooperación, asociación o la hibridación de sus componentes (González, 2008).

Un conjunto difuso, permite la pertenencia de un elemento a un conjunto de forma gradual y no absoluta, como establece la teoría clásica de conjuntos; es decir, admitiendo pertenencias valoradas en el intervalo [0,1] en lugar de en el conjunto {0,1}. Las aplicaciones basadas en dicha teoría, han evolucionado de tal modo que resulta imposible calcular el volumen de negocio que generan en todo el mundo, debido a su amplio uso en diferentes disciplinas y áreas de la ciencia (González, 2008).

La lógica difusa surge para la formalización del razonamiento aproximado; intenta manejar el conocimiento propio del “sentido común” (alto, bajo, poco, mucho, caro, etc.). Es una extensión de los principios de los conjuntos tradicionales, donde los elementos pueden

pertenecer o no a estos, y en su lugar, se asigna un determinado grado de pertenencia de los elementos al conjunto (Fernández Pérez, 2018).

Existen funciones de pertenencia típicas, una de las más usadas en la solución de problemas de calidad de *software* es la triangular (González, 2008):

La función triangular es definida por sus límites inferior *a* y superior *b*, y el valor modal *m*, tal que $a < m < b$ (ver figura 1).

$$A(x) = \begin{cases} 0 & si\ x \leq a \\ \frac{(x-a)}{(m-a)} & si\ a < x \leq m \\ \frac{(b-x)}{(b-m)} & si\ m < x < b \\ 0 & si\ x \geq b \end{cases}$$

Figura 1: Función triangular

Variables lingüísticas

A menudo, en aplicaciones, pretende describirse el estado de un objeto o fenómeno, cuya representación, para que sea útil y apropiada, debe formularse a través de palabras o sentencias, no mediante números. Este es el caso de las variables lingüísticas, cuyo valor establece la descripción.

Una variable lingüística, admite que sus valores sean etiquetas; o sea, términos lingüísticos definidos como conjuntos difusos (sobre cierto dominio subyacente). Una etiqueta, incluye muchos valores posibles. Dichas variables son de utilidad, al constituir una manera de comprimir información; además, ayudan a caracterizar fenómenos que pudieran estar mal definidos o resultar complejos de definir, o ambas cosas. Las mismas son un medio de trasladar conceptos o descripciones lingüísticas a numéricas y tratarlas automáticamente (González, 2008).

Una variable lingüística se define como un conjunto de 5 elementos $(N,U,T(N),G,M)$ [128], donde:

- *N* es el nombre de la variable
- *U* es el dominio subyacente.
- *T(U)* es el conjunto de términos o etiquetas que puede tomar *N.*
- *G* es una gramática para generar las etiquetas de *T(U)*
- *M* es una regla semántica que asocia cada elemento de *T(U)* con un conjunto difuso en *U* de entre todos los posibles: $M:T(\mathrm{U})\otimes F(U)$.

Un ejemplo es la variable evaluación (ver figura 2).

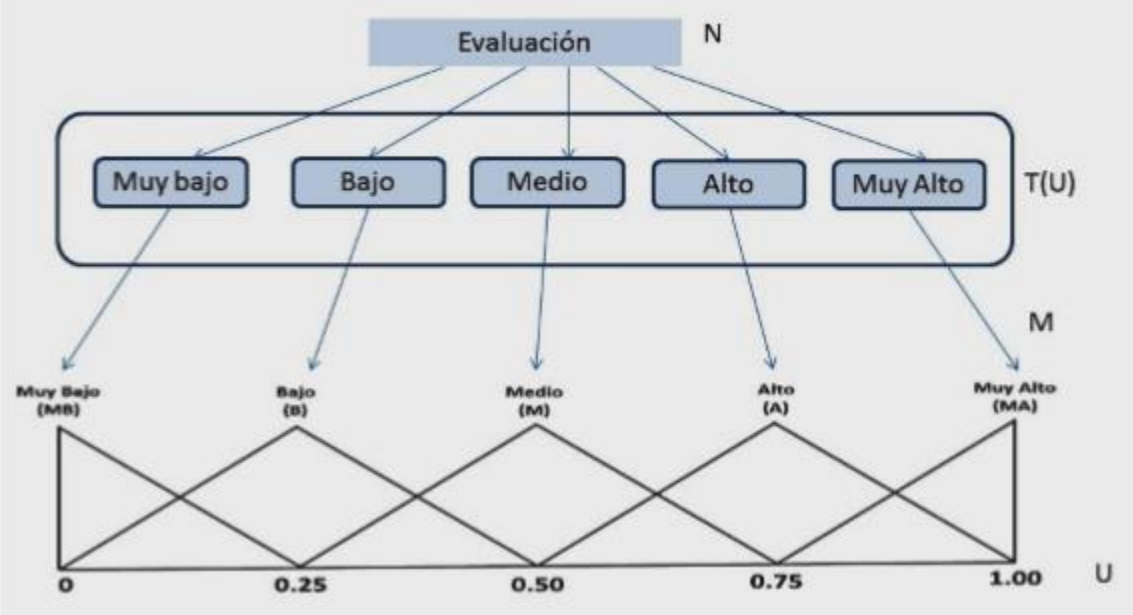


Figura 2: Variable Evaluación

Mapas Cognitivos Difusos

Los Mapas Cognitivos Difusos (MCD), son una técnica desarrollada por Kosko, en 1986, como extensión de los mapas cognitivos introducidos por Axerold, en 1976. Son una herramienta de modelado cualitativa, basada en los conocimientos y experiencia de los expertos.

Kosko, describe un MCD como un grafo difuso dirigido, donde el *feedback* es permitido. Los nodos, representan los conceptos y los arcos pesados, las relaciones causales entre conceptos. En los MCD, existen tres posibles tipos de relaciones entre conceptos: relación positiva, relación negativa o la no existencia de relaciones. El grado de la relación, se describe a través de un número difuso o valor lingüístico. El enlace causal es dinámico,

donde el efecto de un cambio en un concepto o nodo, afecta a los demás nodos (Fernández Pérez, 2018).

En este epígrafe se ha analizado un grupo de herramientas metodológicas que ayudan a solucionar las insuficiencias y retos presentes en el proceso de evaluación de la calidad de los productos de *software* y son la base del modelo Evalprod.

1.2 Calidad de *Software*

Según la Real Academia Española, la calidad es la "propiedad o conjunto de propiedades inherentes a algo, que permiten juzgar su valor". Esta definición está orientada al mercado.

La organización internacional de estándares (ISO/IEC 25000), dentro del contexto de Ingeniería de *Software*, define la calidad de *software* como: "*es el grado en que este satisface las necesidades expresadas o implícitas, cuando se usa bajo condiciones determinadas*" (Oficina Nacional de Normalización(NC),"NC ISO/IEC 25000, 2011).

Se puede decir que el *software* tiene calidad si cumple o excede las expectativas del usuario en cuanto:

1. Funcionalidad (que sirva un propósito)
2. Ejecución (que sea práctico)
3. Confiabilidad (que haga lo que debe)
4. Disponibilidad (que funcione bajo cualquier circunstancia)

Resumiendo, se puede decir, que la calidad de *software* se refiere a: Los factores de un producto de *software* que contribuyen a la satisfacción total de las necesidades de un usuario u organización (Oficina Nacional de Normalización(NC),"NC ISO/IEC 25000, 2011).

El *software* al ser un producto intangible se dificulta determinar los criterios para valorar su calidad. Es difícil concretar las necesidades implícitas y explícitas de los usuarios y traducirlas en indicadores que permita medir la calidad. Por eso han surgido varios modelos y estándares de calidad que organizan jerárquicamente criterios que permiten concretar la calidad de *Software*. A continuación, detallaremos los fundamentales y más usados (Pérez, 2018).

1.3 Modelos de calidad y estándares

Los modelos de calidad presentan un acercamiento para unir diferentes atributos de calidad con los objetivos básicos de: ayudar a entender como varias facetas de calidad contribuyen al todo y enfatizar claramente que la calidad de *software* es mucho más que simplemente defectos y fracasos. Además, los modelos representan una ayuda para navegar por el mapa de características de calidad, subcaracterísticas y medidas apropiadas y, por último, ayudan a definir el perfil de evaluación (Pérez, 2018).

Los modelos de calidad de los productos de *software* más significativos son: McCall (J. A. McCall and J. P. Cavano, 1978), Boehm (B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. McLeod, and M. Merritt, 1978), FURPS (R. B. Grady and D. L. Caswell, 1987), Dromey (Dromey, 1995), ISO 9126 e ISO 25010 (Oficina Nacional de Normalización(NC),"NC ISO/IEC 25000, 2011). Cada uno de ellos fue introducido con el propósito de definir la calidad de una nueva forma, cubriendo todas las perspectivas con diferentes puntos de vista para la evaluación del producto. Dichos modelos, conocidos como básicos, son genéricos y aplicados para cualquier tipo de *software*. Sirven como base para los modelos adaptados o modificados, que se personalizan o especifican para una aplicación y sector u área en particular (Scalone, 2006).

Estos modelos de calidad, conforman una estructura jerárquica, que puede representarse en forma de árbol *n-ario*. En el nivel más alto de la jerarquía (nivel 0) se encuentra la calidad, que se puede definir a través de una gran variedad de conceptos. Estos se pueden descomponer en sub-conceptos, cada uno de ellos derivados en atributos de calidad que se evalúan a través de un conjunto de medidas. Dichos niveles se nombran según el modelo.

La ventaja de los modelos de calidad, es que esta se convierte en algo concreto, que se puede definir, medir y, sobre todo, planificar. Es posible determinar la calidad de manera simple y coherente. Los modelos ayudan a comprender las relaciones existentes entre las diferentes características de un producto de *software*. Además, brindan criterios e indicadores para la medición de sus fortalezas y debilidades, con el propósito de introducir mejoras. Una desventaja es que aún no ha sido demostrada la validez absoluta de ninguno de ellos. Las relaciones establecidas entre las diferentes características, atributos y métricas, se derivan de la experiencia, de ahí la existencia de múltiples modelos.

El más usado en la industria es el representado en la familia de normas ISO 25000, que se detalla a continuación.

ISO/IEC 25000

Conocida como SQuaRE (*System and Software Quality Requirements and Evaluation*), es una familia de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto *software*. La familia ISO/IEC 25000 es el resultado de la evolución de otras normas anteriores, especialmente de las normas ISO/IEC 9126, que describe las particularidades de un modelo de calidad del producto *software*, e ISO/IEC 14598, que abordaba el proceso de evaluación de productos *software*. Esta familia de normas ISO/IEC 25000 se encuentra compuesta por cinco divisiones (ver figura 3).

