Universidad de las Ciencias Informáticas

Facultad 3

Título: Herramienta para el análisis de conflictos entre atributos de calidad.

Trabajo de Diploma para optar por el título de
Ingeniero Informático.

Autor(es): Damarlys Menéndez Aponte

Tutor(es): Ing. Larisa González Álvarez

Ing. Sisley Jiménez Martínez

La Habana, Julio de 2012
"Año 54 de la Revolución"

"La arquitectura sólo se considera completa con la intervención del ser humano que la experimenta"

Tadao Ando

Declaración de autoría

Declaro ser el autor de la presente tesis y reconozco 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 _____.

Damarlys Menéndez Aponte

_____

(Firma del Autor)

Ing. Larisa González Álvarez

_____

(Firma del Tutor)

Ing. Sisley Jiménez Martínez

_____

(Firma del Co-tutor)

Datos del contacto

Tutor: Ing. Larisa Gonzales Álvarez

E-mail: lgalvarez@uci.cu

Graduada en el 2008 de de Ingeniero en Ciencias Informáticas en la Universidad de Ciencias Informáticas (UCI) con título de oro. Actualmente arquitecta principal de Cedrux.

Categoría Científica: Ingeniero.

Categoría Docente: Profesora instructora.

Co- tutor: Ing. Sisley Jiménez Martínez

E-mail: sjmartinez@uci.cu

Graduada en el 2011 de Ingeniero en Ciencias Informáticas en la Universidad de Ciencias Informáticas (UCI) con título de oro. Actualmente se desempeña como arquitecta de datos del proyecto Cedrux.

Categoría Científica: Ingeniero.

Categoría Docente: Adiestrado.

Agradecimientos

A mi abuela Bárbara que tanto amo y quiero, por haberme brindado todo su amor, cariño y dedicación cuando estuvo a mi lado, y aunque ya no esté conmigo físicamente siempre la llevaré en mi corazón por ser un ejemplo a seguir para mí.

A mis padres que tanto quiero por todo el cariño, amor y confianza que me han brindado toda mi vida.

A mi esposo Ángel que amo tanto, por ser la persona más importante que he conocido en mi vida, por estar a mi lado en los momentos bueno y malos, por brindarme su amor y apoyo incondicional y por hacerme feliz cada día.

A mi amigo Jose que quiero como un padre por apoyarme, quererme y ayudarme tanto en mi vida.

A mi abuelo Mariano que quiero mucho por quererme, aconsejarme y ayudarme en los momentos difíciles de mi vida.

A mi amiga Betty que tanto quiero, por brindarme su amistad incondicional y comprenderme en los momentos buenos y malos de mi vida.

A mis tutoras Larisa y Sisley, por sus consejos, ayuda y apoyo, los cuales fueron cruciales para la realización del presente trabajo.

A mi amigo Jose (Puro), por siempre estar y ayudarme cuando lo he necesitado.

A todos los que de una forma u otra me ayudaron en mi formación profesional y en la realización de este trabajo.

…… muchas gracias,

Damarlys

Resumen

En el proceso de evaluación arquitectónica del sistema integral de gestión Cedrux, se propone utilizar para el análisis de interrelaciones de atributos de calidad, un conjunto de mejoras realizadas al método QUASAR. En la aplicación manual de estas mejoras, los datos arrojados no son precisos, los cuales son fundamentales para tomar decisiones de tipo arquitectónico, debido a los complejos cálculos que se necesitan realizar. Este problema ha provocado atraso y no confiabilidad en dicho proceso evaluativo. Es por esto que surge la necesidad de desarrollar una herramienta para el análisis de interrelaciones de atributos de calidad que contribuya a la precisión de los datos para la toma de decisiones en la evaluación de la arquitectura de Cedrux.

Esta herramienta sustentará las mejoras propuestas para QUASAR, las cuales se basan en modelos matemáticos-estadísticos como la media aritmética y desviación estándar, y algoritmos clásicos de programación como el Vuelta Atrás[1]. Además se utilizan varias de las tecnologías actuales como el lenguaje de programación Java, los frameworks Spring, ExtJS e Hibernate y el SGBD[2] PostgreSQL, con el objetivo de que la herramienta que se desarrolle sea libre, multiplataforma, y se ejecute en un entorno web.

Con el desarrollo de la herramienta se mostrarán los datos precisos, que son fundamentales para que los arquitectos tomen sus decisiones. Se infiere además que el proceso evaluativo gane en rapidez, fiabilidad, y centralización de los datos, logrando así definir una sólida arquitectura y alcanzar todos los atributos de calidad requeridos para el sistema, factores fundamentales para lograr el éxito final del producto.

Palabras claves: proceso de evaluación, método de evaluación, datos precisos.

[1] Vuelta Atrás, del inglés *backtracking*: es una técnica empleada en algoritmos recursivos.

[2] SGBD: Sistema gestor de base de datos.

Índice

Introducción

La Universidad de las Ciencias Informáticas (UCI), se encuentra inmersa en el desarrollo de un sistema integral de gestión empresarial denominado Cedrux. En este sistema se tiene como premisa fundamental garantizar la calidad desde que se concibe el mismo, ya que todo lo que no se realice de manera eficaz va a incidir en el éxito del producto final. Un elemento importante para garantizar calidad, es definir una sólida línea base de arquitectura del software y dentro de ella como buena práctica de ingeniería se encuentra el proceso de realizar evaluaciones a las decisiones arquitectónicas, respecto al impacto que causarán sobre las características de calidad.

Para realizar el proceso de evaluación arquitectónica en Cedrux, se propone emplear el método de evaluación ***QUASAR***[3]***.*** Sin embargo según (Firesmith, 2006), este método presenta deficiencias que atentan contra su mejor uso, las cuales son:

1. No contempla la garantía de calidad arquitectónica entre múltiples subsistemas: la elaboración de los casos de calidad se realizan por subsistemas independientes, de manera que las metas globales de calidad y la contribución de cada subsistema a las mismas nunca se contempla.

2. No considera los conflictos de atributos de calidad: esto implica una brecha en el análisis de las decisiones arquitectónicas que tengan implicaciones en más de un atributo o subatributo de calidad.

En aras de eliminar estas deficiencias, se proponen un conjunto de mejoras según encuesta realizada que van a permitir medir la calidad integral de la arquitectura y tener en cuenta el impacto de decisiones arquitectónicas en múltiples indicadores. Estas mejoras se centran en dos aspectos fundamentales:

1. La priorización de los atributos de calidad y la eliminación de los conflictos entre atributos de calidad que se generan debido a esta priorización, tanto a nivel de subsistema como a nivel de sistema en general.

2. El análisis de los conflictos propios que se generan entre los atributos de calidad, debido a la estrecha relación que existe entre todos los atributos, en la cual favorecer uno puede redundar en satisfacción o detrimento de otro.

[3]QUASAR, del inglés Quality Assessment of Software-Intensive System Architectures: Método para la Valoración de Calidad de Arquitecturas de un Sistema Software.

Las mejoras propuestas se basan en métodos matemáticos-estadísticos, como por ejemplo la media aritmética y la desviación estándar, el primero es utilizado para poder priorizar todos los atributos de calidad de los subsistemas y el sistema en general y el segundo para definir cuál de los atributos con igual valor de media debe tener mayor prioridad. También se propone el empleo de algoritmos clásicos de programación, como por ejemplo un algoritmo basado en grafos utilizando Vuelta Atrás para realizar el análisis de interrelaciones de atributos de calidad, con el objetivo de lograr las prioridades de los atributos de calidad sin generar conflictos entre ellos.

Debido a la complejidad de los cálculos que estos métodos y algoritmos proponen, llevarlos a cabo de forma manual en el desarrollo de Cedrux acarrearía los siguientes problemas:

- Los resultados arrojados no serían precisos debido a que se pueden introducir errores, comprometiendo así la integridad de la información.
- Los datos necesarios para realizar el proceso no estarían centralizados, razón por la cual se correría el riesgo de una pérdida de información, infiriéndose que al finalizar el proceso no se obtengan los datos precisos.
- El proceso de evaluación para lograr que los datos arrojados sean precisos, sería tedioso y no confiable a los involucrados (arquitectos, evaluadores), debido a los extensos y complejos cálculos que se necesitan realizar.
- Atraso y en algunos casos fracaso en el desarrollo del proyecto, debido a que los datos obtenidos no son los precisos en el momento de tomar decisiones de tipo arquitectónico.

Debido a los problemas descritos anteriormente surge la necesidad de desarrollar una herramienta para realizar el análisis de interrelaciones de atributos de calidad, que contribuya a la precisión de los datos para la toma de decisiones arquitectónicas. En desarrollar dicha herramienta, se centra el presente trabajo.

Por todo lo anteriormente descrito se ha planteado el siguiente **problema a resolver**: la complejidad de aplicación del método para realizar el análisis de interrelaciones de atributos de calidad en la evaluación de Cedrux, afecta la precisión de los datos para la toma de decisiones.

Para dar solución al problema antes expuesto se ha definido como **objeto de estudio**: la evaluación de la arquitectura del software.

En vista de solucionar el problema planteado anteriormente se tiene como **objetivo general**: desarrollar una herramienta para el análisis de interrelaciones de atributos de calidad que contribuya a la precisión de los datos para la toma de decisiones en la evaluación de la arquitectura de Cedrux, del cual se obtiene como **campo de acción**: herramientas para la evaluación de arquitecturas de software.

El objetivo general antes expuesto para lograrse se divide en los siguientes **objetivos específicos**:

1. Realizar el marco teórico de la investigación para fundamentar los métodos y herramientas de soporte a la solución.
2. Implementar una herramienta informática que soporte el análisis de los conflictos de atributos de calidad, para que los datos sean los precisos en el momento de tomar decisiones en la evaluación arquitectónica de Cedrux.
3. Validar la herramienta.

Para el cumplimiento de los objetivos descritos anteriormente se propone la siguiente **idea a defender**: el desarrollo de una herramienta para el análisis de interrelaciones de atributos de calidad en la evaluación de la arquitectura de Cedrux, contribuirá a la precisión de los datos para la toma de decisiones.

Con el fin de solucionar el problema planteado y dar cumplimiento al objetivo propuesto se realizarán las siguientes **tareas de investigación**:

- Sintetizar el estudio del estado del arte en materia de evaluación de arquitectura de software y del desarrollo de aplicaciones para la automatización de la misma.
- Procesar y evaluar la información obtenida de la investigación del tema y adoptar una posición.
- Seleccionar la metodología y herramientas a utilizar para el desarrollo del producto.
- Desarrollar los requisitos de la aplicación.
- Diseñar la aplicación que soporte los requisitos.
- Implementar la solución propuesta.

- Realizar la validación de la propuesta.

Con el desempeño exitoso de las tareas anteriores se identifica el **posible resultado**: herramienta para el análisis de interrelaciones de atributos de calidad, que contribuirá a la precisión de los datos para la toma de decisiones en la evaluación de la arquitectura de Cedrux.

La investigación está sustentada en los siguientes **métodos científicos**:

Métodos Teóricos

Analítico-Sintético: para comprender a partir de las distintas fuentes bibliográficas consultadas, las características e importancia que se derivan del proceso de evaluación de la arquitectura que se le realiza a todo producto de software.

Histórico-Lógico: para el estudio y análisis de herramientas actuales que contribuyan al proceso de evaluación de la arquitectura, con el objetivo de comprender la esencia de estas herramientas y las causas que han dado origen a su comportamiento evolutivo.

Métodos Empíricos

Observación: para obtener el conocimiento necesario sobre la manera en que se comporta el proceso de evaluación de la arquitectura del software tal y como sucede en la realidad, además de ser una forma factible de obtener información directa e inmediata de dicho fenómeno.

Modelación: utilizado en el momento de diseñar la herramienta a desarrollar, pues se debe definir un modelo de datos que cumpla con los requisitos necesarios.

Experimentación: utilizado para la puesta en práctica de la herramienta a desarrollar y la obtención de los resultados, mediante pruebas y validaciones que se le realizan al sistema en el ambiente real de desarrollo de Cedrux.

Estructura del Documento

El presente trabajo está conformado por tres capítulos.

Capítulo 1. Fundamentación teórica de la investigación

Este capítulo constituye la fundamentación teórica del presente trabajo. Su lectura le ofrecerá información referente a los principales conceptos tratados. Además lo pondrá al tanto respecto al estado del arte en la evaluación de la arquitectura de software y por último se hace un estudio de las

metodologías y herramientas con el objetivo de seleccionar las idóneas para el desarrollo de la herramienta.

Capítulo 2: Modelado del sistema

En este capítulo se exponen los principales artefactos que se generan en la actividad de modelado realizada al sistema a desarrollar, como son: la descripción de los requisitos funcionales y no funcionales, definir la arquitectura que va a sustentar al sistema, los diagramas de clases del diseño, y el modelo de datos con el objetivo de tener una mejor comprensión, documentación del sistema y establecer las bases para la implementación del producto. Además se realiza la validación del diseño propuesto a través de las métricas TOC[4] y RC[5], con el objetivo de comprobar que el diseño definido tiene una buena calidad.

Capítulo 3: Implementación y validación de la solución propuesta

En este capítulo se exponen los principales artefactos que se generan en la actividad de implementación del sistema a desarrollar, como son: diagrama de componentes y diagrama de despliegue. Por último se realizan las pruebas de caja negra, así como la validación del sistema.

[4] TOC: Métrica relacionada con el Tamaño Operacional de la Clase.

[5] RC: Métrica relacionada con las Relaciones entre Clases.

Capítulo 1: Fundamentación teórica

1.1 Introducción

En este capítulo se explica porque es necesaria la calidad en productos de software y la estrecha relación que tiene la calidad con la arquitectura de software. Se abordan los conceptos básicos relacionados con el tema de evaluación de la arquitectura de software, así como la importancia de estas evaluaciones para lograr un producto con la calidad requerida. Se realiza un breve análisis sobre herramientas utilizadas en la actualidad para evaluar arquitecturas, y se describen las tecnologías y herramientas seleccionadas para el desarrollo de la herramienta propuesta.

1.2 Calidad de Software

La calidad de software debe tenerse en cuenta desde que se concibe el sistema, ya que todo lo que no se realice de manera eficaz va a incidir en la calidad del producto final. Un elemento importante para lograr calidad es definir una sólida línea base de la arquitectura, de ahí que esta se evalúe periódicamente para determinar la calidad que le atribuye a un sistema.