- ISO/IEC 2500n: Los estándares que forman esta división definen todos los modelos comunes, términos y definiciones referidos además por todos los demás estándares de la serie SQuaRE.
- ISO/IEC 2501n: Los estándares que forman esta división presentan modelos detallados de calidad para sistemas de computación y productos de *software*, calidad en uso y datos.
- ISO/IEC 2502n: Los estándares que forman esta división incluyen un modelo de referencia de medición de la calidad del producto de *software*, definiciones matemáticas de medidas de calidad y una guía práctica para su aplicación. Las medidas presentadas se aplican a la calidad y la calidad del producto de *software* en uso.
- ISO/IEC 2503n: El estándar que forma esta división ayuda a especificar los requisitos de calidad. Estos requisitos de calidad se pueden utilizar en el proceso de obtención de requisitos de calidad para desarrollar un producto de *software* o como insumo para un proceso de evaluación.
- ISO/IEC 2504n: Los estándares que forman esta división proporcionan requisitos, recomendaciones y pautas para la evaluación de productos de *software*.

Figura 3: Divisiones de la norma ISO 25000

La extensión SQuaRE (ISO / IEC 25050 a ISO / IEC 25099) está diseñada para contener estándares internacionales y / o informes técnicos de calidad de productos de *software* o sistemas que abordan dominios de aplicación específicos o que pueden utilizarse para complementar uno o más estándares internacionales de SquaRE (ISO 25000 calidad del producto de *software*, 2018).

Si es importante definir los criterios para evaluar la calidad, también lo es la definición de un proceso, un marco de trabajo para agregar la información y obtener un índice de calidad. La división 2504n describe qué hacer para valorar la calidad de un producto, pero no cómo hacerlo.

1.4 Evaluación y selección de productos de *software*.

Es importante analizar primero el concepto de evaluación. Según la Real Academia, evaluación es la acción de estimar, apreciar, calcular o señalar el valor de algo. Al evaluar se emite un juicio o valoración sobre algo, teniendo como base la información del objeto a valorar (evidencias, mediciones) y un conjunto de criterios como patrón. Con el resultado de la evaluación, se toman decisiones y se conduce hacia la mejora continua. Evaluar conlleva una valoración subjetiva (cualificación) y objetiva (cuantificación).

La evaluación de la calidad del producto de *software* tiene como propósito emitir un juicio de cuán bueno puede ser este para una función determinada. Permite seleccionar entre dos o más alternativas; predecir los valores de las características de calidad y conceptualizar un nuevo producto (Pérez, 2018).

Como resultado del proceso de evaluación, se obtiene un valor global de la calidad (cuantificación) y una valoración (cualificación), que permite aislar los problemas que pudiera presentar el producto. Dicho proceso es esencial para la toma de decisiones por las empresas productoras, evaluadoras y las que adquieren el *software*.

La evaluación, como cualquier proceso, posee un conjunto de actividades interrelacionadas, que transforman elementos de entrada en salidas. En este caso, tiene como entrada los productos, que se conducen por un flujo de actividades, entre las que se incluyen la medición y el análisis, y obtiene como salida el resultado de la valoración de la calidad (Pérez, 2018).

Para evaluar y juzgar la calidad, se necesita determinar los criterios que caracterizan la calidad del producto. En el *software*, dichos criterios se encuentran estructurados y organizados en los modelos de calidad. Además, están los estándares ISO 14598 y 25040, que ofrecen un *framework* para la evaluación, que detalla el proceso a través de un flujo de actividades y tareas.

El modelo de calidad, representa los criterios a tener en cuenta en el proceso de evaluación, organizados de forma jerárquica. Asociado a estos, se encuentran las medidas que permiten valorarlos. Las medidas provienen de diferentes fuentes: pruebas automáticas, encuestas a expertos, listado de No Conformidades (NC), entre otras.

La evaluación de la calidad de los productos de *software*, se modela como un problema de toma de decisión multicriterio, en grupo y en un ambiente de incertidumbre (Fernández Pérez, 2018). Se destacan los siguientes elementos que permiten tratarla así:

- Se tienen en cuenta varios criterios, estructurados en modelo de calidad
- Se manipula información de diferente naturaleza, tanto objetiva como subjetiva.
- Se maneja información que puede ser considerada vaga, incompleta e imprecisa, a partir de valoraciones de conceptos y de la experiencia de los decisores.
- Es un proceso en el que intervienen varias personas, por lo que se presenta la incertidumbre en diferentes grados y niveles.
- Con el objetivo de tratar la incertidumbre y heterogeneidad de los datos, se combinan estos métodos con los conjuntos difusos.

Las soluciones de los problemas de evaluación de la calidad del *software*, en los últimos años van dirigidas a métodos de decisión multicriterio (J.Pang, 2010). De forma general,

todos estos métodos parten de un conjunto de alternativas que son evaluadas para un conjunto de criterios, y a través de una operatoria dada, establecen un orden entre las alternativas y orientan al decisor sobre cuál o cuáles son las mejores.

Sin embargo, a la hora de valorar la efectividad de estos métodos en el proceso de evaluación de la calidad de un producto de *software*, deben tenerse en cuenta las siguientes especificaciones: se estima una gran variedad de medidas definidas en diferentes dominios de datos, con diversas escalas, los cuales valoran factores de índole objetiva y subjetiva, conllevando a datos heterogéneos e imprecisos; la solución debe ser independiente de los datos; el valor obtenido para cada alternativa, debe encontrarse en un rango que sea posible interpretar y clasificar fácilmente; y se ha demostrado en los modelos de calidad la existencia de interdependencia entre los criterios de un mismo nivel de la jerarquía, lo cual lleva a una estructura más compleja, que obliga a valorar la interacción horizontal de los criterios en cada nivel (Fernández Pérez, 2018).

En la selección de los trabajos más destacados se puedo observar que no se encuentra una solución que resuelva en su conjunto los problemas encontrados.

Dichas soluciones, para garantizar su uso práctico, deben implementarse a través de un sistema informatizado, que abarque todo el proceso y facilite la toma de decisiones. Esto es poco mencionado e implementado en la bibliografía consultada.

1.5 Herramientas informáticas para evaluar la calidad de productos de *software* que usan modelos de calidad.

Se analizaron sistemas que se usan en el proceso de agregación de la información en la evaluación de productos de *software* y sistema general para cualquier problema de toma de decisiones multicriterio.

A partir de las investigaciones realizadas a través de los documentos y los sitios oficiales existentes de los productos que se mencionan a continuación se pudo analizar y resumir brevemente las características de los mismos, atendiendo a los MCDM que implementan.

SERVIQUALITY: Es un *software* que pretende facilitar el proceso de calificación de un producto o bien un Servicio de *software*. Con el fin de poderse integrar con otros productos manejados en las organizaciones, SERVIQUALITY busca entre otros aspectos: proporcionar una herramienta a las empresas que quieren invertir en Tecnologías de la Información (TI) y no cuentan con los mecanismos necesarios para elegir la mejor solución en términos de costo, funcionalidad, disponibilidad, portabilidad, amigabilidad,

entre muchos otros aspectos. Poner a disposición de las compañías de TI una herramienta que permita validar la calidad de sus productos o servicios, con el fin de apoyar el ciclo de mejora continua en la organización. De esta manera, tanto el cliente como el proveedor podrán analizar de una forma más objetiva si el resultado obtenido con el producto o servicio cumple con lo pactado (López, 2012). Este sistema no tiene en cuenta para la evaluación, la interdependencia entre criterios, la ambigüedad y heterogeneidad de los datos.

SOFTWARE MPC 2.0: Es un *software* diseñado para facilitar la aplicación de la metodología de toma de decisiones AHP (*Analytic Hierarchy Process*), basada en la comparación por pares. El programa resulta especialmente útil en las decisiones en las que sea necesario considerar numerosos y diferentes tipos de criterios (cuantificables o no) y/o muchas posibles alternativas, así como también cuando sea preciso tener en cuenta diferentes repeticiones de la decisión de uno o diferentes usuarios (Pérez Rodríguez, 2012).

1000Minds: Se basa en un método de toma de decisiones denominado 'PAPRIKA'. PAPRIKA es un acrónimo de ‘’***P****otentially* ***A****ll* ***P****airwise* ***R****an****K****ings of all possible* ***A****lternatives*’’(Potencialmente todas las clasificaciones por parejas de todas las alternativas posibles). El método PAPRIKA está patentado en tres países: Estados Unidos, Nueva Zelanda, Australia. Este *software* pertenece a la línea de apoyo a la toma de decisión desarrollado por 100Minds Ltd. en el año 2002. El sistema está disponible en el idioma inglés solamente y es una herramienta con licencia privativa que implementa el método PAPRIKA. Este DSS es utilizado mayormente para priorizar o elegir entre alternativas en situaciones donde se necesita considerar varios objetivos o criterios simultáneamente (David Pino Gonzáles, 2016).

Super Decisions: El *software* Super Decisions es un programa comercial que se utiliza para resolver problemas de Decisión de Multicriterio. Incluye la solución de problemas de Procesos de Análisis Jerárquico (AHP) y los del tipo Proceso Analítico de Red (ANP) (David Pino Gonzáles, 2016).

EXPERT CHOICE: Es un *software* para la toma de decisiones, implementa el Proceso Jerárquico Analítico (AHP, *Analytic Hierarchy Process*). Este sistema de apoyo a la toma de decisión fue desarrollado en 1983 por Thomas Saaty y Ernest Forman. Su licencia es privativa (David Pino Gonzáles, 2016).

MultiDecision PAAT: Este sistema, desarrollado en el año 2013, es una herramienta web que implementa los métodos PROMETHEE, AHP, ANP y TOPSIS. No está diseñado para problemas que presenten incertidumbre, y no permite la incorporación de nuevos métodos de solución de problemas de toma de decisión (Táramo, 2013).

Decision Making Helper: Es una herramienta orientada a ayudar a sus usuarios a tomar las decisiones más racionales y coherentes posibles de una lista de posibilidades que tendremos que facilitar previamente. El programa corre sobre todas las versiones de Windows y está disponible en los lenguajes inglés, alemán y francés. Los costos para una licencia de usuario son USD 25 o Euro 20 (Decision Making Helper, 2014).

Estas herramientas que implementan métodos multicriterios de toma de decisiones son genéricas, son para solucionar problemas generales y no orientados a la calidad de *software*. Para usarlas correctamente se necesita un especialista en métodos de toma de decisiones. Además, no son flexibles para adicionarles otros métodos multicriterios y son programas privativos.

Por lo que se puede concluir la necesidad de implementar una herramienta basada en el modelo Evalprod la cual valore la interdependencia entre criterios y la heterogeneidad de los datos.

1.5.1 Modelo de selección y evaluación de productos de *software* (Evalprod).

La UCI desarrolló un modelo que palia las dificultades halladas y constituyó tesis doctoral. A continuación, se explica este modelo llamado **Evalprod**.

Evalprod se basa en las fases definidas para la solución de un problema de toma de decisiones, al cual se adiciona la modelación de la información, el análisis/modelación de la interdependencia entre criterios y el manejo de datos heterogéneos y con incertidumbre. Para esto último incorpora elementos de la Soft Computing[1], tales como la lógica difusa y el modelado lingüístico difuso.

Se establecen los siguientes fundamentos teóricos del modelo:

- La modelación de la información se realiza con la normalización de las medidas, el análisis de los conflictos costo/beneficio de los criterios y la unificación de las

[1] *Soft computing* es una rama de la Inteligencia Artificial que engloba diversas técnicas empleadas para solucionar problemas que manejan información incompleta, con incertidumbre y/o inexacta.

medidas en valores lingüísticos. A cada término lingüístico se asocia un número triangular difuso.