La arquitectura de software según (Camacho, 2004) "*se define como el "puente" entre los requerimientos del sistema y la implementación, sirve como medio de comunicación entre los miembros del equipo de desarrollo, los clientes y usuarios finales, dado que contempla los aspectos que interesan a cada uno"*. La arquitectura es considerada el centro de todo producto informático y determina cuáles serán los niveles de calidad asociados al sistema.

Por lo anterior (Vigil, 2009) plantea que garantizar la calidad del sistema depende en gran medida de garantizar la calidad en la arquitectura, y un elemento fundamental para lograr tal objetivo es realizar el proceso de evaluación arquitectónica. Más adelante y como ejemplo de lo anterior el mismo autor define que para alcanzar un atributo de calidad específico, es necesario tomar decisiones de diseño arquitectónico que requieren un pequeño conocimiento de funcionalidad. Se establece entonces que al considerar una decisión de arquitectura de software, el arquitecto se pregunta cuál será el impacto de esta sobre ciertos atributos y sobre el sistema en general. Es por esto que todo el proceso de concepción y diseño de la arquitectura incide de manera directa en la calidad de un sistema de software.

1.2.1 Atributos de Calidad

Los requerimientos de calidad son requerimientos adicionales del sistema que hacen referencia a características que este debe satisfacer, diferentes a los requerimientos funcionales. Se definen como las propiedades de un servicio que presta el sistema a sus usuarios (Barbacci et al. 1995).

A grandes rasgos, (Bass et al., 1998) establece una clasificación de los atributos de calidad en dos categorías:

1. **Observables vía ejecución:** los atributos que se determinan del comportamiento del sistema en tiempo de ejecución, por ejemplo: Funcionalidad, Confiabilidad, Eficiencia.
2. **No observables vía ejecución:** los atributos que se establecen durante el desarrollo del sistema, por ejemplo: Mantenibilidad, Portabilidad, Usabilidad.

Debido a que la arquitectura influye sobre ciertos atributos de calidad del sistema, es importante tener en cuenta que no puede lograrse la satisfacción de ciertos atributos de calidad de manera aislada. Alcanzar un atributo de calidad, puede tener efectos positivos o negativos sobre otros atributos que, de alguna manera, también se desean alcanzar.

En su mayoría, los atributos de calidad se pueden organizar y descomponer de maneras diferentes, en lo que se conoce como modelos de calidad.

1.3 Evaluación de la arquitectura

La importancia de evaluar la arquitectura de software surge de la estrecha relación que esta posee con la calidad del sistema. Teniendo en cuenta que el cumplimiento de los atributos de calidad, y el éxito en sí del producto, se obtienen de un buen diseño arquitectónico, es preciso entonces, tomar las medidas necesarias para garantizar que este diseño haya sido desarrollado de la manera correcta. Por tanto, la principal tarea que se debe llevar a cabo para asegurar que la arquitectura propuesta cumple con las necesidades del sistema, es evaluar dicha arquitectura.

Según (Salvador, 2007) *“el propósito de realizar evaluaciones a la arquitectura, es para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante ,verificar que los requerimientos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad.”*

La evaluación explícita de la arquitectura de los sistemas de software con respecto a los requerimientos de calidad, minimizará los riesgos de construir un sistema que falle y, consecuentemente, disminuirá el costo de desarrollo del sistema según (Bosch, 2001).

1.3.1 Técnicas de evaluación de la arquitectura

Cuando se habla de cómo evaluar siempre se hace referencia a las técnicas de evaluación como una categoría que determina la manera de medir el estado de una arquitectura. Las clasificaciones de las técnicas de evaluación de arquitectura se muestran en la **Figura 1**.

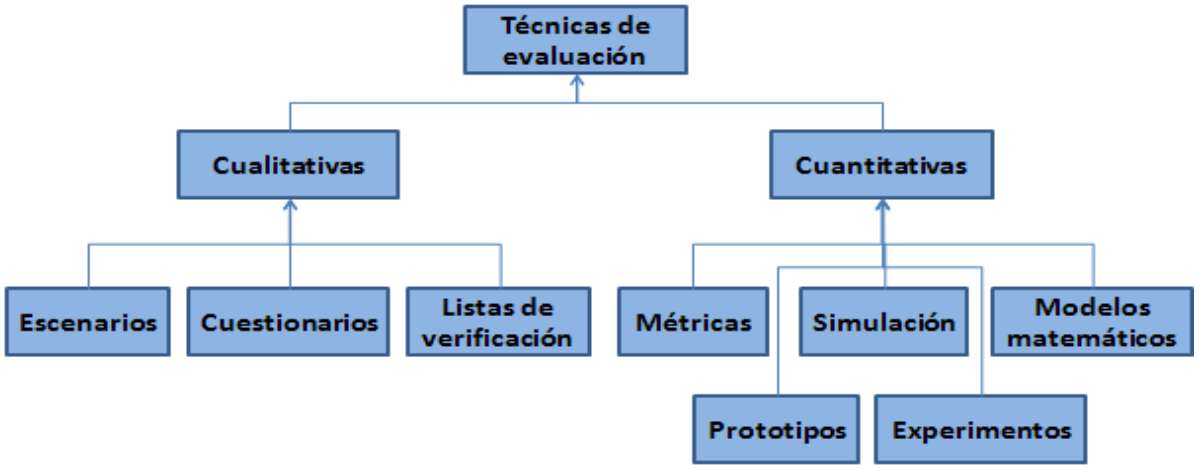

Figura 1. Clasificación de las técnicas de evaluación (Gómez, 2007).

De la **Figura 1** se puede observar que las técnicas de evaluación se dividen en dos grandes grupos, las técnicas cualitativas y las técnicas cuantitativas.

Técnicas cualitativas: permiten determinar estados de las arquitecturas en términos no contables, o sea, en valoraciones de las decisiones arquitectónicas o de los escenarios arquitectónicos e incluso de los atributos de calidad. Muchas veces resultan las más usadas por su aplicabilidad y su posibilidad de evaluar la mayor parte de los indicadores de calidad de un producto de software.

Técnicas cuantitativas: tienen más aceptación en cuanto a la obtención de valores, pero tienen en su contra la necesidad de herramientas que las soporten, porque en tiempo de producción se hace muy engorroso ponerlas en práctica de manera manual.

1.3.2 Métodos de evaluación de la arquitectura

Debido a la gran importancia y necesidad de evaluar la estructura de la arquitectura de software surgen los métodos de evaluación, para soportar los requisitos de software. Los métodos junto a las técnicas de evaluación constituyen el eslabón fundamental dentro del proceso de evaluación**.** El objetivo principal de estos métodos radica en identificar actividades (pasos y/o fases) y roles para la realización de la evaluación arquitectónica.

A continuación se realiza una breve descripción, del Método para la Valoración de Calidad de Arquitecturas de un Sistema Software (***QUASAR***), ya que para realizar el proceso de evaluación arquitectónica de Cedrux, se propone utilizar este método.

Método QUASAR

Este método es considerado más que un método de evaluación de la arquitectura, un método de aseguramiento de la calidad arquitectónica. Entre sus principales ventajas se encuentran:

1. Emplea varias técnicas para la evaluación de los indicadores de calidad.
2. Permite gradualmente la asimilación de mejoras y perfeccionamiento del proceso de evaluación, sin generar cambios drásticos visibles en la metodología de trabajo.
3. Utiliza el enfoque jerárquico de grandes sistemas de software para determinar los casos de calidad a evaluar en cada iteración.
4. Promueve la confección de un artefacto donde se consolida la información arquitectónicamente más significativa en términos de evaluación, posibilitando que el proceso de evaluación sea mucho más fácil para el equipo validador.

Sin embargo según (Firesmith, 2006), ***QUASAR*** presenta deficiencias, las cuales son:

1. No contempla la garantía de la calidad arquitectónica entre múltiples subsistemas: la elaboración de los casos de calidad se realiza por subsistemas independientes, de manera que las metas globales de calidad y la contribución de cada subsistema a las mismas nunca se contempla.
2. No considera los conflictos de atributos de calidad: esto implica una brecha en el análisis de las decisiones arquitectónicas que tengan implicaciones en más de un atributo o subatributo de calidad.

En aras de erradicar las deficiencias antes planteadas fueron propuestas un conjunto de mejoras, las cuales se centran en dos aspectos fundamentales:

1. La priorización de los atributos de calidad y la eliminación de los conflictos entre atributos de calidad que se generan debido a esta priorización, tanto a nivel de subsistema como a nivel de sistema en general, de manera que:

    - Posibilite la toma de decisiones arquitectónicas, para el logro del equilibrio de los factores de calidad, según su orden de prioridad.

    - Establezca un posible orden de prueba del cumplimiento de los atributos de calidad durante la evaluación, en función de prioridades.

2. El análisis de los conflictos propios que se generan entre los atributos de calidad, debido a la estrecha relación que existe entre todos los atributos, en el cual favorecer uno puede redundar en satisfacción o detrimento de otro.

Estas mejoras se sustentan de los siguientes métodos matemáticos y algoritmos clásicos de programación.

Métodos Matemáticos

Media Aritmética: empleada para establecer la prioridad de los atributos y subatributos tanto a nivel de subsistema como a nivel de sistema en general, mediante la siguiente fórmula:

$$\bar{X} = (\sum X_i) / N$$

Donde:

$\mathbf{X_i}$: *Valores de importancia asociados a cada atributo/subatributo.*
N: *Cantidad de requisitos asociados a cada atributo/subatributo.*

Desviación estándar: empleada para saber que atributos con igual valor de media deben tener mayor prioridad, en ese caso sería el que posea menor valor de desviación estándar. Teniendo en cuenta que la desviación estándar es igual a la raíz cuadrada de la varianza, se calcula entonces el valor de la última mediante la siguiente fórmula:

$$S^2 = \sum^n (X_i - M)^2 / N$$

Donde:

S^2: *Varianza.*
X_i: *Valor del atributo en cada caso.*
M: *Valor de la media del conjunto de valores asociados al subatributo/atributo.*
N: *Cantidad total de valores analizados.*

Algoritmos clásicos de programación

Para realizar el análisis de los conflictos propios que se generan entre los atributos de calidad, se propuso un algoritmo que encuentre variantes válidas en cuanto a decisiones arquitectónicas, que permitan lograr las prioridades de los indicadores de calidad, sin generar conflictos entre ellos.

Este algoritmo tiene como entradas las siguientes:

1. La matriz de interrelaciones según ISO 9126 -1.
2. Lista de prioridades de subatributos (mayor valor = mayor prioridad).

El objetivo de este algoritmo es obtener una matriz de interrelaciones, en la cual se logre alcanzar la mayor prioridad para cada atributo sin que se generen conflictos entre ellos.

Para lograr lo anterior, tomando la matriz de interrelaciones como punto de partida, se construye un grafo dirigido y ponderado que tiene tantos nodos como filas tiene la matriz. Las aristas del grafo y sus direcciones se construyen a partir de las filas, o sea, el origen lo determina la fila, y el fin la columna. Los pesos de cada camino se establecen en función de las formas de afectación que se valoran en la matriz, tal y como se muestra en la **Tabla 1.**

Valor en la matriz	Peso en el grafo
+	1
-	-1
Sin valor	0

Tabla 1. Transformación de los valores en las celdas de la matriz por pesos en el grafo.

Una vez construido el grafo se procede a calcular la importancia de cada nodo del grafo. La importancia de un nodo (atributo) se considera una combinación de la cantidad de atributos que afecta y que lo afectan y la importancia de estos, matemáticamente representada quedaría:

$$I_x = C_x * \sum_{i=1}^{C_x} R_i * P_i + A_x * \sum_{i=1}^{A_x} R_i * P_i$$

Donde:

I: *Importancia de un nodo.*
C_x: *Cantidad de nodos que afecta el nodo x.*
A_x: *Cantidad de nodos que afectan al nodo x.*
R: *Prioridad de un nodo.*
P: *Peso.*
x: *Nodo.*

Después de calculada cada importancia para cada nodo del grafo, se pasa a comprobar si el grafo puede ser un candidato a la solución, mediante el siguiente criterio de factibilidad.

$I_x > I_y \quad R_x < R_y$ cuando $I_x \neq 0$

Donde las **I** son las importancias de los nodos asociados a ese grafo y las **R** son las prioridades asociadas a dichos nodos.

Si el grafo es factible se calcula la desviación estándar de las importancias del mismo. La desviación estándar no es más que la raíz cuadrada de la varianza, cuya fórmula es:

$$S^2 = \sum^n (X_i - M)^2 / N$$

Donde:

X_i=*Valor de importancia de un nodo del grafo.*
M =*Media de los valores de importancia del grafo.*
N: *Cantidad de nodos del grafo.*

Después de calculada la desviación estándar se trata de insertar el grafo en una lista donde están los primeros 25 grafos factibles ordenados de menor a mayor valor de desviación estándar. Si el grafo posee menor desviación que alguno de los existentes en la lista, este se inserta en la misma y los demás elementos se corren desechándose siempre el último elemento de la lista. En el caso que el grafo que se quiera insertar no tenga la desviación estándar menor que ninguno de los 25 que conforman la lista, este se desecha. Para el caso en que el grafo que se quiera insertar tenga la desviación estándar igual a alguno de la lista, entonces se pasa a comparar la importancia del primer nodo y si esta es mayor, entonces se inserta el grafo, corriéndose los demás elementos de la lista. Al

finalizar el análisis de todos los grafos se pasa a identificar el caso óptimo, el cual será el grafo de la lista que tenga mayor valor de importancia en el primer nodo. Para el caso de que exista más de un grafo con igual mayor valor de importancia en el primer nodo, entonces el óptimo será el que tenga menor desviación estándar.

Hasta aquí se ha descrito el algoritmo para el grafo construido a partir de la matriz inicial, pero se hace necesario realizar los mismos pasos descritos anteriormente para todos los grafos que se puedan generar a partir del grafo inicial, donde siempre cada grafo que se genere tendrá menos aristas que el inicial. En la generación de estos grafos es donde se emplea un algoritmo Vuelta Atrás. La cantidad de grafos a generar a partir del inicial se puede calcular mediante la siguiente fórmula:

$$\sum_{r=0}^{r-n} \frac{n!}{r!(n-r)!}$$

Donde:

n: *Cantidad de aristas del grafo inicial.*

r: *Cantidad de aristas en cada caso restante.*

Después de expuestos todo los pasos que engloban los métodos matemáticos y algoritmos clásicos de programación, se puede concluir que la aplicación manual de los mismos trae como consecuencia que los datos arrojados no sean precisos en el momento de los arquitectos tomar sus decisiones, debido a los extensos y complejos cálculos que se necesitan realizar. Se infiere además que la demora, no confiabilidad y descentralización de los datos arrojados, traiga como consecuencia el atraso y en algunos casos fracaso en el proceso de evaluación arquitectónica del proyecto.

Por todo lo antes expuesto se propone desarrollar una herramienta que sustente los métodos y algoritmos en que se basan las mejoras realizadas al método QUASAR, con el objetivo de eliminar los problemas antes planteados. En desarrollar dicha herramienta, se centra el presente trabajo.

1.4 Herramientas para la evaluación de arquitecturas

El uso de herramientas para evaluar una arquitectura de software, es alentado para mejorar la toma de decisiones de tipo arquitectónico, factor decisivo para establecer una sólida línea base de la arquitectura.

Durante la investigación del presente trabajo se estudiaron dos herramientas que tienen como objetivo fundamental apoyar el proceso de evaluación de arquitecturas de software, estas son:

Ambiente de evaluación de la calidad de arquitecturas de software (AMECAS) y Aplicación para la automatización de la evaluación de la arquitectura de software en los proyectos productivos de la UCI. A continuación se explican las funcionalidades de estas herramientas.

Ambiente de Evaluación de la Calidad de Arquitecturas de Software (AMECAS): desarrollada en Venezuela en el año 2006. Consiste en un conjunto de plug-ins que se instalan en la plataforma Eclipse, facilitando la ejecución de las actividades asociadas a la aplicación de los métodos y técnicas de evaluación de arquitecturas. Tiene como objetivo brindar las siguientes funcionalidades: especificación de calidad de la arquitectura (específicamente mediante el modelo ISO/IEC 9126 instanciado para arquitecturas de software), especificación de la arquitectura (a través del ***ADL***[6] ACME), apoyar el proceso de evaluación de arquitecturas (específicamente con los métodos ***ATAM***[7] y ***ARID***[8]***)*** y documentar el proceso de evaluación de una arquitectura (para dejar constancia de la funcionalidad que espera el usuario y que dirigen las decisiones al momento de la construcción y evaluación de la arquitectura del sistema).

Aplicación para la automatización de la evaluación de la arquitectura de software en los proyectos productivos de la UCI: herramienta para la gestión de la información que se genera durante la evaluación de la arquitectura de software en los proyectos productivos de la UCI. Tiene como objetivo organizar y centralizar la información que se maneja durante las evaluaciones arquitectónicas, tratando con ello de simplificar el procedimiento evaluativo.

En la **Tabla 2** se realiza una comparación entre estas herramientas teniendo en cuenta los indicadores fundamentales que debe contemplar la herramienta que de soporte a las mejoras propuestas.

Indicadores	AMECAS	Aplicación UCI
Gratuito	Si	Si

[6] ADL, del inglés *Architecture Descrition Language*: Lenguaje de descripción de arquitecturas. "*Es una notación estándar que permite representar arquitecturas de software, permite analizar y especificar una arquitectura*"(Bass et al, 1998).

[7] ATAM, del inglés *ArchitectureTrade offs Analisys Method*: Método de Análisis de Acuerdos de Arquitectura. Es uno de los métodos de evaluación de arquitecturas de software existente en la actualidad.

[8] ARID, del inglés *Active Reviews for Intermediate Designs*: Revisión Activa para el Diseño Intermedio. Es uno de los métodos de evaluación de arquitecturas de software existente en la actualidad.

Multiplataforma	Si	Si
Modelos de calidad	ISO/IEC 9126	ISO/IEC 9126
Métodos de evaluación	ATAM, ARID	ninguno
Técnicas de evaluación cuantitativa	No, se basa en escenario	ninguna
Priorización de atributos de calidad	Si	No
Análisis de interrelaciones entre atributos de calidad	Si	No

Tabla 2. Diferencias entre las herramientas AMECAS y Aplicación UCI.

Luego de analizar la **Tabla 2** se llega a la conclusión que AMECAS es la que más se acerca a los indicadores que debe contemplar la herramienta que de soporte a las mejoras realizadas al método ***QUASAR***. Esta herramienta cumple con las funcionalidades principales que las mejoras proponen, las cuales son: Priorizar los atributos de calidad y Realizar el análisis de interrelaciones de atributos de calidad. Sin embargo en AMECAS, estas funcionalidades se basan en técnicas cualitativas de escenarios y las funcionalidades que proponen las mejoras son sustentadas por técnicas cuantitativas, debido al uso de modelos matemáticos-estadísticos como la media aritmética y desviación estándar, además de utilizar algoritmos clásicos de programación como el Vuelta Atrás. Por este motivo se decide realizar una nueva herramienta que se sustente de técnicas cuantitativas, para poder brindar las funcionalidades necesarias que proponen estas mejoras.

Se decide por parte del cliente que la herramienta a desarrollar sea una aplicación web debido a las ventajas que ofrece este tipo de aplicaciones con respecto a las aplicaciones de escritorio. Para el desarrollo de la misma se utilizan algunas herramientas y tecnologías actuales que exponen tendencias de la programación web y que en su mayoría son libres y de código abierto.

1.5 Metodologías y herramientas

Metodologías

”*Las metodologías de desarrollo de software surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental en el momento de desarrollar un producto de software”* según (Carrillo, y otros, 2008). Según la propia fuente existen dos grandes grupos de metodologías de software, los cuales son:

Metodologías robustas: centradas especialmente en el control del proceso, establecen rigurosamente las actividades involucradas, los artefactos que se deben producir, las herramientas y notaciones que se usarán.

Metodologías ágiles: orientadas a la interacción con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Son efectivas en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

Se decide utilizar una metodología ágil en el desarrollo de la solución propuesta, debido a que son las que más se acercan a las características que presenta el proceso de desarrollo de la herramienta que se va a realizar. Estas características son:

1. Equipo de desarrollo de una persona, la cual desempeñará diferentes roles.
2. Pueden ocurrir cambios en los requisitos en cualquier etapa del desarrollo del producto.
3. Tiempo de desarrollo de 6 meses.
4. Los artefactos que se pretenden generar para guiar el proceso de desarrollo son pocos.
5. Es un producto de software que no se rige por un contrato prefijado.

En la actualidad existen varias metodologías ágiles capaces de adaptarse a las características antes expuestas, entre las cuales se destacan ***AUP***[9] y ***XP***[10], por su fácil y rápida implementación en dicho proceso de desarrollo. Debido a que cualquiera de estas dos metodologías puede ser utilizada en el desarrollo de la solución propuesta, en la **Tabla 3** se realiza una comparación teniendo en cuenta los indicadores más significativos para el equipo de desarrollo, con el objetivo de seleccionar una de ellas:

Indicadores	Metodología XP	Metodología AUP

[9] AUP, del inglés *Agile Unified Process*: Proceso unificado ágil.

[10] XP, del inglés *eXtreme Programming*: Programación extrema.

Principios	Retroalimentación Simplicidad Cambios incrementales Aceptar el cambio Aceptar la responsabilidad.	Simplicidad Agilidad Centra actividades de alto valor Adaptación del producto para satisfacer las necesidades
Fases	Exploración Planificación Iteraciones por entregas Producción Mantenimiento Muerte	Inicio Elaboración Construcción Transición
Actividades	Codificar Hacer pruebas Escuchar Diseñar	Modelo Implementación Despliegue Prueba
Artefactos	Historia de usuarios Diagrama de clases	Especificación de requisitos Especificación de la arquitectura Diagrama de diseño Modelo de datos Modelo de componentes Modelo de despliegue Diseño de caso de prueba

Tabla 3. Diferencias entre las metodologías ágiles XP y AUP.

Luego de analizar estas metodologías ágiles en la **Tabla 3** se decide utilizar la metodología ***AUP*** en el desarrollo de la herramienta propuesta, por los siguientes motivos:

1. Define exactamente los principios y fases por las que debe regirse el proceso de desarrollo.
2. Cumple con todas las actividades necesarias a realizar por parte del equipo de desarrollo.
3. Genera los artefactos que permiten obtener la documentación necesaria para una mejor comprensión de la herramienta a desarrollar.

Herramientas de Ingeniería de Software asistidas por computadoras (CASE)

Las herramientas CASE, no son más que aplicaciones informáticas que persiguen incrementar la productividad y calidad durante el desarrollo de un software mediante la reducción de tiempo, esfuerzo y dinero. Estas automatizan el dibujo de diagramas y ayudan en la creación de relaciones en la base de datos, provocando que el trabajo de diseño del software sea más fácil y agradable. El

principal objetivo de estas herramientas es proporcionar un lenguaje para describir el sistema general, que sea lo suficientemente explícito para generar todos los programas necesarios.

Visual Paradigm

En la actualidad existen varias herramientas CASE capaces de soportar todo el ciclo de desarrollo de la herramienta del presente trabajo. Sin embargo, se decide utilizar la herramienta Visual Paradigm, por ser una herramienta multiplataforma. Además, a través de sus diagramas ayuda a una más rápida construcción de aplicaciones de calidad con un menor costo. Permite realizar ingeniería directa e inversa, es decir, obtener el código a partir de diagramas, así como diagramas a partir de un código previamente escrito.

Lenguaje de programación

Un lenguaje de programación es un lenguaje artificial que intenta parecerse al lenguaje humano, utilizado para definir una secuencia de instrucciones capaces de ser interpretadas y ejecutadas por una computadora. Este lenguaje establece un grupo de reglas sintácticas y semánticas, las cuales rigen la estructura del programa de computación que se escribe.

JAVA

En la actualidad existen varios lenguajes de programación que se pueden utilizar en el desarrollo de la herramienta del presente trabajo. Sin embargo, se decide utilizar el lenguaje de programación Java, debido a que es un lenguaje orientado a objetos, multiplataforma, y presenta varias de las características que el equipo de desarrollo necesita que tenga el lenguaje seleccionado, las cuales son:

1. Sencillo y familiar.
2. Robusto y Seguro.
3. Alto rendimiento, Multi-hilo.

Entorno de Desarrollo Integrado (IDE)

Un Entorno de Desarrollo Integrado ***IDE***[11] es una aplicación informática compuesta por un conjunto de herramientas de programación, encargadas de facilitar el trabajo de los desarrolladores de

[11] IDE, del inglés *Integrate Development Enviroment*: Entorno de Desarrollo Integrado.

software a la hora de llevar a cabo una aplicación. Permite dedicarse a un único lenguaje de programación o hacer uso de varios de ellos.

Eclipse

Entre los diferentes ***IDE*** existentes en la actualidad se decide utilizar Eclipse, debido a que al igual que otros creado para escribir, compilar, depurar y ejecutar programas escritos en Java, aunque puede servir para otros lenguajes de programación. Es un proyecto de código abierto y gratuito, con una gran base de usuarios. Posee una comunidad en constante crecimiento y es además multiplataforma.

Frameworks

Un frameworks es la representación de una arquitectura de software que se adapta a las particularidades de un determinado dominio de aplicación, suministrando una estructura y una metodología de trabajo que va a utilizar las aplicaciones de dicho dominio. Normalmente incluye soporte de programas, bibliotecas y un lenguaje interpretado, entre otros programas para ayudar a desarrollar y unificar los diferentes componentes de un proyecto.

Spring Framework

En la actualidad existen varios frameworks capaces de ofrecer respuesta a los diferentes problemas que se pueden encontrar a la hora de desarrollar una aplicación web usando el lenguaje de programación java. Sin embargo, se decide utilizar Spring Framework, debido a que el mismo es un framework gratuito y de código abierto. Implementa el patrón ***MVC***[12] y tiene un módulo ***ORM***[13] que provee una forma conveniente para construir la capa de acceso a datos. Además el costo de utilizar Spring en términos de rendimiento, tanto para la aplicación que se realiza como para el sistema y hardware que lo soporta es despreciable, debido a que su consumo es mínimo.

ExtJS Framework

En la actualidad existen varios frameworks Javascript para el desarrollo de aplicaciones web. Sin embargo, se decide utilizar ExtJS porque permite la implementación de interfaces visuales muy

[12] MVC, del inglés *Model View Controller*: Modelo-Vista-Controlador: Es un patrón de arquitectura utilizado en sistemas web para separar los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos.

[13] ORM, del inglés *Object Relational Mapping*: Persistencia automática y transparente de objetos de una aplicación en una base de datos relacional utilizando meta datos que describen la correspondencia entre el objeto y las tablas de la base de datos.

potentes, debido a la rica colección de componentes para el diseño de interfaces del lado del cliente. Tiene incluido la mayoría de los controles de formulario web incluyendo tablas para mostrar datos y elementos semejantes a la programación desktop, como los formularios, paneles, barras de herramientas, menús y otros. Dentro de su librería de componentes incluye componentes para el manejo de datos, lectura de ***XML***, lectura de datos ***JSON*** e implementaciones basadas en ***AJAX***. Presenta el uso de Javascript como una programación orientada a objetos. Además posee un grupo de características que fueron necesarias en el momento de definir un framework, para realizar el diseño de la interfaz de la herramienta del presente trabajo, las cuales son:

- ✓ Alto rendimiento.
- ✓ Interfaces personalizables.
- ✓ Muy buena arquitectura que permite extender los componentes gracias a su buen modelado.
- ✓ API fácil de utilizar.
- ✓ Licencia comercial y gratis.

Hibernate

En la actualidad existen varios frameworks ***ORM*** capaces de solucionar el problema de la diferencia entre dos modelos utilizados para organizar y manipular datos: el orientado a objetos en las aplicaciones y el relacional en las bases de datos. Sin embargo se decide utilizar el framework Hibernate, debido a que es gratuito y de código abierto, cuenta con amplia documentación y aceptación dentro de la comunidad Java. Además entre sus características principales se encuentran:

1 Soporta el paradigma de orientación a objetos de una manera natural: herencia, polimorfismo, composición y la librería de colecciones de java.

2 Provee un sistema de cache de dos niveles y puede ser usado en un clúster. Permite inicialización perezosa (lazy) de objetos y colecciones.