- La interdependencia entre criterios se trata mediante Mapas Cognitivos Difusos (MCD).

El modelo para la evaluación y selección de productos de *software* tiene como entrada, al conjunto de los diferentes productos de *software* a valorar y el conjunto de requisitos a tener en cuenta para la evaluación. Como resultado se obtiene la recomendación de un producto mediante el ordenamiento de estos de acuerdo a su calidad. Dicha recomendación se realiza según el Índice de Calidad (IQ) calculado y el tipo de evaluación deseada. IQ indica la medida en que la calidad del producto es valorada, haciéndose representar con la variable lingüística Nivel de Calidad (NQ). NQ se describe sin especificar su representación, debido a que el modelo propuesto se abstrae de la definición de los términos lingüísticos y el dominio de datos al que pertenece la variable. Estos son definidos de acuerdo a los intereses del cliente y deben ser ajustados en dependencia del contexto de aplicación (Pérez, 2018).

1.5.2 Descripción de los componentes del modelo

La articulación entre los diferentes componentes del modelo, está dada sobre la base de las propias relaciones de información establecidas entre las entradas y salidas que en cada uno de ellos se utilizan y procesan (ver figura 4).

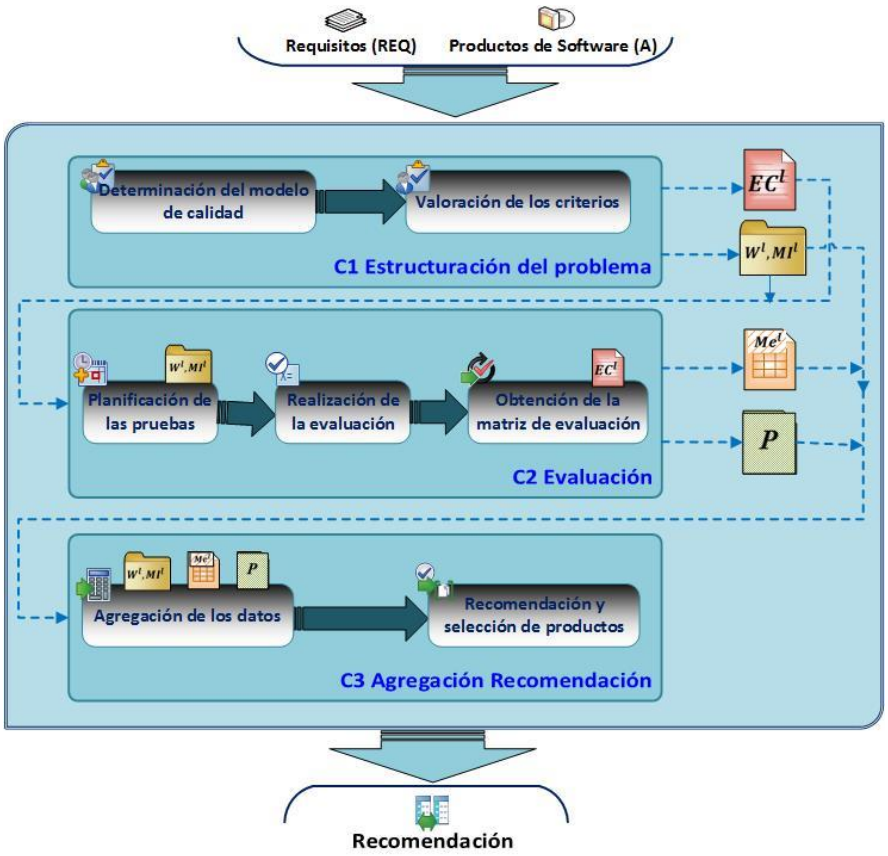


Figura 4: Modelo Evalprod

El primer componente: “Estructuración del problema”, identificado como C1, tiene como entradas el conjunto de alternativas a evaluar y los requisitos de calidad. Se determinan los elementos del Modelo de calidad: los criterios y medidas a evaluar; el peso de los criterios e interdependencia entre los mismos. C1 se divide en dos subcomponentes:

C1.1 “Determinación del modelo de calidad”: establece y adapta el modelo de calidad a partir de los requisitos, sin el análisis de los enlaces horizontales. Formaliza cada medida seleccionada definiendo un dominio de datos con una escala determinada.

C1.2 “Valoración de los criterios”: determina el peso de los criterios en dependencia de los requisitos de calidad. Este puede establecerse a partir de la utilización de diferentes métodos, ya sea por determinación directa de los expertos o mediante la comparación por pares, obteniendo el autovector. La suma de los pesos de los hijos de igual padre (hermanos), ha de ser igual a 1. Modela y analiza la interdependencia entre criterios a través de los MCD.

El segundo componente, denominado “Evaluación”, se identifica con C2. Su función principal es obtener una matriz de evaluación normalizada y realizar la evaluación de los diferentes productos de *software*. C2 se descompone en 2 subcomponentes:

C2.1 Realización de la evaluación: Se ejecutan los diferentes tipos de pruebas de *software*, evaluando cada alternativa, teniendo en cuenta el juicio de los expertos. Este proceso puede ser automático, manual o ambos. Se obtiene el valor de cada medida.

C2.2 Obtención de la matriz de evaluación: Se normaliza la información de las diferentes medidas y se unifican al transformarlas a un mismo dominio de datos; en este caso, números difusos triangulares. Dicha operación es válida sólo en problemas donde la información con que se cuenta es heterogénea. Se conforma la matriz de evaluación con los valores de las medidas normalizadas y unificadas para cada alternativa.

El tercer componente, denominado “Agregación y recomendación”, se identifica como C3. En C3, se realiza la agregación de la información para obtener el índice de calidad IQ de cada alternativa. Dicho índice permite ordenar las alternativas de mejor a más baja calidad. También, puede asociarse una valoración cualitativa de cada alternativa, a través de la variable lingüística nivel de calidad (NQ). Este componente tiene dos subcomponentes: C3.1 Agregación y C3.2 Recomendación.

El primer subcomponente C3.1 Agregación: Al valor de cada criterio, se agrega la influencia del resto, mediante la operatoria de los MCD. A continuación, se pondera el valor obtenido con los pesos de cada uno. Seguidamente, se suman las medidas que conforman las sub-características obteniéndose el valor de cada subcaracterística para cada alternativa. El proceso, se repite para obtener los valores de las diversas características. Por último, se itera de nuevo para conseguir los diferentes IQ de las distintas alternativas.

El subcomponente C3.2. Recomendación: Tiene como su función principal formular una recomendación. Debe mostrarle al usuario, para cada alternativa, su valor de calidad expresado de forma cuantitativa, el índice de calidad asociado a la misma y por último el lugar que ocupa en el ranking de alternativas de acuerdo a su calidad (Pérez, 2018).

Es importante para los especialistas que desarrollan procesos de evaluación de la calidad y selección de productos de *software*, tener una aplicación que soporte el modelo a utilizar. Las soluciones planteadas y obtenidas hasta ahora, de poco sirven si no van acompañadas de un marco operativo que permita aplicarlas en situaciones reales

concretas. Para el proceso de toma de decisiones, es esencial disponer de varios modelos de calidad que permitan evaluar y seleccionar el producto. También se necesitará la implementación de varios métodos de agregación de la información, para poder comparar y evaluar las soluciones obtenidas por estos métodos.

Con esta base, el siguiente epígrafe se enfoca en describir las herramientas, metodologías y lenguaje de desarrollo para la implementación de una herramienta informática de *software* libre que permite informatizar el modelo descrito.

1.6 Herramientas, Metodologías y Lenguajes de Desarrollo

Metodología de desarrollo

Las metodologías de desarrollo de *software* son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos *software*. En la misma se van indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado, indicando además qué personas deben participar en el desarrollo de las actividades y qué papel deben de tener. Además, detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla (BR. Sintya Milena Meléndez Valladarez, 2016).

Existen numerosas metodologías que con el pasar del tiempo se han ido adaptando a las características del producto informático a desarrollar. Un conjunto de estas metodologías son llamadas metodologías tradicionales debido a que priorizan el control del proceso, establecen las actividades involucradas, los diferentes artefactos que se deben producir y las herramientas a utilizar para producir los mismos. A parte de este grupo de metodologías también se pueden encontrar las llamadas metodologías ágiles, debido a que se enfoca más en la colaboración con el cliente y en el proceso de desarrollo incremental del producto basado en iteraciones de corto plazo. Las metodologías ágiles han mostrado una alta efectividad en proyectos con requerimientos muy cambiantes y en aquellos casos en los que se exige reducir notablemente los tiempos de desarrollo, pero sin descuidar la alta calidad del producto (David Pino Gonzáles, 2016).

Características de la Metodología XP: Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de *software*, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los

participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. XP es una metodología ágil para pequeños y medianos equipos desarrollando *software* cuando los requerimientos son ambiguos o rápidamente cambiantes. Además, consta de cuatro fases fundamentales: Planeación, Diseño, Codificación y Pruebas (BR. Sintya Milena Meléndez Valladarez, 2016) (ver figura 5).

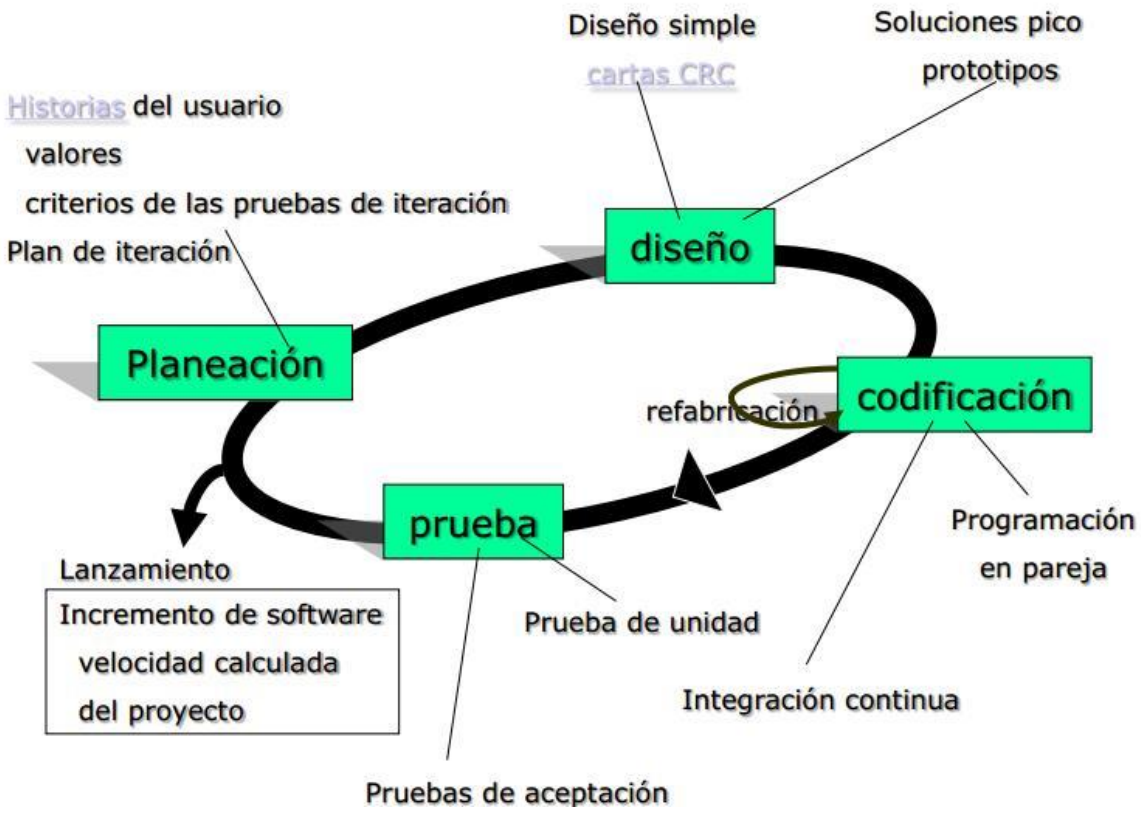


Figura 5: Fases de XP

Las fases de **Planeación** y **Diseño** se detallan en el capítulo 2, en correspondencia con los artefactos generados en las mismas.

Codificación

Los conceptos más importantes de codificación en esta metodología son los siguientes:

Disponibilidad del cliente, uno de los requerimientos de XP es tener al cliente disponible durante todo el proyecto. No solamente como apoyo a los desarrolladores, sino formando parte del grupo. El involucramiento del cliente es fundamental para que pueda

desarrollarse un proyecto con la metodología XP. Al comienzo del proyecto, este debe proporcionar las historias de usuarios. Pero, dado que estas historias son expresamente cortas y de alto nivel, no contienen los detalles necesarios para realizar el desarrollo del código. Estos detalles deben ser proporcionados por el cliente, y discutidos con los desarrolladores, durante la etapa de desarrollo.

Uso de estándares: XP promueve la programación basada en estándares, de manera que sea fácilmente comprensible por todo el equipo, y que facilite la recodificación.

Programación Dirigida por las Pruebas: En las metodologías tradicionales, la fase de pruebas, incluyendo la definición de los *test*, es usualmente realizada sobre el final del proyecto, o el final del desarrollo de cada módulo. La metodología XP propone un modelo inverso, primero se escribe los *test* que el sistema debe pasar. Luego, el desarrollo debe ser el mínimo necesario para pasar las pruebas previamente definidas. Las pruebas a las que se refiere esta práctica, son las pruebas unitarias, realizadas por los desarrolladores. La definición de estos test al comienzo, condiciona o dirige el desarrollo.

Programación en pares: XP propone que se desarrolle en pares de programadores, ambos trabajando juntos en un mismo ordenador. Si bien parece que ésta práctica duplica el tiempo asignado al proyecto, al trabajar en pares se minimizan los errores y se logran mejores diseños, compensando la inversión en horas. El producto obtenido es por lo general de mejor calidad que cuando el desarrollo se realiza por programadores individuales.

Integraciones permanentes: Todos los desarrolladores necesitan trabajar siempre con la última versión del *software*. Realizar cambios o mejoras sobre versiones antiguas causan graves problemas, y retrasan al proyecto. Es por eso que XP promueve publicar lo antes posible las nuevas versiones, aunque no sean las últimas, siempre que estén libres de errores. Idealmente, todos los días deben existir nuevas versiones publicadas. Para evitar errores, solo una pareja de desarrolladores puede integrar su código a la vez.

Propiedad colectiva del código: En un proyecto XP, todo el equipo puede contribuir con nuevas ideas que apliquen a cualquier parte del proyecto. Asimismo, una pareja de programadores puede cambiar el código que sea necesario para corregir problemas, agregar funciones o recodificar.

Ritmo sostenido: La metodología XP indica que debe llevarse un ritmo sostenido de trabajo. El concepto que se desea establecer con esta práctica es planificar el trabajo de

forma a mantener un ritmo constante y razonable, sin sobrecargar al equipo (BR. Sintya Milena Meléndez Valladarez, 2016).

Pruebas

Cuando se tienen bien implementadas las pruebas no habrá temor de modificar el código del otro programador en el sentido que, si se daña alguna sección, las pruebas mostrarán el error y permitirán encontrarlo. Uno de los elementos que podría obstaculizar que un programador cambie una sección de código funcional es precisamente hacer que este deje de funcionar. Si se tiene un grupo de pruebas que garantice su buen funcionamiento, este temor se mitiga en gran medida. Las pruebas con las que consta XP son: pruebas unitarias y de aceptación.

Se decidió hacer uso de esta metodología para el desarrollo de la aplicación debido a las ventajas que brindan sus características. Alguna de estas es la disminución del tiempo de desarrollo sin que se afecte la calidad del producto. Además, genera poca cantidad de artefactos y se encuentra enfocada principalmente al código (David Pino Gonzáles, 2016).

Lenguajes de programación

Python 2.7, es un lenguaje interpretado, multiplataforma, que usa tipado dinámico; se emplea como lenguaje de programación del lado del servidor. Su sintaxis, favorece el código legible. Como lenguaje multiparadigma, soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Posee una licencia de código abierto, denominada *Python Software Foundation License*, llegando a ser uno de los más difundidos en el ámbito científico, y siendo la base para los desarrollos de robustos paquetes de cálculo numérico, como *numpy, scikit-fuzzy, scikit-criteria, scikit-learn, mathplolib (W- ictea, 2018)*.

HTML 5 y JavaScript, son utilizados como lenguajes de programación en el lado del cliente; *HTML 5* (*HyperText Markup Language*), para estructurar y presentar el contenido para la web. Debido a su alto grado de portabilidad, se visualizan las páginas con cualquier sistema operativo. *JavaScript*, por su parte, es un lenguaje de programación interpretado, que no necesita ser compilado. Consigue mejoras en la interfaz de usuario y páginas web dinámicas, además, está totalmente integrado con HTML y CSS (Guía web 2.0, 2016).

Entorno integrado de desarrollo

PyCharm 2018.1, desarrollado por la empresa JetBrains, es un IDE o entorno de desarrollo integrado multiplataforma utilizado para desarrollar en el lenguaje de programación Python. Proporciona análisis de código, depuración gráfica, integración con VCS / DVCS y soporte para el desarrollo web con Django, entre otras bondades (DesdeLinux, 2018).

Sistema Gestor de Base de Datos

PostgreSQL 9.5, PostgreSQL es un gestor de bases de datos orientadas a objetos (SGBDOO o ORDBMS en sus siglas en inglés) muy conocido y usado en entornos de *software* libre porque cumple los estándares SQL92 y SQL99, y también por el conjunto de funcionalidades avanzadas que soporta, lo que lo sitúa al mismo o a un mejor nivel que muchos SGBD comerciales (Sistema gestor de bases de datos, 2013).

Framework:

Django 1.9, es un *framework* web gratuito, de código abierto, escrito en Python, que sigue el patrón arquitectónico *Model-Template-View* (*MVT*). Su objetivo principal es facilitar la creación de sitios web complejos basados en bases de datos. Enfatiza la reutilización y la "capacidad de conexión" de los componentes. Además, posibilita el desarrollo rápido y el principio de no repetir. Asimismo, proporciona una interfaz administrativa opcional de creación, lectura, actualización y eliminación, generada dinámicamente mediante la introspección. Se configura a través de modelos de administración. Requiere *Python 2.7, 3.4 o 3.5*. Dicha versión, continúa extendiendo el soporte para funcionalidades específicas de *PostgreSQL*, integrando ahora funciones específicas de agregación. Se usan los paquetes *django-grappelli* y *django-reversion,* que son extensiones del *framework* para el trabajo con las interfaces gráficas y el control de la administración (Django, 2018).

Control de versiones

Git, es un *software* de control de versiones que permite la gestión distribuida, diseñado por Linus Torvalds, pensando en la eficiencia y confiabilidad del mantenimiento de versiones de aplicaciones cuando las mismas tienen un gran número de archivos de código fuente. Entre sus ventajas se encuentra la gestión eficiente de proyectos grandes, dada la rapidez de gestión de diferencias entre archivos, entre otras mejoras de optimización de la velocidad de ejecución. Es destacable, asimismo, el fuerte apoyo al desarrollo no lineal, por ende, su rapidez en la gestión de ramas y mezclado de diferentes versiones. Los almacenes de información pueden publicarse por HTTP, FTP, rsync, o

mediante un protocolo nativo, ya sea a través de una conexión TCP/IP simple o de cifrado SSH (W- ictea, 2018)

Conclusiones Parciales:

A partir del análisis y el estudio realizado a los diferentes elementos necesarios para modelar el problema, se llegó a las consideraciones siguientes:

- El uso de técnicas de Soft Computing como la lógica difusa, la modelación lingüística y los mapas cognitivos difusos permiten la resolución de problemas como la heterogeneidad, ambigüedad de la información y la interdependencia entre criterios.
- Los sistemas informáticos que permiten dar solución a problemas de toma de decisión multicriterio en su mayoría son sistemas privativos que se enmarcan en la solución de problemas genéricos; además, ninguno integra los principios y características necesarios para llevar a cabo las actividades relacionadas al proceso de evaluación de la calidad de software.
- El modelo de selección y evaluación de productos de software Evalprod es engorroso en su ejecución, si no se desarrolla un marco operativo que permita aplicarlo de forma eficiente en situaciones reales concretas.
- Se ha seleccionado como guía para el desarrollo de software la metodología XP, además de un conjunto de herramientas y tecnologías unificadas por un elemento en común, el lenguaje de desarrollo Python.

Capítulo 2. Análisis y diseño de la aplicación

Introducción

En el presente capítulo se describe la propuesta de solución al problema planteado, así como, los distintos elementos de la planificación; diseño e implementación relacionados con la metodología utilizada. Para ello, se inicia con un análisis de las características, componentes, particularidades y se describen cada una de las funcionalidades del sistema que se desea implementar. Como parte del proceso ingenieril, se exponen los principales artefactos generados por la metodología seleccionada como los requisitos de *software* identificados, las historias de usuario y la arquitectura empleada. Se exponen por último las conclusiones parciales sobre el trabajo realizado.

2.1 Recopilación de información para el sistema a desarrollar

Para el levantamiento informacional del sistema se le realizaron entrevistas al cliente. En ellas se recopiló la información necesaria para identificar las funcionalidades a desarrollar como: la entrada y gestión de los modelos de calidad, de los productos a valorar, la relación entre las características, el peso de los criterios, el cálculo de los índices de calidad y la recomendación dada.

2.2 Características del sistema

El modelo Evalprod está soportado por SoftQuality, la plataforma de ayuda a la toma de decisiones en el proceso de evaluación y selección de productos de *software*, según su calidad. La herramienta implementa variantes difusas para la agregación de la información heterogénea y ambigua, facilitando al usuario poder elegir la solución más apropiada en base a la información disponible.

El diseño flexible de la herramienta, hace posible la adaptación del modelo de calidad, según las necesidades de evaluación en cuanto a requisitos. Esto le proporciona al usuario objetividad y rapidez al obtener el resultado de una evaluación. El sistema se desarrolló en Python como lenguaje de programación y Django como *framework* de desarrollo, Django es un *framework* web de alto nivel que fomenta el desarrollo rápido y el diseño limpio y pragmático.