3 Presenta un potente mecanismo de transacciones.

4 Proporciona el lenguaje HQL el cual provee una independencia del SQL de la base de datos, tanto para el almacenamiento de objetos como para su recuperación.

Sistema Gestor de Base de Datos (SGBD)

Un Sistema Gestor de Bases de Datos (*Database Management System*, SGBD), es un tipo de software que sirve de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan, que pretenden manejar de manera clara, sencilla y ordenada un conjunto de datos. Además es el responsable de tratar todas las peticiones de información de los usuarios, permiten definir, construir y mantener una base de datos, asegurando su integridad, confiabilidad y seguridad. "*Es un sistema computarizado para guardar registros; es decir que su finalidad general es almacenar información y permitir a los usuarios recuperar y actualizar esa información*" según (Date, J.C, 2003).

PostgreSQL

Entre los diferentes Sistema Gestor de Base de Datos existentes en la actualidad se decide utilizar PostgreSQL, ya que las características que presenta este SGBD son totalmente compatibles con las que necesita el equipo de desarrollo para la realización de la herramienta, debido a que es un gestor de base de datos relacional, multiplataforma, libre y gratuito. Además entre sus características fundamentales de gran beneficio para la herramienta a desarrollar se encuentran las siguientes, según (López, y otros, 2011):

Atomicidad: asegura la realización de una operación, por lo que ante un fallo del sistema esta no queda a medias.

Consistencia: posibilita la ejecución de aquellas operaciones, que no van a romper las reglas y directrices de integridad de la base de datos.

Aislamiento: mediante un sistema de acceso concurrente multiversión (***MVCC***[14]), asegura que una operación no pueda afectar a otras, de esta manera dos transacciones sobre la misma información no genera error.

Servidor Web

Un servidor Web es un programa que implementa el protocolo HTTP, con el propósito de transferir hipertextos, páginas Web o páginas HTML, figuras, formularios, entre otro conjunto de objetos y animaciones. Este se mantiene atento a las peticiones realizadas por el cliente, conocido como navegador Web y las responde adecuadamente. En dependencia del tipo de petición, el servidor

[14] MVVC, del inglés *Multiversion Concurrency Control*.

Web busca una página Web o bien ejecuta un programa en el servidor. De cualquier manera, siempre devuelve algún tipo de resultado HTML al cliente o navegador que realizó la petición.

Apache Tomcat

En la actualidad existen varios servidores Web capaces de soportar la ejecución de aplicaciones Web realizadas con Java. Sin embargo se decide utilizar Apache Tomcat, debido a que es un servidor gratuito, ligero, multiplataforma, de fácil instalación, configuración y uso y es compatible con las ***API*** más recientes de Java. Además de ser un servidor web muy estable y confiable, debido a la amplia documentación que posee y a los miles de desarrolladores que contribuyen con código.

Navegador Web

Un ***navegador web*** o explorador web, es un programa (aplicación de software), por lo general gratuito que permite visualizar páginas web a través de Internet o en el propio ordenador. Además posibilita acceder a otros recursos de información alojados también en servidores web, como pueden ser videos, imágenes, audio y archivos XML(Garriazo, Luna y otros, 2011).

Mozilla Firefox 11.0

Entre los diversos navegadores web existentes en la actualidad se decide utilizar Mozilla Firefox, debido a que es un navegador multiplataforma, gratuito y de código abierto. Entre las características fundamentales de este navegador web tenemos que incorpora bloqueo de ventanas emergentes, navegación por pestañas, marcadores dinámicos, compatibilidad con estándares abiertos, mecanismo para añadir funciones mediante extensiones, corrector ortográfico y búsqueda integrada (Montesino Afonso y otros, 2008).

1.6 Conclusiones parciales

Tras el estudio realizado en este capítulo se puede arribar a las siguientes conclusiones:

- La evaluación de una arquitectura de software, es un elemento fundamental para lograr una sólida línea base de la arquitectura que garantice la calidad en todo producto de software.
- En el proceso de evaluación de arquitectura de software de Cedrux, se proponen un conjunto de mejoras realizadas con el objetivo de eliminar las deficiencias del método QUASAR.
- La aplicación manual de las mejoras propuestas, no arrojan los resultados precisos en el momento de tomar decisiones de tipo arquitectónico, debido a los extensos y complejos cálculos que se necesitan realizar.
- Se comprobó que las herramientas estudiadas para la evaluación de arquitecturas de software, no satisfacen todos los indicadores que se necesitan para cumplir con lo que propone las mejoras realizadas al método QUASAR.
- Se propuso la realización de una nueva herramienta que cumpla con todas las funcionalidades que propone las mejoras realizadas al QUASAR, con el objetivo de obtener los datos precisos que contribuyan a la toma de decisiones arquitectónica.

Para la realización de la nueva herramienta, se definió la metodología y herramientas a utilizar las cuales son las siguientes:

1. Como metodología ágil de desarrollo de software se decide utilizar AUP, debido a su factibilidad y características expuestas.
2. Como herramienta CASE se decide utilizar Visual Paradigm Enterprise Edition, fundamental para el modelado y construcción de la herramienta que se desea implementar.
3. Como lenguaje de programación se decide utilizar Java, debido a que es multi-hilo, característica fundamental para ejecutar de forma rápida y eficiente grandes y complejos algoritmos recursivos, como por ejemplo el backtracking, el cual constituye el algoritmo base para el desarrollo de la herramienta propuesta.

4. Como entorno de desarrollo integrado ***IDE*** se decide utilizar Eclipse, para implementar la propuesta de solución por su flexibilidad y potencia.

5. Como frameworks se deciden utilizar Spring, debido a que permite crear aplicaciones web implementadas en Java con una mayor rapidez y seguridad, ExtJS debido a que permite la implementación de interfaces visuales muy potentes e Hibernate, para facilitar toda la interacción con el sistema gestor de base de datos.

6. Como sistema gestor de base de datos (SGBD) se decide utilizar PostgreSQL, debido a que posee un alto rendimiento y procesamiento de los datos.

7. Como servidor web se decide utilizar Apache Tomcat, debido a que es multiplataforma, gratuito, y es compatible con las ***API*** más recientes del lenguaje seleccionado.

8. Como navegador web se decide utilizar Mozilla Firefox, debido a que es multiplataforma, libre y soporta todos los estándares de programación web.

Capítulo 2: Modelado del sistema

2.1 Introducción

Este capítulo se centra en la actividad de modelado del sistema de la herramienta a desarrollar basado en la metodología ágil de desarrollo de software AUP. Se describen los principales artefactos generados en dicha actividad, con el objetivo de lograr una mayor documentación y comprensión del sistema a desarrollar, facilitando el proceso de implementación de la misma. Además se realiza la validación del diseño propuesto con el objetivo de comprobar que el diseño realizado tiene una buena calidad.

2.2 Requerimientos del sistema

En la fase de elaboración de la metodología AUP, se define como primera actividad de modelado profundizar en los requerimientos que debe tener el sistema a desarrollar. Esta actividad permite asegurar que el equipo realmente puede implementar un sistema que cumpla los requisitos del cliente en vista de solucionar su problema o conseguir su objetivo. Una vez definidos los requisitos del sistema se genera un artefacto llamado especificación de requisitos, donde se realiza una descripción textual de los mismos con el objetivo de entender la funcionalidad asociada a cada requisito. La confección de este artefacto fue entregado al cliente y con el objetivo de que los usuarios que consulten el presente trabajo tengan conocimiento de lo que este artefacto engloba, la especificación del requisito: Gestionar Requisito no funcionales para el sistema se encuentra en el **¡Error! No se encuentra el origen de la referencia.** del presente trabajo.

Es importante tener en cuenta que para el proceso de captura de los requerimientos, fue utilizado por parte del equipo de desarrollo del presente trabajo la siguiente técnica:

Entrevistas: le permiten al analista tomar conocimiento del problema y comprender los objetivos de la solución buscada (M.J. Escalona, y otros 2008). En el desarrollo de la solución propuesta se hicieron entrevistas con el cliente que es la persona que más sabe sobre el tema que está indagando, con el objetivo de obtener un mejor esclarecimiento y comprensión de todos los requerimientos que debe contemplar la herramienta a desarrollar, logrando así el éxito final del producto y por consecuente la satisfacción del cliente.

Requisitos Funcionales

Una vez realizada la entrevista con el cliente se esclarecieron y capturaron todos los requerimientos funcionales y no funcionales que la herramienta debe contemplar. Los requisitos funcionales fueron agrupados por paquetes como se muestra en la **Figura 2**, con el objetivo de tener una mejor comprensión del sistema logrando así una mejor organización y cómodo manejo.

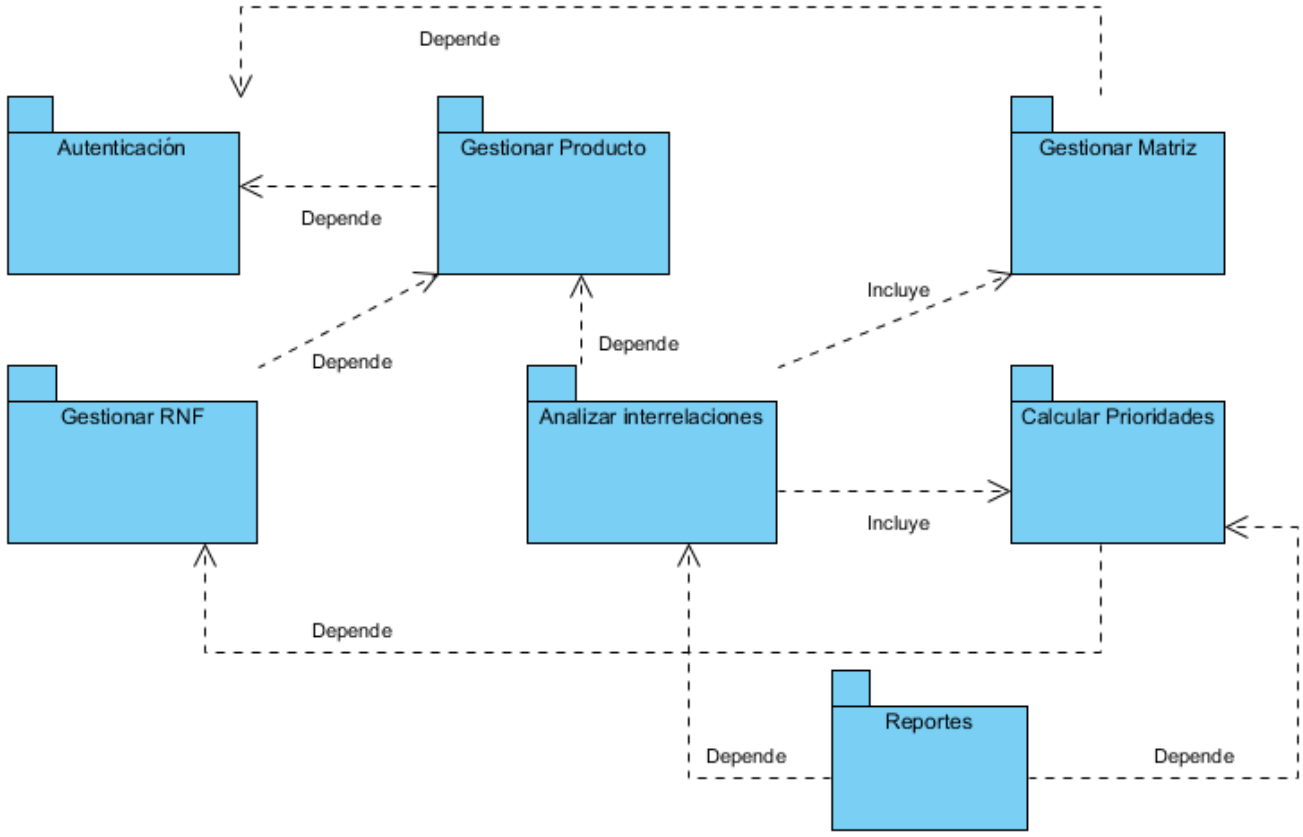

Figura 2. Diagrama de paquetes de requisitos

Del análisis de la **Figura 2** podemos constatar que el sistema cuenta con un total de 7 paquetes de requisitos. Se puede observar que estos paquetes de requisitos se encuentran relacionados a través de saetas que muestran el flujo del comportamiento de los requerimientos en el sistema y las dependencias que existen entre estos. A continuación se describen cada uno de los paquetes de requisitos, con el objetivo de obtener un mejor conocimiento acerca de los requerimientos que engloban cada uno de estos paquetes y las dependencias que existen entre estos.

Paquete Autenticación: se basa en el requisito Autenticarse. Como se puede apreciar en la **Figura 2** de este paquete dependen las funcionalidades generales del sistema: Gestionar producto y Gestionar matriz, debido a que para acceder a estas dos funcionalidades, se debe Autenticar en el sistema.

Paquete Gestionar Producto: engloba los requisitos: Gestionar sistema y Gestionar subsistema. Como se puede apreciar en la **Figura 2***,* este paquete tiene total dependencia con el paquete *Autenticación*, debido a que para acceder a las funcionalidades de Gestionar Producto se debe Autenticar el usuario.

Paquete Gestionar Matriz: se basa en el requisito Gestionar matriz de interrelaciones de atributos de calidad. Este paquete constituye una de las dos entradas fundamentales para realizar el paquete de requisitos arquitectónicamente más significativo para el negocio: *Analizar Interrelaciones*. Como se puede apreciar en la **Figura 2***,* este paquete tiene total dependencia con el paquete *Autenticación*, debido a que para acceder al requisito: Gestionar matriz de interrelaciones de atributos de calidad se debe Autenticar el usuario.

Paquete Gestionar RNF: se basa en el requisito Gestionar requisitos no funcionales de subatributos de calidad para un subsistema y sistema en general. Este paquete de requisito es la base del sistema a desarrollar, debido a que de estos se derivan los valores utilizados para realizar los cálculos internos en el sistema, con el fin de darle respuesta a las funcionalidades que engloba el paquete *Calcular Prioridades*. Como se puede apreciar en la **Figura 2***,* este paquete tiene total dependencia con el paquete *Gestionar Producto*, debido a que para acceder a la funcionalidad que engloba el paquete *Gestionar RNF,* se debe haber accedido primeramente a las funcionalidades que engloba el paquete *Gestionar Producto*.

Paquete Calcular Prioridades: engloba los requisitos: Realizar priorización para un subsistema y Realizar priorización para un sistema en general. Estas funcionalidades son las que muestran al usuario los datos que permiten obtener un orden de prioridad de los atributos de calidad asociados a un sistema o subsistema. Como se puede apreciar en la **Figura 2***,* el paquete tiene total dependencia con el paquete *Gestionar RNF*, debido a que para realizar la funcionalidad del cálculo de prioridades se debe haber realizado las funcionalidades que engloba este paquete para el(los) subsistema(s) o sistema(s). Además este paquete constituye la otra entrada fundamental para realizar el paquete de requisitos *Analizar Interrelaciones*, el cual es el arquitectónicamente más significativo para el negocio.

Paquete Analizar Interrelaciones: se basa en el requisito: Realizar análisis de interrelaciones de atributos de calidad. Este paquete de requisito es el arquitectónicamente más significativo para el negocio, debido a que es la funcionalidad principal de la herramienta. Como se puede apreciar en la **Figura 2***,* el paquete tiene total dependencia con el paquete *Gestionar Producto*, debido a que antes de poder acceder a la funcionalidad en que se basa el paquete *Analizar Interrelaciones*, se debe haber realizado primeramente las funcionalidades que engloba el paquete *Gestionar Producto*. Además se puede apreciar también en la **Figura 2***,* que el paquete *Analizar Interrelaciones* incluye a los paquetes *Calcular Prioridades* y *Gestionar Matriz*, debido a que las funcionalidades que engloban estos dos paquetes son las entradas fundamentales y necesarias que se requieren para poder realizar la funcionalidad en que se basa el paquete *Analizar Interrelaciones*.

Paquete Reportes: engloba los requisitos: Mostrar resultado de la priorización de atributos de calidad para un subsistema, Mostrar resultado de la priorización de atributos de calidad para un sistema y Mostrar resultado del análisis de interrelaciones de atributos de calidad. Estos requisitos son los que muestran al usuario los resultados arrojados por los cálculos internos del sistema en dependencia de las funcionalidades requeridas por el usuario. Como se puede apreciar en la **Figura 2***,* el paquete tiene total dependencia con el paquete *Calcular Prioridades* y *Analizar Interrelaciones*, debido a que se deben realizar todos los cálculos internamente en el sistema para poder mostrar los resultados al usuario.

Requisitos no Funcionales

A continuación se enumeran los requisitos no funcionales que el sistema a desarrollar debe cumplir:

Usabilidad

RNF1: La herramienta podrá ser usada por cualquier persona que posea conocimientos básicos de computación y trabajo en la web, brindando facilidades que permitan interactuar y navegar con el sistema de forma fácil y rápida.

Apariencia o interfaz externa

RNF 2: La herramienta debe poseer una interfaz amigable para el usuario, basada en web, que combine los colores, tipo de letra y tamaño. No debe contener muchas imágenes ni animaciones para evitar demora en las respuestas a los usuarios. La navegabilidad debe ser sencilla, permitiendo el reconocimiento visual de las funcionalidades.

Rendimiento

RNF 3: La herramienta debe ser capaz de gestionar la información y dar respuesta a las solicitudes lo más rápido posible.

Seguridad

RNF 4: Se podrá acceder a las funcionalidades de la herramienta después de autenticarse.

Soporte

RNF 5: La herramienta contará antes de su puesta en marcha con un período de pruebas.

Software

Cliente:

RNF 6: Sistema operativo con interfaz gráfica y soporte para red.

RNF 7: Las interfaces deben ser compatibles con Mozilla Firefox 11.0 ó superior.

Servidor:

RNF 8: Sistema operativo Ubuntu Server 8.04 o superior, o Windows XP.

RNF 9: Servidor web Apache Tomcat 5.5 o superior.

RNF 10: Gestor de base de datos PostgreSQL 8.3 o superior.

Hardware

Cliente:

RNF 11: Requerimientos mínimos: Procesador Pentium IV a 800 MHz, 128mb de memoria RAM y un navegador web.

Servidor:

RNF 12: Requerimientos mínimos: Procesador Dual Core a 3.00 GHz, 1gb de memoria RAM y una capacidad de 40gb de disco duro.

Patrones utilizados en la definición de los requisitos

Para determinar las funcionalidades de la herramienta se hizo necesario utilizar patrones para hacer mucho más fácil y organizado el tratamiento a los requerimientos, estos son:

El nombre revela la intención: los requisitos fueron nombrados teniendo en cuenta que su nombre revelará su intención. Para ello se decidió que la primera palabra fuese un verbo y la segunda un

sustantivo relacionado con la operación asociada al requisito, lo que permite que al leer su nombre se determine el propósito del mismo.

Completar una única meta: a medida que se identificaban los requisitos se verificó que los mismos estuvieran dirigidos hacia una completa y bien definida meta.

CRUD (Creating, Reading, Updating, Deleting): muy utilizado en los requisitos que necesitan gestión, los cuales son: Matriz, Sistema, Subsistema y Requisito No Funcional de subatributos de calidad.

Técnicas de validación de requisitos

En aras de demostrar que la definición de los requerimientos antes expuestos detallan el sistema que necesita o desea el cliente, se hace necesario por parte del equipo de desarrollo del presente trabajo, validar estos requerimientos a través de la siguiente técnica de validación:

Matriz de trazabilidad: consiste en marcar los objetivos del sistema y chequearlos contra los requisitos del mismo. Es necesario ir viendo que objetivos cumple cada requisito, de esta forma se podrán detectar inconsistencias u objetivos no cubiertos.

Para tener un mejor esclarecimiento acerca de los objetivos del sistema, así como de los requisitos funcionales definidos para el logro de dichos objetivos, se muestra la matriz de trazabilidad en la **Tabla 4**.

Objetivos del sistema	PR Usuario	PR Gestionar Producto	PR Gestionar RNF	PR Gestionar Matriz	PR Calcular Prioridades	PR Analizar Interrelaciones	PR Reportes
Seguridad del sistema.	X						
Permitir el trabajo con el sistema y subsistemas.		X					
Permitir el trabajo con los requisitos no funcionales de atributos de calidad (RNFAC) de un			X				

sistema o subsistema.							
Permitir establecer un peso para cada (RNFAC).			X				
Permitir el cálculo para priorizar los (RNFAC) para un subsistema y sistema en general.					X		
Permitir el trabajo con la matriz de interrelaciones de atributos de calidad				X			
Permitir el cálculo para el análisis de interrelaciones de atributos de calidad para un sistema en general.						X	
Mostrar resultados de la priorización de subsistema ó sistema en general y análisis de interrelaciones de atributos de calidad.							X

Tabla 4. Matriz de trazabilidad.

Como se puede apreciar en la **Tabla 4** los requisitos definidos por parte del equipo de desarrollo son los necesarios para cumplir con todos los objetivos del sistema.

2.3 Arquitectura del sistema

En la fase de elaboración de la metodología AUP, se propone como siguiente actividad de modelado definir la arquitectura del sistema que se va a desarrollar. La arquitectura es "*un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción de un software, define, de manera abstracta, los componentes que llevan a cabo alguna tarea, sus interfaces y la comunicación entre ellos, y además establece los fundamentos para que analistas, diseñadores y programadores trabajen en una línea común que permita alcanzar los objetivos del sistema, cubriendo así todas las necesidades*" según (Telmo, Pérez y otros, 2010). La arquitectura seleccionada, debe brindar la posibilidad de escribir código de calidad que funcione y que cumpla con los requerimientos definidos anteriormente.

Arquitectura en Capas

Uno de los objetivos del cliente y del equipo de desarrollo, es que la herramienta sea desarrollada teniendo en cuenta que en un futuro sea posible añadirle o modificarles funcionalidades. Por este motivo, se decide utilizar el estilo arquitectónico de llamada y retorno, el cual hace énfasis en la modificabilidad, adaptabilidad y escalabilidad. Dentro de este estilo se selecciona la arquitectura en capas, debido a que presenta una serie de características que le permiten al equipo de desarrollo construir una aplicación de forma rápida, segura y organizada, las cuales son:

1. Abstracción total acerca del origen de datos.
2. Bajo costo de desarrollo y mantenimiento de las aplicaciones.
3. Mejor calidad en las aplicaciones.
4. Reutilización de código.

En la arquitectura basada en capas, cada capa tiene funcionalidades y objetivos bien precisos, así la implementación de una capa se encuentra desacoplada de la programación de cualquier otra y la comunicación con una capa inferior se realiza a través de interfaces. Esta característica asegura que el acoplamiento sea el más bajo posible y la abstracción del funcionamiento de la capa inferior sea casi total.

La distribución de las capas y la interacción entre ellas se muestran en la **Figura 3**.

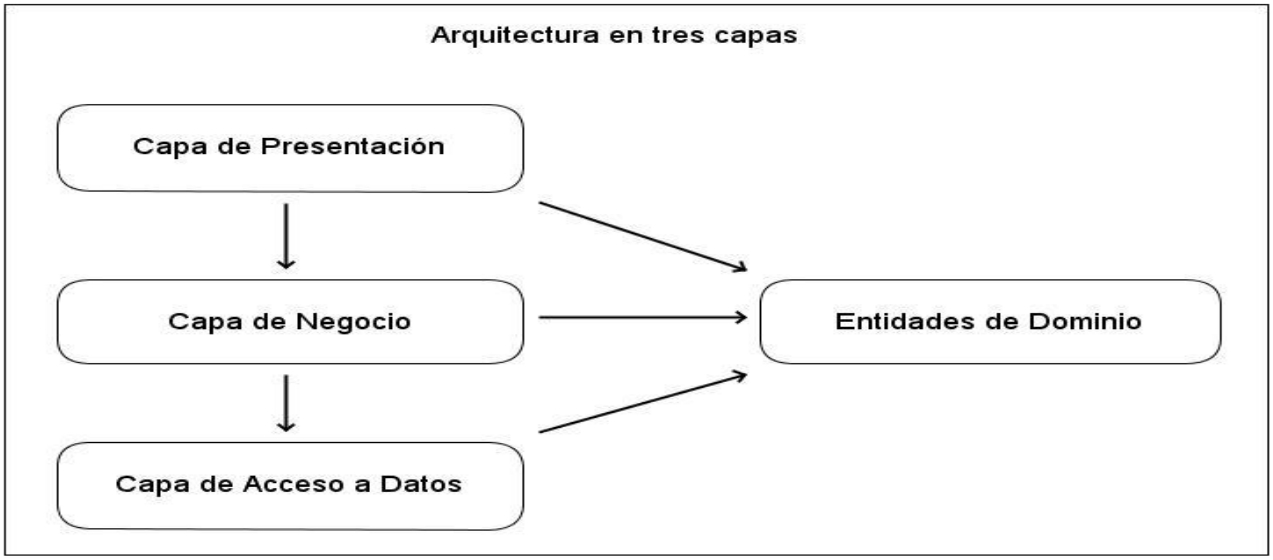

Figura 3. Arquitectura en capas.

En la **Figura 3** podemos constatar las capas por las que estará compuesta la aplicación, donde las saetas indican la dirección de la dependencia entre estas y se muestra la relación de todas las capas con las entidades del dominio.

A continuación se realiza una descripción de cada una de las capas por las que estará compuesta la aplicación.

Capa de Presentación

La capa de presentación define e implementa todo lo relacionado con la interfaz gráfica de usuario. Aquí residen la definición de las peticiones que el usuario puede realizar sobre la aplicación, los controladores que manejan el flujo web y la comunicación con la interfaz de la capa de negocio. Además se encuentran las vistas HTML y JPG que se muestran al usuario y la implementación del comportamiento dinámico de los documentos HTML a través de Javascript.

Las principales responsabilidades de esta capa son: la navegabilidad del sistema, el formateo de los datos de salida, la validación de los datos de entrada y la construcción de la interfaz gráfica de usuario.

Capa de Negocio

La capa de negocio define e implementa las funcionalidades que responden directamente a los requisitos, de manera que se conserve la integridad del sistema y de los datos. Los métodos expuestos por la Fachada[15] de esta capa son la única puerta de entrada posible al sistema, por lo que deben ofrecer todas las funcionalidades requeridas por este. Esta capa se comunica con la capa de acceso a datos a través de sus interfaces, para interactuar con el gestor, almacenar o recuperar de este.

Capa de Acceso a Datos

La capa de acceso a datos es la responsable de recobrar y persistir información desde y hacia la base de datos y de la comunicación con el gestor de base de datos. Esta capa contiene los objetos que encapsulan la lógica de acceso a datos e interfaces brindadas, para ser accedida desde la capa de negocio.

Entidades de Dominio

Son las clases persistentes y no persistentes que van a ser manejadas por todas las capas y que constituyen la representación estática de la información en el sistema informático.

Una vez identificados los requisitos que debe cumplir el sistema y definida la arquitectura del mismo, quedan sentadas las bases para realizar el diseño del sistema, con el objetivo de detallar y tener una mejor comprensión de la solución propuesta.

2.3 Diagrama de clases de diseño

En la fase de elaboración de la metodología AUP, una vez definida la arquitectura a utilizar se tiene como siguiente actividad de modelado, la realización de los diagramas de clases del diseño (DCD) teniendo en cuenta la arquitectura definida. Estos diagramas tienen el objetivo de describir gráficamente las relaciones entre clases e interfaces del sistema y proveer una abstracción de la implementación del sistema. La información contenida en estos diagramas será la siguiente:

1. Clases, asociaciones y atributos.
2. Interfaces, con sus operaciones y constantes.
3. Métodos.
4. Información sobre los tipos de los atributos.

[15] Fachada, del inglés Facade.

5. Dependencias.

A continuación se muestran los diagramas de clases del diseño de las funcionalidades fundamentales que se necesitan realizar, para que el sistema cumpla con el paquete de requisitos arquitectónicamente más significativo para el negocio *Analizar Interrelaciones*. Estas funcionalidades son: Gestionar requisitos no funcionales de subatributos de calidad, Gestionar matriz de interrelaciones de atributos de calidad, Priorizar atributos de calidad para un sistema y Realizar análisis de interrelaciones de atributos de calidad.