En el siguiente acápite, se expone una descripción de la herramienta propuesta, y los principales conceptos que se tuvieron en cuenta en su desarrollo. Se realiza el levantamiento de requisitos junto al cliente, dando como resultado las historias de usuario

que define la metodología de desarrollo seleccionada, y los requisitos no funcionales que debe garantizar el sistema en cuanto a rendimiento, *hardware*, *software*, soporte, interfaz y usabilidad.

Con motivo de realizar una eficiente distribución de las responsabilidades ante el trabajo a realizar, la metodología XP propone una serie de actores que intervienen en el proceso de creación del *software*. Los roles que propone la metodología, así como, las personas asignadas en el presente trabajo se muestran a continuación en la tabla 1.

Tabla 1: Asignación de los roles que propone XP.

Roles que define la metodología XP	**Encargado**
Cliente	Yamilis Fernández, Naivy Pujol
Programador	Ramón Guillermo Álvarez
Consultor	Ramón Guillermo Álvarez
Tester (Encargado de pruebas)	Ramón Guillermo Álvarez
Tracker (Encargado de seguimiento)	Ramón Guillermo Álvarez
Entrenador (*Coach*)	Ramón Guillermo Álvarez
Gestor (*Big Boss*)	Ramón Guillermo Álvarez

2.2.1 Requisitos del sistema

El proceso de captura de requisitos es una etapa de suma importancia dentro del proceso de desarrollo de *software*. Este se encarga de analizar las necesidades del cliente, describirlas en detalle para conformar el sistema de la manera más precisa, cumpliendo siempre con las especificaciones deseadas. En el presente trabajo, se identificaron los requisitos que se muestran a continuación en la tabla 2:

Tabla 2: Requisitos del sistema

Requisitos	**Descripción**
Autenticar usuario	El usuario debe autenticarse en el sistema con el rol correspondiente
Gestionar modelo	Gestionar el modelo de calidad por el cual se va a realizar la evaluación
Gestionar características	Gestionar las características para la

	adaptación del modelo de calidad
Gestionar subcaracterísticas	Gestionar las subcaracterísticas para la adaptación del modelo de calidad
Gestionar medidas	Gestionar las medidas para la adaptación del modelo de calidad
Gestionar productos	Gestionar los productos que se desean evaluar
Conformar la matriz de interdependencia para las características	Introducir los datos correspondientes en la matriz de interdependencia para las características
Conformar la matriz de interdependencia para las subcaracterísticas	Introducir los datos correspondientes en la matriz de interdependencia para las subcaracterísticas
Conformar la matriz de interdependencia para las medidas	Introducir los datos correspondientes en la matriz de interdependencia para las medidas
Conformar la matriz de evaluación	Introducir los datos correspondientes en la matriz de evaluación
Generar los datos	Formalizar los datos introducidos
Mostrar la recomendación de alternativas	Mostrar la recomendación de productos conformada por los parámetros: número difuso, índice de calidad y ranking
Reiniciar datos	Reiniciar los datos para realizar una nueva evaluación
Mostrar ayuda	Mostrar una ayuda que contiene los pasos y consejos necesarios para realizar la

	evaluación

Los requisitos se identificaron a partir de las entrevistas con el cliente, mediante los mismos se pudo confeccionar las historias. Las historias de usuario sirvieron como base para la programación de las funcionalidades y la creación de las interfaces de usuario para el sistema.

2.2.2 Requisitos no funcionales

Para lograr mayor aceptación por parte del usuario final del sistema conformado, se tienen en cuenta también los requisitos no funcionales (RNF). Los RNF se definen como requerimientos que deben dar respuesta a la operatividad del sistema final (ResearchGate, 2018). Se encuentran divididos en dos categorías fundamentales:

- Cualidades de ejecución, como la seguridad y facilidad de uso, que son observables en tiempo de ejecución.
- Cualidades de evolución, como la capacidad de prueba, mantenimiento, ampliación y escalabilidad, que se enfoca en la estructura estática del sistema.

Para el siguiente trabajo, atendiendo a lo anterior, se especifican los requisitos no funcionales de acuerdo a las categorías que propone la norma ISO 9126-1 (ISO/IEC, 2000) en la tabla 3.

Tabla 3: Requisitos no funcionales definidos

Categoría	Número	Descripción
Confiabilidad	RNF1	Comprobar que los datos proporcionados por los usuarios sean valores válidos. Garantizar que el sistema esté disponible las 24 horas del día para garantizar el acceso en todo momento.
Hardware	RNF2	La computadora que hará uso del sistema a desarrollar tendrá como CPU, i3, similar o superior. La memoria RAM debe tener como mínimo 4 Gb o superior.
Software	RNF3	El sistema deberá poder ser ejecutado desde cualquier sistema operativo.
Interfaz de usuario	RNF4	La interfaz tendrá los componentes visuales necesarios para las operaciones del usuario sin sobrecarga de imágenes
Usabilidad	RNF5	La solución debe tener una interfaz gráfica atractiva permitiendo el uso del sistema sin necesidad de mucho conocimiento informático y el acceso a todas

		sus funcionalidades de manera sencilla y directa.
Restricciones de diseño	RNF6	Restricciones de diseño: La aplicación se desarrollará en Python 2.7 como lenguaje de programación, se hará uso de Pycharm 2018.1 como herramienta IDE y se utilizará como gestor de bases de datos PostgreSQL 9.5.

Con la identificación de los requisitos no funcionales se aseguraron los requerimientos necesarios que dan respuesta a la operatividad del sistema SoftQuality.

2.3 Fase de Planeación

La metodología XP plantea la planeación como un diálogo continuo entre las partes involucradas en el proyecto, incluyendo al cliente, a los programadores y a los coordinadores. El proyecto comienza recopilando las historias de usuarios, las que constituyen a los tradicionales casos de uso. Una vez obtenidas estas historias de usuarios, los programadores evalúan rápidamente el tiempo de desarrollo de cada una (BR. Sintya Milena Meléndez Valladarez, 2016).

Los conceptos básicos de la planeación son:

- Las Historias de Usuarios, las cuales son descritas por el cliente, en su propio lenguaje, como descripciones cortas de lo que el sistema debe realizar.
- El Plan de Entregas, establece que las historias de usuarios serán agrupadas para conformar una entrega y el orden de las mismas. Este cronograma será el resultado de una reunión entre todos los actores del proyecto.

Plan de Iteraciones, las historias de usuarios seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo al orden preestablecido.

Reuniones Diarias de Seguimiento, el objetivo es mantener la comunicación entre el equipo y compartir problemas y soluciones (BR. Sintya Milena Meléndez Valladarez, 2016).

2.3.1 Historias de usuarios

Las historias de usuario (HU) representan un requisito (o agrupación de ellos) de *software* escrito en una o dos frases utilizando el lenguaje común del usuario. Proporcionan los detalles sobre la estimación del riesgo y el tiempo necesario para la implementación de

dicha HU. Además, son una forma simple de administrar los requisitos de los usuarios, lo que permite responder rápidamente a los cambios.

Durante el análisis en la fase de Planeación fueron identificadas 14 HU, que recogen los requisitos previamente identificados, además, proporcionan una idea al equipo de desarrollo sobre cómo debe ser su posterior implementación.

Para realizar una correcta planificación de la implementación se debe tener en cuenta la prioridad de cada HU. Partiendo de las necesidades de desarrollo a cada una se le asigna una clasificación que puede ser:

- Alta: fundamentales para el desarrollo del sistema.
- Media: poseen funcionalidades necesarias, pero no imprescindibles para el sistema.
- Baja: constituyen funcionalidades que sirven de ayuda al control de los elementos asociados al equipo de desarrollo y a la estructura.

Para conocer la dificultad y posibles errores durante la implementación de cada HU, el equipo de desarrollo clasifica el riesgo en:

- Alto: cuando la implementación de la HU se considera la posible existencia de errores que lleven a la inoperatividad del código.
- Medio: cuando puedan aparecer errores en la implementación de la HU que puedan retrasar la entrega de la versión.
- Bajo: cuando puedan aparecer errores que pueden ser corregidos con relativa facilidad sin que ocasionen prejuicios para el desarrollo del *software*.

El tiempo determinado para cada HU conocido como puntos estimados, se establece en semanas ideales (40 horas) a lo que se puede concluir que: 0.1 equivale a medio día de trabajo (4 horas). Las HU son representadas mediante tablas con las siguientes secciones:

- Código: código de la HU el cuál es diferente en cada una.
- Nombre de la HU: identificador de la HU que se describe entre los desarrolladores y el cliente.
- Prioridad en negocio: prioridad de acuerdo a la necesidad de desarrollo.
- Riesgo en desarrollo: riesgo de acuerdo a la posibilidad de ocurrencia de errores en el proceso de implementación de la HU.
- Iteración asignada: número de la iteración donde se va a desarrollar la HU.

- Puntos estimados: tiempo estimado de desarrollo de la HU.
- Puntos reales: tiempo real de desarrollo de la HU.
- Descripción: breve descripción de la HU.
- Prototipo de interfaz: imagen de la interfaz de la HU.

Como resultado de un intercambio con el cliente, fueron confeccionadas un total de 14 HU, cada una de ellas respondiendo a las diferentes funcionalidades solicitadas por el cliente. A continuación, en las tablas 4 y 5 se describen dos de las más importantes por su alta prioridad para el sistema.

Tabla 4: Gestionar características

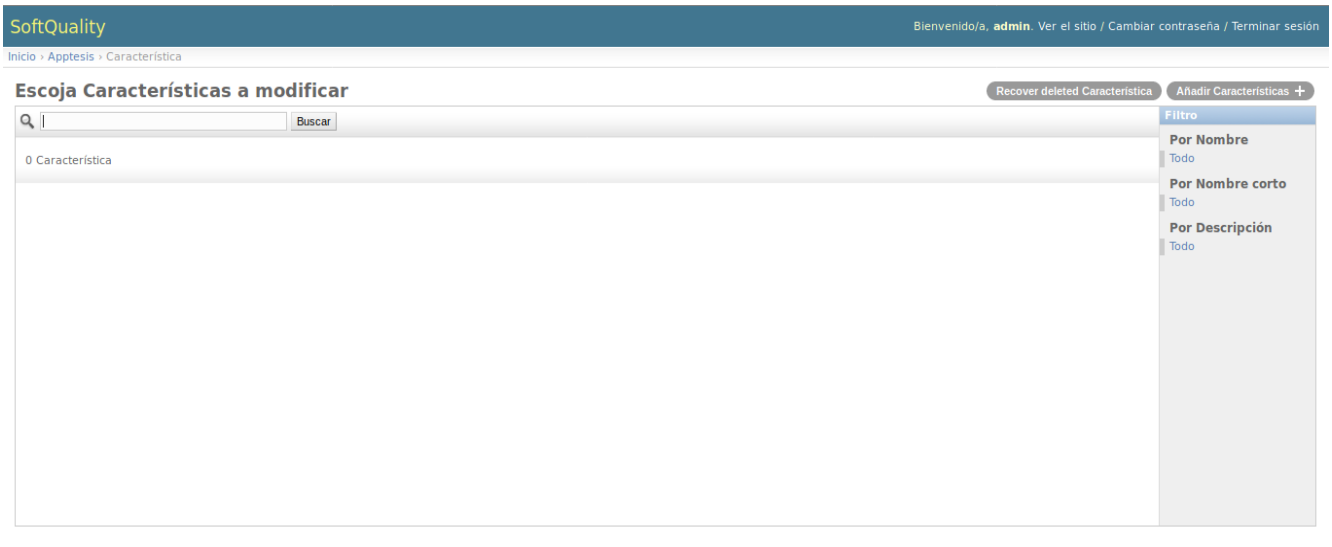
Historia de Usuario	
Código: HU3	Nombre: Gestionar características
Referencia: HU2	Prioridad: Alta
Iteración asignada: 1	Puntos estimados: 0.2
Riesgo en desarrollo: Medio	Puntos reales: 0.2
Descripción: La aplicación debe permitir gestionar las características del modelo de calidad.	
Prototipo de Interfaz: 	

Tabla 5: Confeccionar matriz de interdependencia para las medidas

Historia de Usuario	
Código: HU9	Nombre: Confeccionar matriz de interdependencia para las medidas

Referencia: HU8	Prioridad: Alta
Iteración asignada: 2	Puntos estimados: 0.4
Riesgo en desarrollo: Alto	Puntos reales: 0.4
Descripción: La aplicación debe permitir confeccionar la matriz de interdependencia para las medidas	
Prototipo de Interfaz: 	

2.3.2 Plan de entrega

Teniendo en cuenta el esfuerzo asociado a las historias de usuario y las prioridades del cliente se define una entrega que sea de valor y que tenga una duración de pocos meses. La entrega es dividida en iteraciones de no más de 3 semanas, asignando a cada iteración un conjunto de historias de usuario que serán implementadas (Sanchez , Letelier Torres, & Canós Cerdá, 2003). En la tabla 6 se aprecia la planificación realizada para el presente trabajo.

Tabla 6: Plan de entrega

Plan de entregas		
Iteración	**Fecha de inicio**	**Fecha de entrega**
1	4/ 4/ 18	21/ 4/ 18
2	1/ 5/ 18	16/ 5/ 18

2.3.3 Plan de iteraciones

En la metodología XP, para que exista mayor organización a la hora de desarrollar un *software*, es confeccionado el plan de duración de las iteraciones. El mismo tiene como objetivo fundamental mostrar la iteración en que será implementada cada HU según el

orden correspondiente a cada una de ellas, así como, el tiempo que se ha destinado para cada iteración. A continuación, en la tabla 7 se muestra el plan de iteraciones para el sistema SoftQuality

Tabla 7: Plan de iteraciones

Iteración	Orden de la HU a implementar	Duración de semanas
1	Autenticar usuario	2.0
	Gestionar modelo	
	Gestionar características	
	Gestionar subcaracterísticas	
	Gestionar medidas	
	Gestionar productos	
2	Conformar la matriz de interdependencia para las características.	2.8
	Conformar la matriz de interdependencia para las subcaracterísticas.	
	Conformar la matriz de interdependencia para las medidas	
	Conformar la matriz de evaluación	
	Generar datos	
	Ejecutar la evaluación	
	Reiniciar datos	
	Mostrar ayuda	

2.4 Fase de Diseño

La metodología XP hace especial énfasis en los diseños simples y claros. Los conceptos más importantes de diseño en esta metodología son los siguientes:

Tarjetas CRC, las tarjetas CRC (Clase-Responsabilidad-Colaborador) es un artefacto generado por la metodología XP durante la fase de Diseño para diseñar la solución informática según el paradigma de la programación orientada a objetos

Simplicidad, un diseño simple se implementa más rápidamente que uno complejo. Por ello XP propone implementar el diseño más simple posible que funcione.

Recodificación, consiste en escribir nuevamente parte del código de un programa, sin cambiar su funcionalidad, a los efectos de crearlo más simple, conciso y entendible. La metodología XP sugiere recodificar cada vez que sea necesario

Metáforas, XP sugiere utilizar este concepto como una manera sencilla de explicar el propósito del proyecto, así como guiar la estructura del mismo. Una buena metáfora debe ser fácil de comprender para el cliente y a su vez debe tener suficiente contenido como para que sirva de guía a la arquitectura del proyecto (BR. Sintya Milena Meléndez Valladarez, 2016).

2.4.1 Tarjetas CRC

En el proceso de diseño, las tarjetas CRC jugaron un importante papel debido a que sirvieron de base para el modelo entidad relación, se confeccionó a través de ellas las bases de datos del sistema. Cada Tarjeta CRC se convirtió en un objeto, sus responsabilidades en métodos públicos y sus colaboradores en llamados a otras clases. A continuación, en la tabla 8 se presenta la tarjeta CRC para la clase Característica.

Tabla 8: Tarjeta CRC para Características

Clase: Característica	
Responsabilidades:	**Colaboradores:**
Nombre (Devuelve el nombre de una característica)	Característica
Nombre corto (Devuelve el nombre corto de una característica)	Característica
Descripción (Devuelve la descripción de una característica)	Característica
Peso (Devuelve el peso de una característica)	Característica
Id (Devuelve el id de una característica)	Característica
Padre (Devuelve una instancia de su clase padre)	Modelo de calidad

2.5 Arquitectura

La arquitectura de *software* es el conjunto de técnicas metodológicas desarrolladas con el fin de facilitar la programación. Hace referencia a un grupo de abstracciones y patrones que brindan un esquema de referencia útil para guiarse en el desarrollo de *software* dentro de un sistema informático. Establece todos los fundamentos para que los analistas,

diseñadores y programadores trabajen en una línea común que permita alcanzar los objetivos y necesidades del sistema (David Pino Gonzáles, 2016).

2.5.1 Patrón de arquitectura

Los patrones de arquitectura son aquellos que expresan un esquema organizativo estructural fundamental para sistemas de *software*.

El patrón de arquitectura implementado, ha sido el MTV de Django, por las ventajas que este ofrece (Ver figura 6). Django fue diseñado para promover la estricta separación entre las piezas de una aplicación. Si se sigue esta filosofía, es fácil hacer cambios en un lugar particular de la aplicación sin afectar otras piezas. Es importante separar la lógica de negocios de la lógica de presentación para lograr esto se usa el sistema de plantillas de Django. Con la capa de la base de datos, se aplica la misma filosofía para el acceso lógico a los datos.

Estas tres piezas juntas — la lógica de acceso a la base de datos, la lógica de negocios, y la lógica de presentación — comprenden un concepto que a veces es llamado el patrón de arquitectura de *software* Modelo-Vista-Controlador (MVC).

En este patrón, el "Modelo" hace referencia al acceso a la capa de datos, la "Vista" se refiere a la parte del sistema que selecciona qué mostrar y cómo mostrarlo, y el "Controlador" implica la parte del sistema que decide qué vista usar, dependiendo de la entrada del usuario, accediendo al modelo si es necesario.

Django sigue el patrón MVC tan al pie de la letra que puede ser llamado un *framework* MVC. Someramente, la M, V y C se separan en Django de la siguiente manera:

- M, la porción de acceso a la base de datos, es manejada por la capa de la base de datos de Django.
- V, la porción que selecciona qué datos mostrar y cómo mostrarlos, es manejada por la vista y las plantillas.
- C, la porción que delega a la vista dependiendo de la entrada del usuario, es manejada por el *framework* mismo siguiendo la URLconf y llamando a la función apropiada de Python para la URL obtenida.

Debido a que la "C" es manejada por el mismo *framework* y la parte más importante se produce en los modelos, las plantillas y las vistas, Django es conocido como un Framework MTV. En el patrón de arquitectura de *software* MTV (ver imagen 6):

- M significa "*Mode*l" (Modelo), la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos: cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tiene, y las relaciones entre los datos.
- T significa "*Template*" (Plantilla), la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación: como algunas cosas son mostradas sobre una página web u otro tipo de documento.
- V significa "*View*" (Vista), la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada: puedes pensar en esto como un puente entre los modelos y las plantillas (Django Software Corporation, 2015).

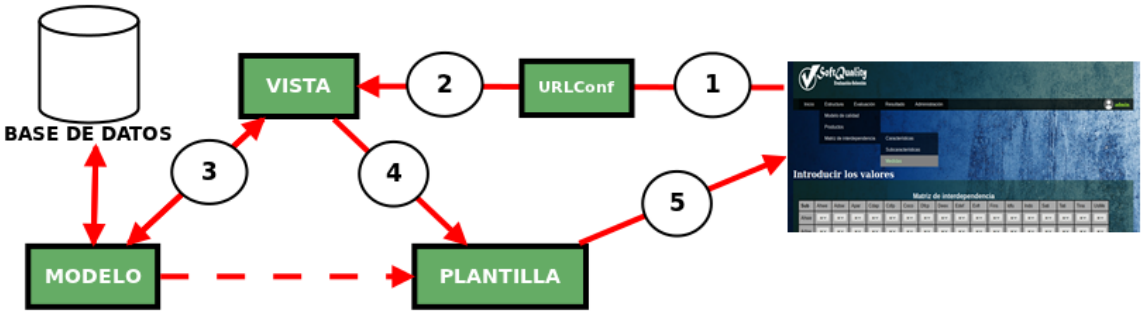


Figura 6: Patrón de arquitectura MTV

2.6 Patrones de diseño

Los patrones de diseño brindan un esquema que facilita el trabajo y aportan mayor organización y claridad en la estructura de la aplicación. El sistema que se desea implementar basa su arquitectura en 3 capas, empleando patrones arquitectónicos y de diseño, donde tiene gran importancia la utilización de patrones GRASP y los patrones GOF.

2.6.1 Patrones GRASP (Patrones de *Software* para la Asignación General de Responsabilidad)

Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable.

El uso de estos patrones es un eslabón fundamental para el desarrollo de una aplicación con la calidad requerida. Estos patrones se describen a continuación:

Bajo acoplamiento: El Bajo Acoplamiento es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño. Estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento (David Pino Gonzáles, 2016). En SoftQuality se ve reflejado el uso de este patrón en las clases pertenecientes a models.py, donde no existe herencia entre ninguna de ellas, y en el caso de la clase QualityModels esta no depende en lo absoluto de las demás.

Alta Cohesión: Asignar una responsabilidad de modo que la cohesión siga siendo alta. Representa una clase con responsabilidades moderadas en un área funcional, colaborando con otras para concretar tareas. Diseño más claro y comprensible. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme (David Pino Gonzáles, 2016). El uso de este patrón se evidencia en las clases pertenecientes a models.py, un ejemplo es la clase Subcharacteristic la cual se encarga únicamente de definir las subcaracterísticas mediante los atributos correspondientes.

Experto: Es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen. Ofrece una analogía con el mundo real (David Pino Gonzáles, 2016).Un ejemplo de la aplicación de este patrón en SoftQuality es cuando utilizas el método Obejct de Django en un objeto, este método brinda los atributos de la clase por la cual se define ese objeto.

Creador: El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento, se debe buscar una clase de objeto que agregue, contenga y realice otras operaciones sobre este tipo de instancias (David Pino Gonzáles, 2016). En SoftQuality las clases pertenecientes a models.py son las encargadas de crear los objetos.