El diagrama de clases del diseño de la funcionalidad: Gestionar requisitos no funcionales de subatributos de calidad, se muestra en la **Figura 4.**

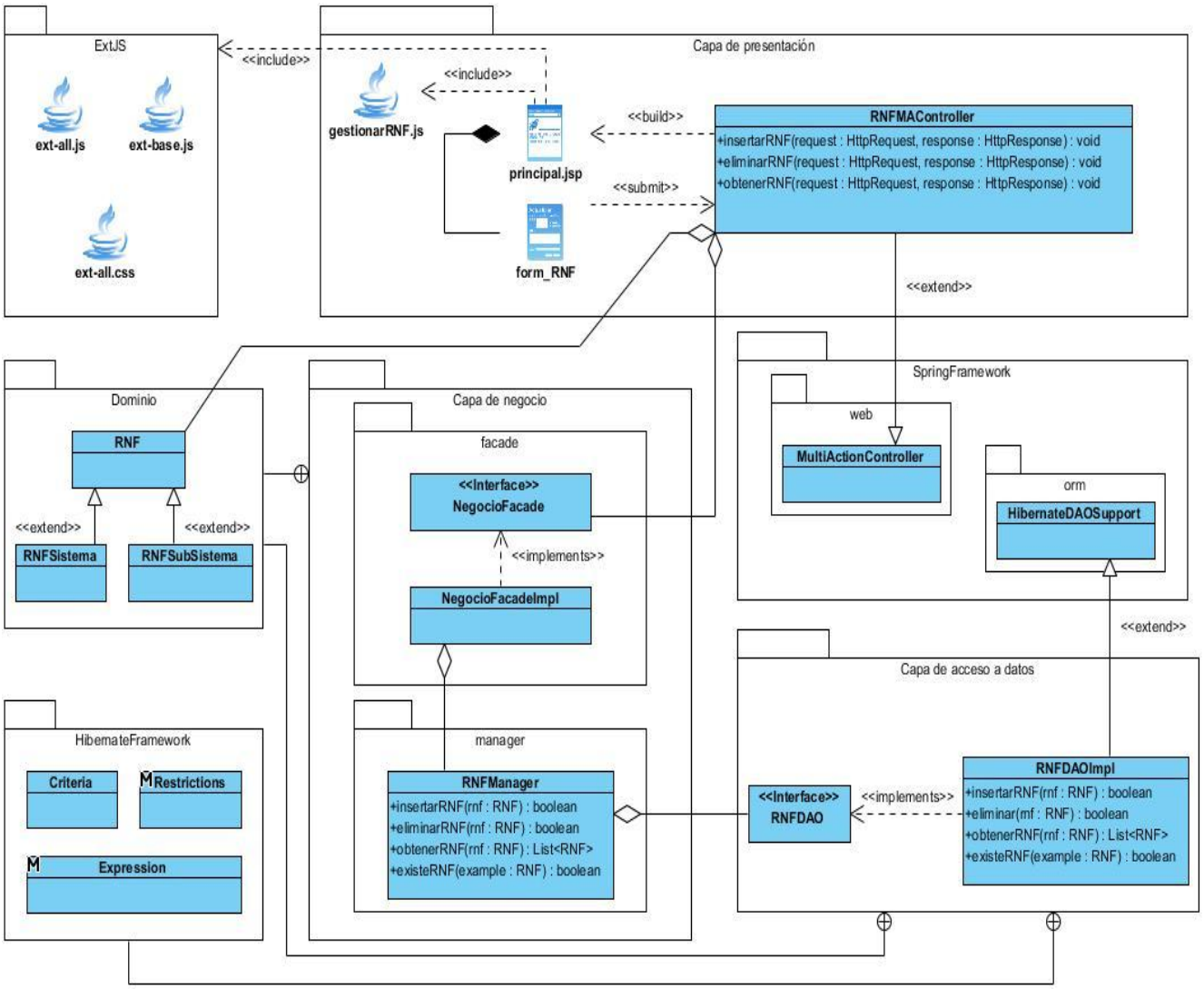

Figura 4. DCD: Gestionar requisitos no funcionales de subatributos de calidad.

En el diagrama de clases del diseño de la **Figura 4** se muestran las clases que intervienen en la implementación de la funcionalidad: Gestionar requisitos no funcionales (RNF) de subatributos de calidad. Se puede ver como se relacionan las clases y como las mismas se encuentran organizadas de acuerdo a la arquitectura utilizada. En la capa de presentación se tiene la página principal.jsp, la cual mostrará toda la información asociada a la gestión de RNF. Esta pagina principal.jsp utiliza el fichero Javascript gestionarRNF.js, que a su vez utiliza clases del framework ExtJS para darle funcionalidad a esta página en el cliente y contiene un formulario que se usa para enviar hacia el servidor los nuevos datos que se entren. Todas las peticiones que se generen de esta página serán manejadas por la clase controladora RNFMAController, la cual hereda de la clase MultiActionController del módulo web del framework Spring. La clase controladora accede a la capa de negocio mediante la interfaz NegocioFacade. En el diseño de la capa de presentación se hace uso del patrón Controlador, el cual propone que se use una clase para que maneje todas las peticiones provenientes de una página web. En la capa de negocio se encuentra la interfaz NegocioFacade, la cual es el único punto de entrada a esta capa. Esta interfaz no contiene ninguna lógica de negocio porque es usada para mostrar todas las funcionalidades de dicha capa, por lo que la verdadera implementación del negocio recae en las clases manejadoras, en este caso RNFManager. Esta clase manejadora interactúa con la capa de acceso a datos mediante la interfaz RNFDAO. En esta capa se hace uso del patrón Fachada, el cual posibilita un bajo acoplamiento entre la capa de presentación y la capa de negocio. Por último se encuentran las clases que intervienen en la capa de acceso a datos, las cuales se diseñaron teniendo en cuenta el patrón ***DAO***[16]***,*** en este caso RNFDAOImpl, la cual hereda de la clase HibernateDAOSupport, del módulo ***ORM*** del framework Spring y a la vez utiliza varias clases del framework Hibernate. El patrón ***DAO*** posibilita que se creen clases que sean las encargadas de implementar toda la comunicación con la fuente de datos. Por último se puede ver como todas las clases utilizadas para darle respuesta a esta funcionalidad, utilizan las clases que forman una jerarquía de herencia: RNF, RNFSistema y RNFSubSistema, pertenecientes al dominio de la aplicación.

El diagrama de clases del diseño de la funcionalidad: Gestionar matriz de atributos de calidad, se muestra en la **Figura 5.**

[16] DAO, del inglés *Data Access Object*: Objeto de acceso a datos.

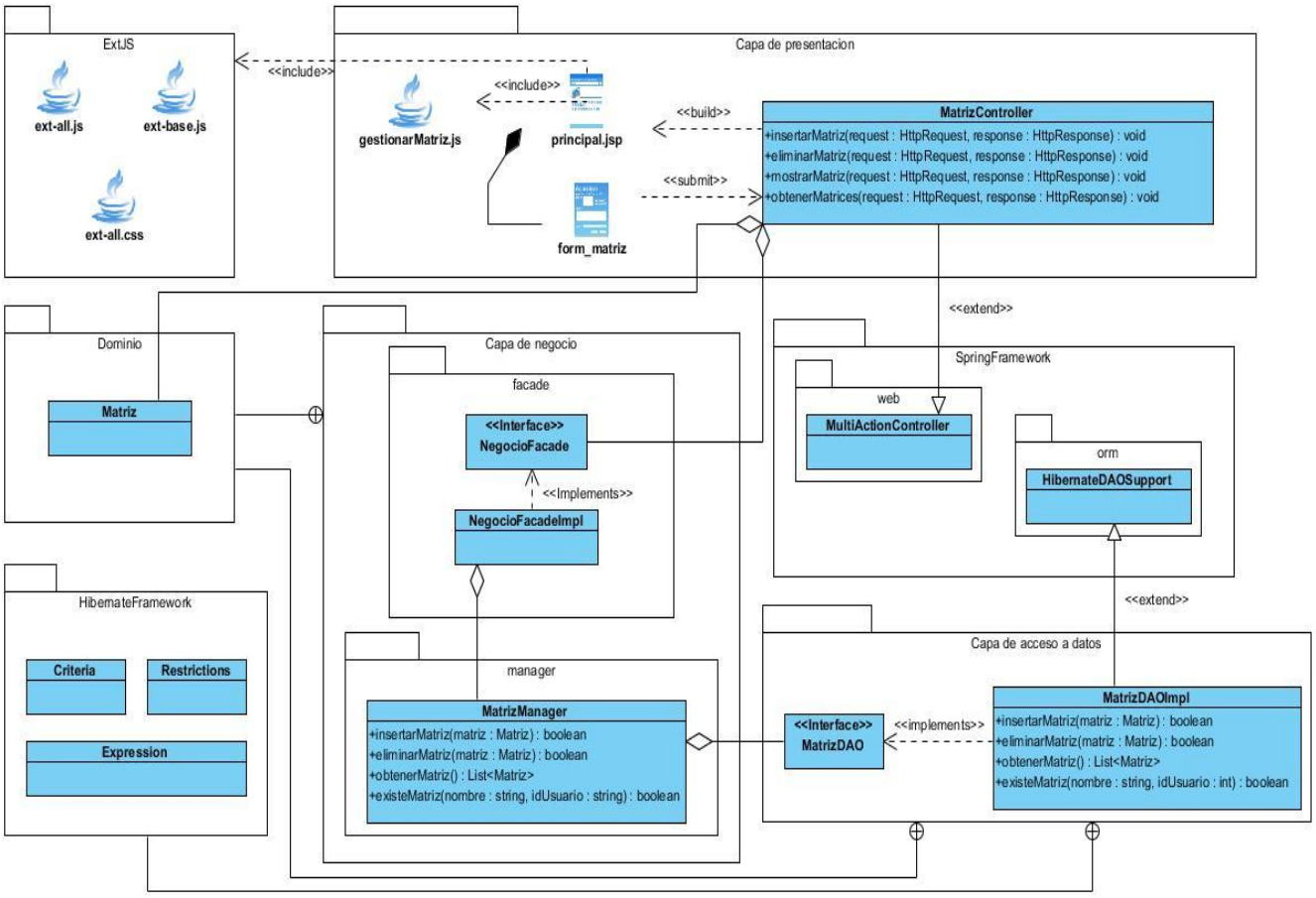

Figura 5.DCD: Gestionar matriz de interrelaciones de atributos de calidad.

En el diagrama de clases del diseño de la **Figura 5** las relaciones entre clases y el uso de patrones, se mantienen igual que en la **Figura 4**, pero las responsabilidades de cada clase varían. Por este motivo solo se realizará una descripción de las clases que intervienen en la implementación de la funcionalidad: Gestionar matriz de interrelaciones de atributos de calidad. En la capa de presentación de este diagrama se encuentra la página principal.jsp, la cual contiene un formulario y utiliza el fichero gestionarMatriz.js que a su vez utiliza clases del framework ExtJS. Para darle funcionalidad a esta página en el cliente, todas las peticiones que se generen de esta página serán manejadas por la clase controladora MatrizMAController. Esta clase controladora accede a la capa de negocio, mediante la interfaz NegocioFacade. En la capa de negocio se encuentra la clase NegocioFacadeImpl, la cual implementa la interfaz NegocioFacade. Además se encuentra en esta capa también la clase MatrizManager, la cual se relaciona con la capa de acceso a datos, mediante la interfaz MatrizDAO que es implementada por la clase MatrizDAOImpl. Por último se puede ver como todas las clases utilizadas para darle respuesta a esta funcionalidad utilizan la clase Matriz, perteneciente al dominio de la aplicación.

El diagrama de clases del diseño de la funcionalidad: Realizar priorización para el sistema y Realizar análisis de interrelaciones de atributos de calidad, se muestra en la **Figura 6**.

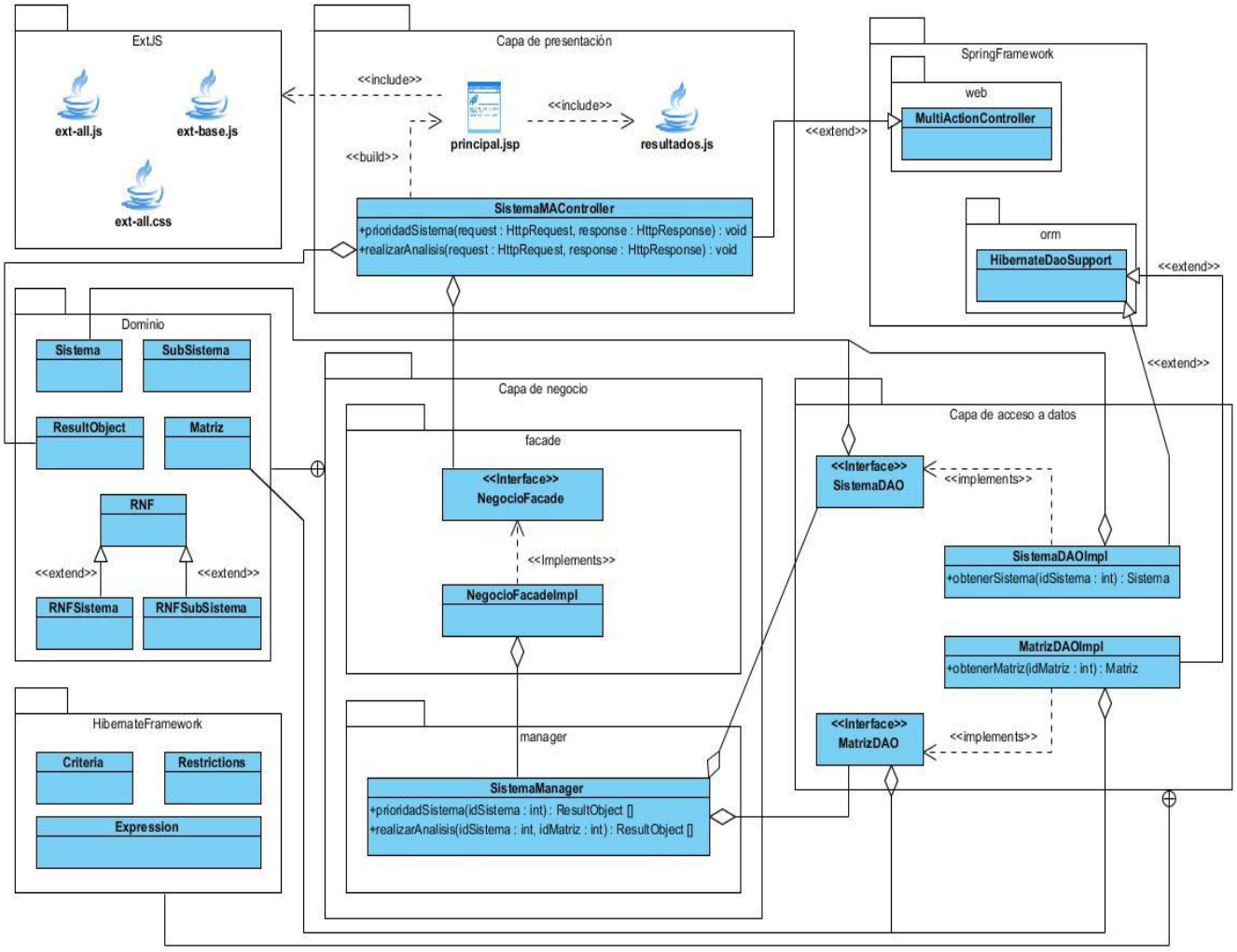

Figura 6. DCD: Priorizar Sistema y Realizar análisis de interrelaciones de atributos de calidad.