Controlador: Es un evento generado por actores externos. Se asocian con operaciones del sistema, operaciones del sistema como respuestas a los eventos del sistema, tal como se relacionan los mensajes y los métodos (David Pino Gonzáles, 2016). SoftQuality hace uso de este patrón en la capa de la lógica de negocios (*Views*), que es la encargada de decidir lo que se muestra en las plantillas.

2.6.2 Otros patrones importantes.

Los patrones GOF se descubren como una forma indispensable de enfrentarse a la programación a raíz del libro "*Design Patterns—Elements of Reusable Software*" de Erich Gamma, Richard Helm, Ralph Jonson y John Vlissides, a partir de entonces estos patrones son conocidos como los patrones de la pandilla de los cuatro (GoF, gang of four), de estos patrones se utilizaron los siguientes en el desarrollo del sistema:

Patrón Fachada: Es un patrón de diseño de tipo estructural. Proporciona una interfaz unificada de alto nivel para un subsistema, que oculta las interfaces de bajo nivel de las clases que lo implementan. Con esto se consiguen dos objetivos fundamentales: hacer el subsistema más fácil de usar y desacoplar a los clientes de las clases del subsistema (David Pino Gonzáles, 2016).

Conclusiones Parciales

A partir del análisis y diseño del sistema se arribó a las siguientes consideraciones:

- La definición de las 14 HU y los 7 requisitos no funcionales brinda una guía descriptiva de cómo se comportará la implementación del sistema.

- La correcta identificación de los requisitos del sistema, la caracterización del problema, el uso del patrón de arquitectura de software *Model-Template-View*, unido al empleo de los patrones de diseño GRASP, permitieron obtener un diseño robusto de la aplicación.

- Las definiciones de las Tarjetas CRC aportaron en la identificación de las responsabilidades de las clases y las colaboraciones de las mismas.

Capítulo 3 Implementación y pruebas al sistema

Introducción

El presente capítulo tiene como objetivo esencial presentar los resultados de la implementación y pruebas realizadas a la aplicación. Se describen los estilos de programación utilizados, las pruebas unitarias realizadas al código y las pruebas de aceptación realizadas en conjunto con el cliente.

3.1 Tareas de Ingeniería

La Metodología XP define como tareas de ingeniería a todas las tareas que se realizan en cada iteración. A cada una de las tareas se le asigna un programador, el cual, será responsable de ejecutarla.

A continuación, en la tabla 9 se muestra las tareas de ingeniería definidas para la realización de las historias de usuario.

Tabla 9: Tareas de ingeniería

Iteración	Historias de usuario	Tareas
1	Autenticar usuario	-Realizar la autenticación de usuarios
	Gestionar modelo	-Crear la clase QualityModel en models.py -Registrar y configurar la clase QualityModel en admin.py -Hacerle pruebas de aceptación
	Gestionar características	-Crear la clase Characteristic en models.py -Registrar y configurar la clase Characteristic en admin.py -Hacerle pruebas de aceptación

	Gestionar subcaracterísticas	-Crear la clase Subcharacteristic en models.py -Registrar y configurar la clase Subcharacteristic en admin.py -Hacerle pruebas de aceptación
	Gestionar medidas	-Crear la clase Measure en models.py -Registrar y configurar la clase Measure en admin.py -Hacerle pruebas de aceptación
	Gestionar productos	-Crear la clase Product en models.py -Registrar y configurar la clase Product en admin.py -Hacerle pruebas de aceptación
2	Confeccionar matriz de interdependencia para características	-Crear prueba unitaria -Implementar funcionalidad mtz_inter_chara -Ejecutar prueba unitaria
	Confeccionar matriz de interdependencia para subcaracterísticas	-Crear prueba unitaria -Implementar funcionalidad mtz_inter_subchara -Ejecutar prueba unitaria
	Confeccionar matriz de interdependencia para medidas	-Crear prueba unitaria -Implementar funcionalidad mtz_inter_measure -Ejecutar prueba unitaria

	Confeccionar matriz de evaluación	-Crear prueba unitaria -Implementar funcionalidad mtz_eval -Ejecutar prueba unitaria
	Generar datos	-Crear prueba unitaria -Implementar funcionalidad general_vari -Ejecutar prueba unitaria
	Ejecutar la evaluación	-Crear prueba unitaria -Implementar funcionalidad final_result -Ejecutar prueba unitaria
	Reiniciar datos	-Crear prueba unitaria -Implementar funcionalidad reset_data -Ejecutar prueba unitaria
	Consultar ayuda	-Crear prueba unitaria -Implementar funcionalidad help -Ejecutar prueba unitaria

3.2 Estilos de programación

Los estilos de programación son reglas que se utilizan para estructurar el código fuente de una aplicación en desarrollo. En este acápite se describirán los estándares y reglas de programación utilizadas en la implementación del sistema.

Nombre de las clases: El estilo de capitalización utilizado para la notación de las clases es propio, cada subpalabra que conforme el nombre de una función comenzará con una letra minúscula (Ver figuras 7 y 8).

Se definieron un grupo de reglas con el fin de lograr una mayor claridad en el código:

- Utilizar nombres descriptivos para los nombres de las estructuras, atributos y parámetros.
- Los comentarios deben estar en el mismo nivel del código.

- Se deberá realizar una sola declaración por línea con el fin de facilitar los comentarios.
- No debe haber espacios en blanco entre los nombres de los métodos y el paréntesis (que abre su lista de parámetros).
- Cuando una expresión no entra en una línea, se rompe de acuerdo con estos principios:
- Romper después de una coma.
- Romper antes de un operador.

Ejemplo:

```
#ejecutando algoritmo de evaluacion

def final_result(request):
    errors = []
    matrixview = []
```

Figura 7: Declaración de una funcionalidad

```
for element in mtz_result_1:
    for element_1 in width_measure:
        if (element[1] == element_1):
            mtz_result_2[element] = [mtz_result_1[element][0] * width_measure[element_1],
                                     mtz_result_1[element][1] * width_measure[element_1],
                                     mtz_result_1[element][2] * width_measure[element_1]]
```

Figura 8: Salto de línea en una asignación

3.2.1 Pruebas de *software*

En el proceso de desarrollo de *software* una de las fases más importante es la fase de pruebas, debido a que a través de ella se validan que los requisitos del *software* han sido cumplidos y garantiza que el sistema posee la calidad requerida.

3.2.2 Pruebas Unitarias

Las pruebas unitarias son diseñadas por los desarrolladores con el objetivo de verificar el código y detectar posibles errores durante la ejecución del *software* a las funcionalidades de las clases. Todos los módulos deben pasar este tipo de pruebas antes de ser liberados o publicados. Por otra parte, como se mencionó anteriormente, las pruebas deben ser definidas antes de realizar el código. Que todo código liberado pase correctamente las pruebas unitarias, es lo que habilita que funcione la propiedad colectiva del código. Cuando se encuentra un error éste debe ser corregido inmediatamente, y se deben tener

precauciones para que errores similares no vuelvan a ocurrir. Asimismo, se generan nuevas pruebas para verificar que el error haya sido resuelto (BR. Sintya Milena Meléndez Valladarez, 2016).

Las pruebas unitarias se realizaron utilizando la clase TestCase del *framework* de desarrollo Django. Esta librería permite conformar las pruebas unitarias correspondientes a una funcionalidad, luego estas son ejecutadas en la consola de Python mediante el comando Shell, pudiéndose determinar si la funcionalidad se comporta de la manera esperada. A continuación,en la tabla 10 se muestra un ejemplo de prueba unitaria realizada al sistema.

Tabla 10: Prueba unitaria para la clase "final_result"

Prueba unitaria	
Nombre: final_result	
Estado: Satisfactorio	Tipo de prueba: Caja blanca
Ejecutado por: Yamilis Fernández	Verificado por: Yamilis Fernández
Criterio de aceptación: Obtener una recomendación de las alternativas	
Resultado: Satisfactorio	

Análisis de las pruebas unitarias

Luego de haber implementado las historias de usuario planificadas en la primera iteración del desarrollo del sistema se realizó la primera iteración de pruebas unitarias detectándose veinticinco no conformidades (NC). La mayor parte de estas, estaban asociadas a la entrada de datos al sistema y al manejo de los atributos en las clases. Luego de haber implementado las historias de usuario planificadas en la segunda iteración del desarrollo del sistema, corregidas ya todas las NC encontradas, se realizó la segunda iteración de pruebas unitarias detectándose catorce NC, dos de las cuales estaban asociadas al manejo de las clases models.py en la vista (views.py) y a la visualización de los datos de salidas que muestra el sistema, el resto a la solución del problema.

3.2.3 Pruebas de Aceptación

Las pruebas de aceptación tienen como objetivo fundamental validar el nivel la satisfacción del cliente con respecto al *software* desarrollado. Estas son creadas en base a las historias de usuarios, en cada ciclo de la iteración del desarrollo del *software*. El cliente debe especificar uno o diversos escenarios para comprobar que una historia de

usuario ha sido correctamente implementada. Asimismo, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución. Una historia de usuario no se puede considerar terminada hasta que pase correctamente todas las pruebas de aceptación (BR. Sintya Milena Meléndez Valladarez, 2016).

A continuación, en la tabla 11 se muestra un ejemplo de prueba de aceptación realizada al sistema.

Tabla 11: Prueba de aceptación de la historia de usuario "Conformar matriz de evaluación"

Prueba de aceptación	
Nombre: Prueba de aceptación Conformar matriz de evaluación	Historia de usuario: Conformar matriz de evaluación
Responsable: Yamilis Fernández	
Descripción: Prueba de funcionalidad para Conformar la matriz de evaluación.	
Condiciones de ejecución: Debe haberse elegido un método y una norma ISO.	
Entrada/Pasos de ejecución: Selecciona en la interfaz el botón evaluación. - Selecciona en la interfaz el botón Matriz de evaluación. - Se llenan los campos de la matriz de evaluación. - Luego el usuario selecciona el botón enviar.	
Resultado esperado: Se envía y procesa correctamente la matriz de evaluación.	
Evaluación de la prueba: Prueba satisfactoria	

Análisis de las pruebas de aceptación

En las pruebas de aceptación se realizaron cuatro iteraciones. Una vez implementadas las historias de usuario de la primera iteración de la implementación del sistema, se realizó la primera iteración de las pruebas de aceptación, se detectó en la misma 6 no conformidades, de estas cinco estaban relacionadas con la entrada de datos al sistema y la restante con la relación entre las clases Characteristic y Measure.

Después, una vez implementadas las historias de usuario de la segunda iteración y haber corregido las no conformidades detectadas en la iteración anterior, se desarrolló la segunda iteración de pruebas de aceptación detectándose tres no conformidades, de estas, dos relacionadas con el correcto funcionamiento de la funcionalidad reset_data y la restante con el correcto funcionamiento de la funcionalidad datos. Después, una vez implementadas las historias de usuario de la tercera iteración y haber corregido las no

conformidades detectadas en la iteración anterior, se desarrolló la tercera iteración de pruebas de aceptación detectándose nueve no conformidades, de estas, cinco relacionadas con el correcto funcionamiento de la funcionalidad mtz_eval y las otras cuatro con el correcto funcionamiento de la funcionalidad final_result.