En el diagrama de clases de diseño de la **Figura 6** las relaciones entre clases y el uso de patrones, se mantienen igual que en la **Figura 4** y la **Figura 5**, pero las responsabilidades de cada clase varían. Por este motivo solo se realizará una descripción de las clases que intervienen en la implementación de la funcionalidades: Priorizar atributos de calidad para un sistema y Realizar análisis de interrelaciones de atributos de calidad. Este diagrama incluye estas dos funcionalidades, debido a que se realizó teniendo en cuenta que estos requisitos están asociados a un sistema. En la capa de presentación se tiene la página principal.jsp, la cual utiliza el fichero resultados.js que a su vez utiliza clases del framework ExtJS, para formatear los datos que serán mostrados por esta

página. La clase que se encarga de enviar los datos a la página principal.jsp es la clase SistemaMAController, la cual hereda de la clase MultiActionController perteneciente al módulo web del framework Spring. Esta clase controladora se comunica con la capa de negocio, mediante la interfaz NegocioFacade, la cual delega la responsabilidad en la clase SistemaManager por ser la principal clase asociada a estas dos funcionalidades. Esta clase es la más importante para el sistema, debido a que en esta se encuentra la implementación de la funcionalidad arquitectónicamente más significativa para el negocio: Realizar análisis de interrelaciones de atributos de calidad. Se muestra la relación de la clase SistemaManager con las interfaces SistemaDAO y MatrizDAO de la capa de acceso a datos, para poder obtener los datos necesarios de la base datos. Como se puede observar las clases que implementan las interfaces SistemaDAO y MatrizDAO, heredan de la clase HibernateDaoSupport, perteneciente al módulo ***ORM*** del framework Spring y utilizan varias clases del framework Hibernate. Por último se ve cómo se utilizan las entidades del dominio en todas las capas de la aplicación.

Patrones de diseño de software

En la realización de los diagramas de clase de diseño se usan varios patrones de diseño de software. Los patrones se definen como "*una solución probada para un problema general de diseño en un contexto determinado. Encierran la experiencia que programadores e ingenieros han adquirido en la solución de problemas comunes. Los patrones de diseño pueden incrementar o disminuir la capacidad de comprensión de un diseño o de una implementación, disminuirla al añadir accesos indirectos o aumentar la cantidad de código, incrementarla al regular la modularidad, separar mejor los conceptos y simplificar la descripción. Benefician la documentación y el mantenimiento de los sistemas pues proveen una especificación de los objetos y clases y sus relaciones, así como el objetivo del diseño" según (Suárez, 2009).*

Los patrones utilizados en el diseño de la herramienta a desarrollar son:

Controller (Controlador): utilizado en la capa de presentación del sistema a desarrollar. Propone asignar la responsabilidad de controlar el flujo de eventos de una aplicación a clases específicas llamadas controladores. Los controladores no ejecutan las tareas, sino que las delegan en otras clases, con las que mantiene un modelo de alta cohesión.

Facade (Fachada): utilizado en la capa de negocio del sistema a desarrollar. Simplifica el acceso a un conjunto de objetos proporcionando uno, llamado NegocioFacade, el cual la capa de presentación

utiliza para comunicarse con el negocio. Proporciona un mejor acoplamiento y facilita el cambio de componentes sin afectar a la capa de presentación.

DAO (Objeto de acceso a datos): utilizado en la capa de acceso a datos. Centraliza todo el acceso a datos en una capa independiente aislándolo del resto de la aplicación. Reduce la complejidad de los objetos de negocio al abstraerlos de la implementación con la fuente de datos.

Alta Cohesión: resuelve el problema de cómo lograr que la complejidad sea lo más manejable posible. Se aplica de manera general en el diseño de toda la aplicación, al agrupar las clases en dependencia de los requerimientos y los procesos que se necesitan automatizar. De este modo cada clase implementa las operaciones que se encuentran en su misma área funcional. Un ejemplo de esto, es la clase SistemaManager que solamente se encarga de la adición, modificación y eliminación de sistemas, así como otras operaciones que se realizan sobre esta área y no de otro tipo de operación.

Bajo Acoplamiento: resuelve el problema de cómo lograr baja dependencia, alta reutilización entre las clases y reducir el impacto de los cambios. Este patrón se evidencia entre la capa de presentación y la capa de negocio porque para acceder a las funcionalidades que brinda el negocio desde la presentación se utiliza solamente la interfaz NegocioFacade, logrando así una baja dependencia del negocio y un mínimo impacto en la capa de presentación con respecto a los cambios en la capa de negocio.

2.4 Diseño de la base de datos

En la fase de elaboración de la metodología AUP, una vez realizados los diagramas de clases del diseño se pasa a diseñar la base de datos, con el objetivo de que la herramienta pueda gestionar toda la información que es necesaria almacenar.

Normalización

Para evitar anomalías de actualización en las estructuras de los datos se efectuó por parte del equipo de desarrollo, el proceso de normalización en el momento de realizar el modelo lógico de la base de datos. La normalización es un estándar que consiste básicamente, en un proceso de conversión de las relaciones entre entidades que evita según (Caselles, 2008):

La redundancia de los datos: repetición de los datos en un sistema.

Anomalías de actualización: inconsistencias de los datos como resultado de datos redundantes y actualizaciones parciales.

Anomalías de borrado: pérdida no intencionada de datos, debido a que se han borrado otros datos.

Anomalías de inserción: imposibilidad de adicionar datos en la BD, debido a la ausencia de otros datos.

El diagrama entidad-relación de la herramienta a desarrollar, se muestra en la **Figura 7**.

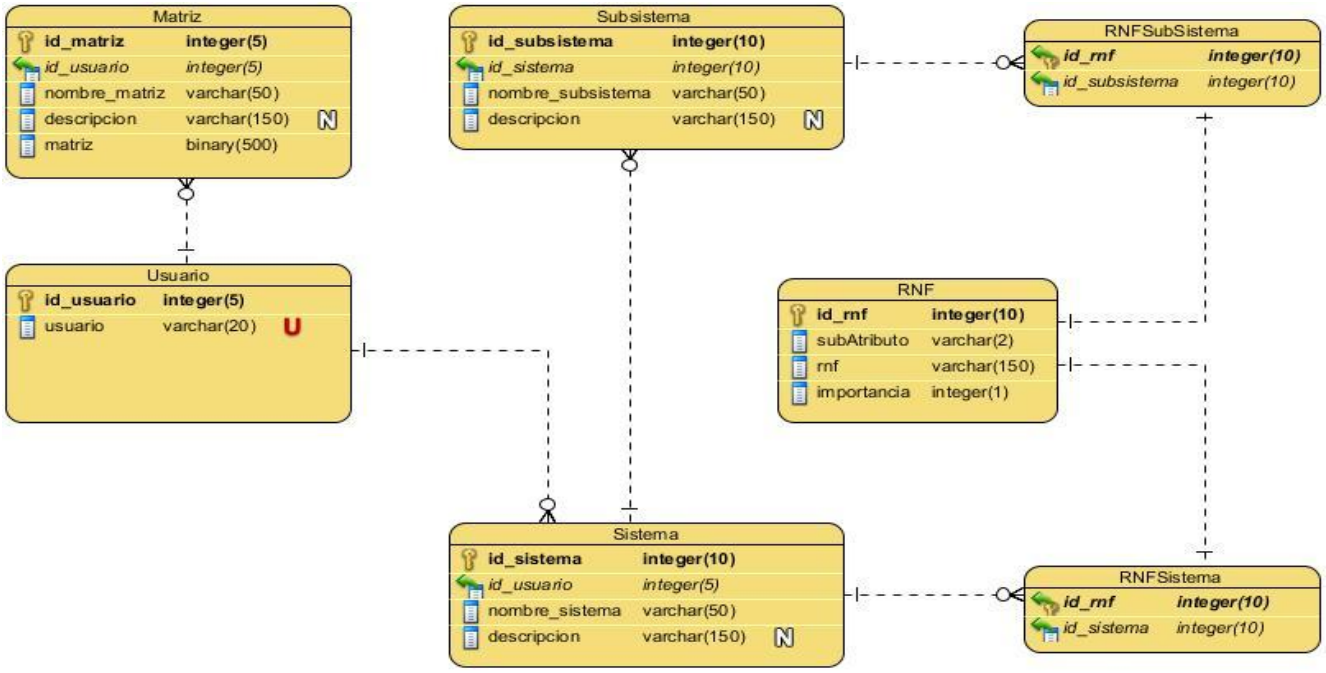

Figura 7. Diseño de la base de datos.

En la **Figura 7** se puede constatar que la base de datos del sistema va a estar compuesta por 8 tablas, que representan el almacenamiento de los datos que persistirán más allá de la ejecución del software. Además se puede apreciar que la base de datos, está normalizada ya que cumple con la tercera forma normal, debido a lo siguiente:

- ✓ Los atributos de cada una de las tablas son atómicos, no siendo ni multivaluados ni compuestos, cumpliendo así con la primera forma normal.
- ✓ No existen tablas con llaves compuestas en la cual los atributos presenten dependencias funcionales parciales, cumpliendo así con la segunda forma normal.

- ✓ Todos los atributos de las tablas dependen directamente de la llave primaria, cumpliendo así con la tercera forma normal.

Una vez demostrado que el diseño de la base de datos se encuentra normalizado hasta la tercera forma normal, se realiza una breve descripción de las tablas que constituyen las entradas fundamentales para dar cumplimiento al requisito arquitectónicamente más significativo para el negocio: Realizar Análisis de interrelaciones de atributos de calidad.

Entidad Matriz: constituye una de las dos entradas fundamentales para dar cumplimiento al requisito: Realizar análisis de interrelaciones de atributos de calidad, debido a que es la encargada de almacenar los valores de las matrices de interrelaciones de atributos de calidad que va a tener cada usuario.

Entidad RNF: es la tabla base para realizar los requisitos: Realizar priorización del sistema y Realizar priorización del subsistema, debido a que es la encargada de almacenar los valores que se utilizarán para priorizar los mismos. Esta priorización constituye la otra entrada fundamental para poder realizar el requisito arquitectónicamente más significativo para el negocio: Realizar Análisis de interrelaciones de atributos de calidad. De esta tabla heredan las tablas *RNFSistema* y *RNFSubsistema,* las cuales tienen una relación de uno a uno con las tablas *Sistema* y *Subsistema* respectivamente.

Patrones de diseño de base de datos

Se utilizaron varios patrones de diseño en el modelo entidad-relación, dentro de los que se encuentran los patrones descritos por Kyle Brown y Bruce G. Whitenack. A continuación se describen la utilización de los mismos.

Representación de objetos como tablas

Problema: ¿Cómo representar un objeto en un esquema de base de datos relacional?

Solución: Definir una tabla, para cada clase de objetos persistentes. Los atributos de las clases serán las columnas de las tablas.

Identificador de objetos

Problema: ¿Cómo mantener la identidad de un objeto en una base de datos relacional? Cada identidad de objetos individuales debe ser presentada en la base de datos.

Solución: Asignar un identificador independiente a cada objeto persistente. Se recomienda el uso de un generador de secuencias si hay alguno disponible en la base de datos ya que los identificadores son generalmente enteros largos, que garantizan que sean únicos para una clase de objetos en particular.

Referencia de llaves foráneas

Problema: ¿Cómo representar objetos que referencian otros objetos que no son de tipos de datos base?
Solución: Asignar a cada objetos un identificador único. Luego añadir una columna por cada variable de instancia que no tenga un tipo de dato base o sea una colección. En esa columna almacenar el identificador del objeto referenciado y declarar la columna como llave foránea.

Llaves subrogadas

Son un identificador único que se asigna a cada registro de una tabla. Son de tipo numérico secuencial sin significado especial y debe ser el único campo que sea clave primaria de cada tabla.

2.5 Validación del diseño propuesto

Una vez realizado el diseño de la herramienta a desarrollar se hace necesario por parte del equipo de desarrollo validar el mismo, para comprobar si el diseño realizado tiene una calidad aceptable. Para esto se aplicaron las métricas de calidad: Tamaño operacional de la clase (TOC) y Relaciones entre clases (RC), debido a que brindan un esquema sencillo de implementar y que a la vez cubre los principales atributos de calidad de software, siendo esto la principal razón de la concepción de las métricas (Pressman, 2002).