Una vez corregidas las no conformidades de la tercera iteración de pruebas de aceptación se realizó la cuarta y última iteración de pruebas de aceptación no detectándose así ninguna no conformidad obteniéndose un resultado de satisfactorio. A continuación, se muestra una gráfica que contiene los resultados de las pruebas de aceptación (Ver figura 9).

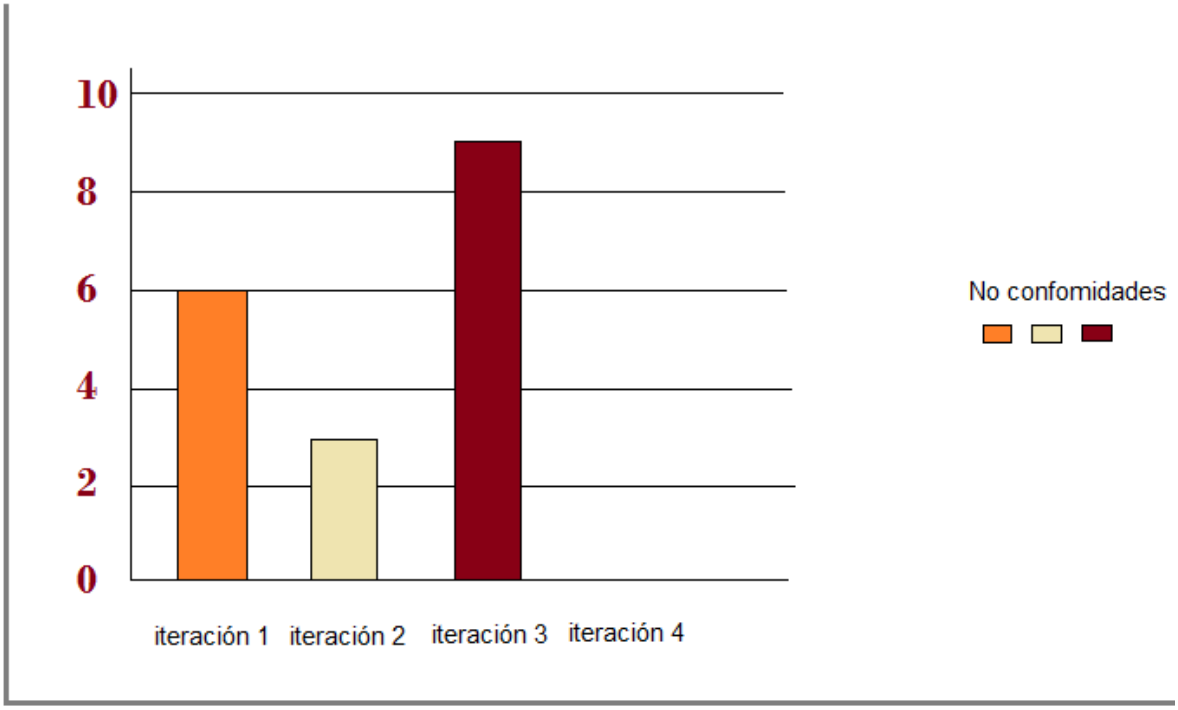


Figura 9: Gráfica de no conformidades detectadas en las pruebas de aceptación

Conclusiones parciales

En el presente capítulo se describen los elementos necesarios para completar el proceso de implementación y prueba del sistema SofQuality. Se describen las Tareas de Ingeniería, así como los estilos de programación utilizados en la implementación del sistema. También se realiza el análisis de los resultados arrojados por las pruebas de *software* realizadas a la aplicación llegando a las siguientes consideraciones:

- Con la definición de las 40 tareas de ingeniería se evitó la sobrecarga de trabajo en el equipo de desarrollo y se agilizó el proceso.

- La aplicación de las pruebas unitarias permitió validar las funcionalidades críticas del sistema, mientras que las pruebas de aceptación en conjunto con el cliente posibilitaron comprobar la correcta implementación de las historias de usuarios definidas con anterioridad.

- Con la realización de las pruebas de *software* se llegó a la conclusión que la implementación satisface los requerimientos definidos por el cliente.

- La solución propuesta brinda una mejoría en el proceso de toma de decisiones haciendo el mismo más eficiente y eficaz debido a que fueron informatizados todos los cálculos y los resultados de calidad son semejantes a los que reflejan los productos de *software* en explotación.

Conclusiones generales

A partir del desarrollo de la presente investigación se logra arribar a las siguientes conclusiones generales:

- La plataforma SoftQuality garantiza el uso práctico del modelo Evalprod en la industria del *software* y facilita la toma de decisiones.

- El uso de técnicas de *Soft Computing* como la lógica difusa, la modelación lingüística y los mapas cognitivos difusos permite la resolución de problemas en el proceso de selección y evaluación de productos de *software* y conduce a la obtención de resultados más precisos.

- El uso de un enfoque de desarrollo ágil con XP, así como de herramientas y tecnologías tales como el *framework* Django, el lenguaje de programación Python y el IDE de desarrollo Pycharm, facilitó el correcto desarrollo del sistema informático.

- Los requisitos del sistema, la arquitectura MTV y el uso de los patrones GRASP, permitieron obtener un diseño robusto, flexible y óptimo de la aplicación.

- Con la realización de las pruebas de *software* se demostró que la implementación satisface los requisitos definidos por el cliente.

Recomendaciones

Teniendo en cuenta los resultados obtenidos en esta investigación y basado en la experiencia adquirida, se recomienda:

- Realizar el tratamiento de la agregación total o parcial incluyendo el vector de penalización.

- Extender el uso de la aplicación a otros centros que realizan la evaluación de la calidad de *software*.

Referencias bibliográficas

“International Standard. ISO/IEC 9126-1 (2001) Quality Model”. 2001. *Institute of Electrical and Electronics Engineering.* 2001. p. 82.

B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. McLeod, and M. Merritt. 1978. *“Characteristics of Software Quality”.* North Holland : s.n., 1978.

Beck, Kent and Andres, Cynthia. 1999. *Extreme Programming Explained: Embrace Change.* EEUU : Addison-Wesley Professional, 1999. Vol. 2.

BR. Sintya Milena Meléndez Valladarez, BR. Maria Elizabeth Gaitan, BR. Neldin Noel Pérez Reyes. 2016. *METODOLOGIA ÁGIL DE DESARROLLO DE SOFTWARE PROGRAMACION EXTREMA.* Managua : s.n., 2016.

Cuccaro Goggi, Lucila Ana. 2010. *Adecuación de la metodología de desarrollo Extreme Programming a proyectos llevados a cabo en la materia Laboratorio III de la Facultad de Ingeniería de la Universidad Austral.* 2010.

David Pino Gonzáles, Tony Castillo Martin. 2016. *Sistema informático de ayuda a la decisión para la evaluación de la calidad de productos de software.* Universidad de las Ciencias Informáticas. La Habana : s.n., 2016.

Decision Making Helper. 2014. 2014.

DesdeLinux. 2018. DesdeLinux. [Online] 4 26, 2018. https://blog.desdelinux.net/pycharm-un-entorno-de-desarrollo-para-python/.

Django. 2018. Django. [Online] 2018. https://www.djangoproject.com/.

Django Software Corporation. 2015. *La guía definitiva de Django.* Celayita : s.n., 2015.

Dromey, R. G. 1995. *“A Model for Software Product Quality”.* 1995.

EcuRed. 2018. EcuRed. [Online] 2018. https://www.ecured.cu/Pycharm.

Fernández Pérez, Yamilis. 2018. *Modelo computacional para la evaluación y selección de productos de software.* Granada : s.n., 2018.

González, Juan Ramón González. 2008. *Herramientas de Soft Computing.* Granada : Editorial de la Universidad de Granada, 2008.

Guía web 2.0. 2016. Guía para el desarrollo de sitios web. [Online] 2016. http://www.guiadigital.gob.cl/guiaweb_old/guia-v2/glosario.html.

ISO 25000 calidad del producto de software. 2018. ISO 25000 calidad del producto de software. [Online] 2018. http://iso25000.com/index.php/normas-iso-25000/12-iso-iec-25050-25099.

ISO/IEC. 2000. *Information technology -Software product quality-.* s.l. : ISO, 2000.

J. A. McCall and J. P. Cavano. 1978. *"A framework for the measurement of software quality" .* 1978. pp. 133–139.

J.Pang, X.Liu and. 2010. *A Fuzzy Synthetic Evaluation Method for Software Quality.* 2010. pp. 1-4. Conf. E-bus. Inf. Syst. Secur..

López, Daniel Esteban Cano. 2012. *Sistema para evaluar la calidad de un producto o servicio Software.* Departameneto de Ingenniería de Sistemas Universidad EAFIT, Escuela de Ingeniería. Medelín : s.n., 2012.

Mejorando la gestión de historias de usuario en eXtreme Programming . **Sanchez , Emilio , Letelier Torres, Patricio and Canós Cerdá, Jose Hilario. 2003.** 2003, ISSI, pp. 6-8.

Mestras, Juan Pavón. 2009. *El patrón Modelo-Vista-Controlador (MVC).* Madrid : s.n., 2009.

Oficina Nacional de Normalización(NC),"NC ISO/IEC 25000. 2011. *Ingenería de software-Requisitos de la calidad y evaluación de productos software(SQuaRE)-Guía para SQuaRE.".* 2011.

Pérez Rodríguez, Fernando y Rojo Alboreca, Alberto. 2012. *MPC 2.0, software para la aplicación del método AHP de toma de decisiones multicriterio.* Minerva : Repositorio Institucional da Universidade de Santiago de Compostela:, 2012.

Pérez, Yamilis Fernández. 2018. *Modelo computacional para la evaluación y selección de productos de software.* Granada : s.n., 2018.

R. B. Grady and D. L. Caswell. 1987. *Software Metrics: Establishing a Company-Wide.* Prentice-Hall : s.n., 1987.

ResearchGate. 2018. [Online] 2018. https://www.researchgate.net/publication/323255768_Elicitacion_de_requisitos_no_funcionales_basada_en_la_gestion_de_conocimiento_de_los_stakeholders_Non-functional_requirements_elicitation_based_on_stakeholders%27s_knowledge_management.

Scalone, F. 2006. *"Estudio comparativo de los modelos y estandares de calidad del software".* Buenos Aires : s.n., 2006. Universidad Tecnológica Nacional. Facultad Regional.

Sistema gestor de bases de datos. 2013. Sistema gestor de bases de datos. [Online] 2 25, 2013. http://poppets001.blogspot.com/.

Stellman, Andrew and Greene, Jennifer. 2005. *Applied Software Project Management.* EEUU : O'Really Media, 2005.

Táramo, Yanet Peña. 2013. *Implementación del método TOPSIS en un sistema de ayuda en el proceso de Toma de Decisiones Multicriterio.* 2013.

W- ictea. 2018. W- ictea. [Online] ICTEA, 2018. http://www.ictea.com/cs/index.php?rp=/knowledgebase/3484/iQue-es-el-software-de-control-de-versiones-GIT.html.