Tamaño operacional de clase

Esta métrica está relacionada con el número de métodos asignados a una clase. Después de realizado un análisis profundo en cada una de las clases utilizadas, la evaluación de la métrica TOC, reflejó los siguientes resultados:

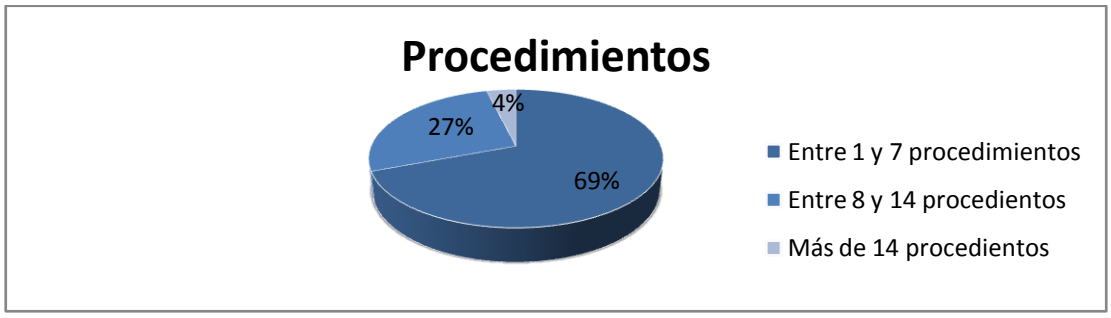

Figura 8. Representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

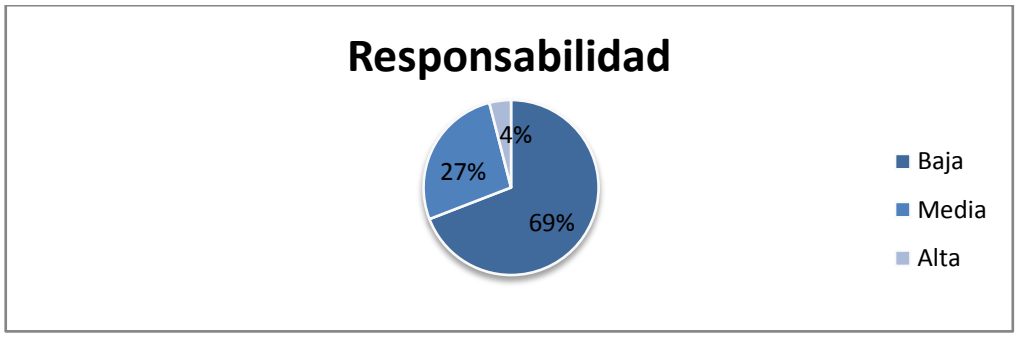

Figura 9. Representación de la incidencia de los resultados de la métrica TOC para el atributo Responsabilidad.

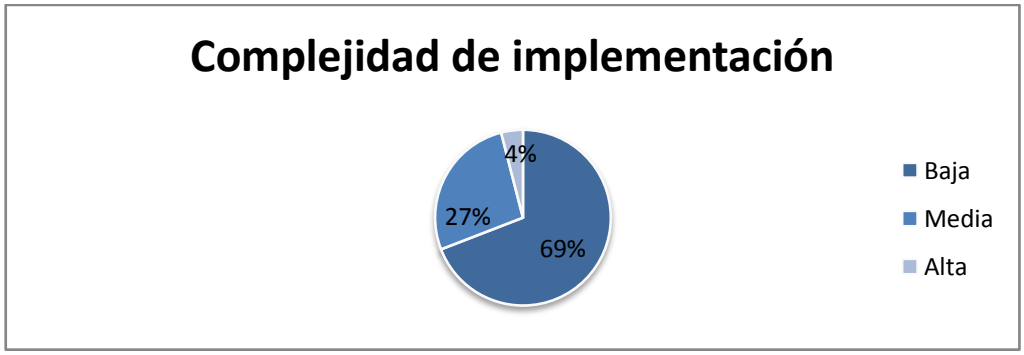

Figura 10. Representación de la incidencia de los resultados de la métrica TOC para el atributo Complejidad de implementación.

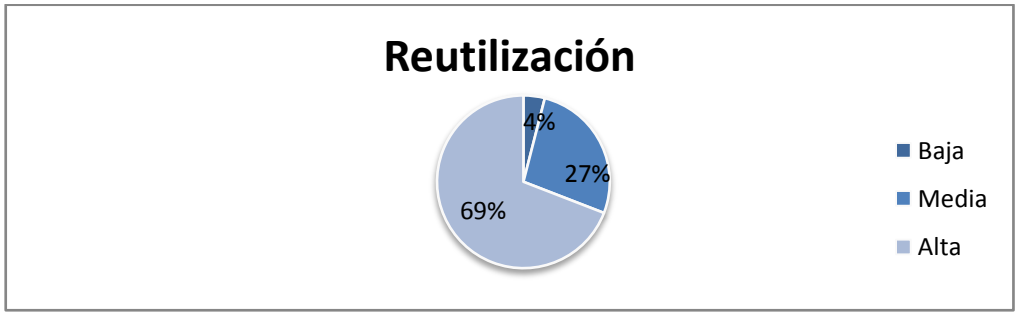

Figura 11. Representación de la incidencia de los resultados de la métrica TOC para el atributo Reutilización.

Al aplicar la métrica TOC se determinó que la aplicación consta de 26 clases y un total de 190 procedimientos, con promedio de 7 procedimientos por clases. La **Figura 8** muestra el promedio de clases por procedimientos, donde se puede observar que el 69% de las clases poseen a lo máximo 7 procedimientos por clases. La **Figura 9**, **Figura 10**, y **Figura 11** muestran los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC en los diferentes atributos de calidad. Estos resultados demuestran el diseño de la herramienta es lo más simple posible, permitiendo que un 69% de las clases tengan una sencilla implementación y responsabilidad entre ellas. Además el 69% de las clases poseen un alto nivel reutilización, el cual es un elemento clave en el proceso de desarrollo. Por todo lo anterior se puede concluir que el diseño tiene una calidad aceptable.

Relaciones entre clases

Esta métrica está dada por el número de relaciones de uso de una clase con otras. Después de realizado un análisis profundo en cada una de las clases utilizadas y sus dependencias, la evaluación de la métrica RC, reflejó los siguientes resultados:

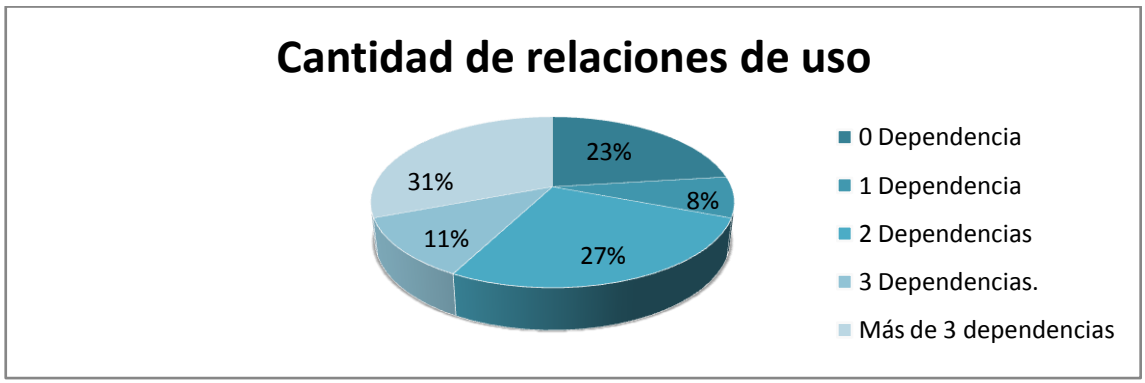

Figura 12. Representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

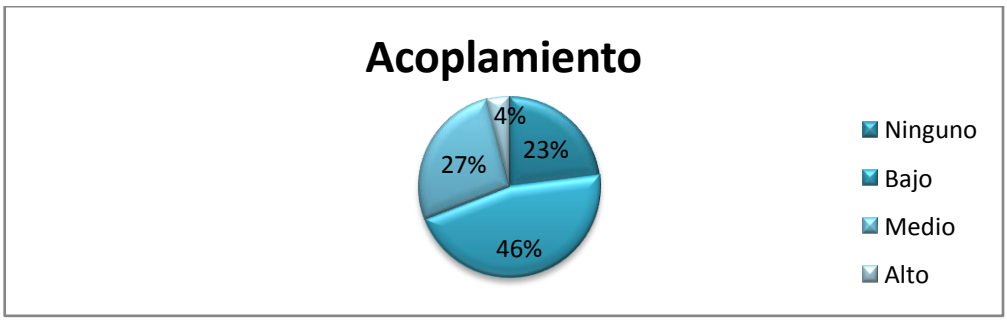

Figura 13. Representación de la incidencia de los resultados de la evaluación de la métrica Relaciones entre clases del atributo Acoplamiento.

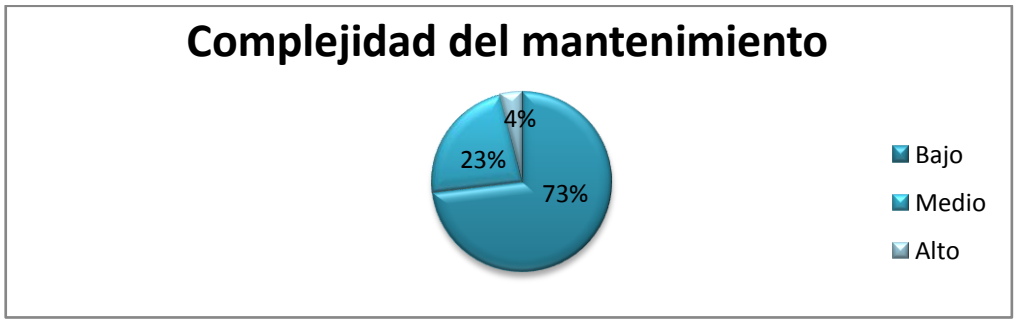

Figura 14. Representación de la incidencia de los resultados de la evaluación de la métrica Relaciones entre clases del atributo Complejidad del mantenimiento.

Figura 15. Representación de la incidencia de los resultados de la evaluación de la métrica Relaciones entre clases del atributo Reutilización.

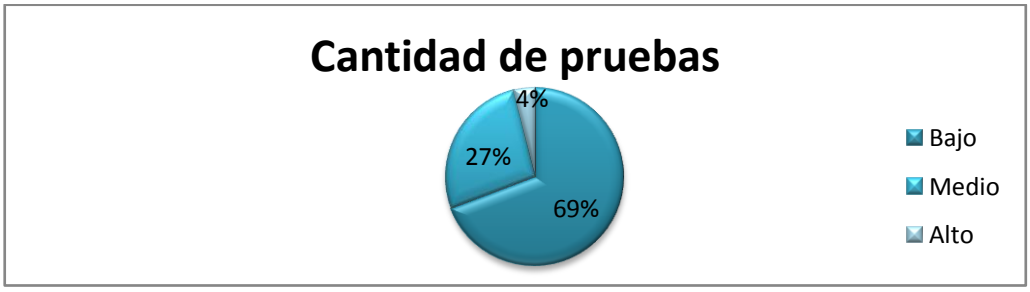

Figura 16. Representación de la incidencia de los resultados de la evaluación de la métrica Relaciones entre clases del atributo Cantidad de pruebas.

Al aplicar la métrica RC se determinó que la aplicación consta de 26 clases y un total de 70 dependencias entre estas, con promedio de 3 dependencias por clases. La **Figura 12** muestra el promedio de clases por cantidad de dependencias, donde se puede observar que el 69% de las clases poseen menos de 3 dependencias por clases. La **Figura 13**, **Figura 14**, **Figura 15** y **Figura 16** muestran los resultados obtenidos en los diferentes atributos de calidad. Estos resultados demuestran que el diseño propuesto para la herramienta a desarrollar tiene una aceptable calidad, modificabilidad e implantación del diseño, debido a que el 69% de las clases poseen una alta reutilización y un 73% de las clases poseen un bajo acoplamiento y complejidad del mantenimiento.

Al aplicar las métricas a las clases del diseño definidas para dar cumplimiento a la herramienta a desarrollar, se cumple que las métricas TOC y RC, favorecen las restricciones para la aceptación de las clases.

Después de precisadas todas las funcionalidades que debe tener la herramienta, la arquitectura que va a regir a la misma, generado los artefactos descritos anteriormente y realizada la validación del diseño propuesto, quedan sentadas las bases que le permiten al equipo de desarrollo del presente trabajo, tener las condiciones fundamentales para pasar a la implementación de la herramienta.

2.6 Conclusiones Parciales

El estudio realizado en este capítulo, es la base para poder pasar a la implementación de la herramienta del presente trabajo debido a que:

- ✓ Se definieron todos los requerimientos del sistema, lo cuales responden a las funcionalidades que debe brindar la herramienta a desarrollar.
- ✓ Se fundamentaron los patrones de requisitos empleados en el desarrollo de la solución propuesta, los cuales hicieron mucho más fácil y organizado el tratamiento a los requerimientos del sistema.
- ✓ La arquitectura en capas seleccionada para el sistema, permite construir aplicaciones web con una mayor calidad y organización.
- ✓ Se realizaron y describieron los diagramas de clases del diseño del requisito arquitectónicamente más significativo para el negocio, así como de los requerimientos que son la base para darle solución al mismo. Estos posibilitan tener un mejor entendimiento acerca de cómo será la implementación del sistema, para darle cumplimiento a estos requisitos y como va a ser la comunicación entre las clases e interfaces del sistema entre cada capa, teniendo en cuenta la arquitectura seleccionada.
- ✓ Se fundamentaron los patrones de diseño empleados en el desarrollo de la solución propuesta, los cuales proporcionan un mejor acoplamiento, baja dependencia, alta reutilización y reduce el impacto de los cambios entre las clases del sistema.
- ✓ Se diseñó y describió el modelo de base de datos por el cual va a estar regido el sistema, permitiéndole a los usuarios y cliente final tener un mejor entendimiento acerca de las tablas persistentes que va a tener el sistema, así como las relaciones entre estas.
- ✓ Se fundamentaron los patrones de base de datos empleados en el desarrollo de la solución propuesta, los cuales fueron esenciales para transformar los objetos y sus relaciones de un modelo orientado a objetos, a un modelo relacional.
- ✓ Se realizó la validación del diseño, el cual demostró que el diseño realizado tiene una calidad aceptable.

Capítulo 3: Implementación y validación de la solución propuesta

3.1 Introducción

Este capítulo se centra en la actividad de implementación de la herramienta a desarrollar, basado en la metodología ágil de desarrollo de software AUP, seleccionada por el equipo de desarrollo para guiar el proceso de desarrollo del presente trabajo. Esta actividad es la encargada de convertir los diagramas antes expuestos en código ejecutable. Se exponen los artefactos generados en esta actividad y todo lo concerniente a la realización de las pruebas de aplicación y validación de la herramienta propuesta.

3.2 Diagrama de componentes

En la fase de construcción de la metodología AUP, una vez realizada la actividad de implementación del sistema, se realiza el diagrama de componentes como uno de los artefactos que se generan en esta actividad. Este diagrama es utilizado para modelar la vista estática del sistema y representar las dependencias y organización entre los componentes que lo componen.

El diagrama de componentes de la herramienta propuesta se muestra en la **Figura 17.**

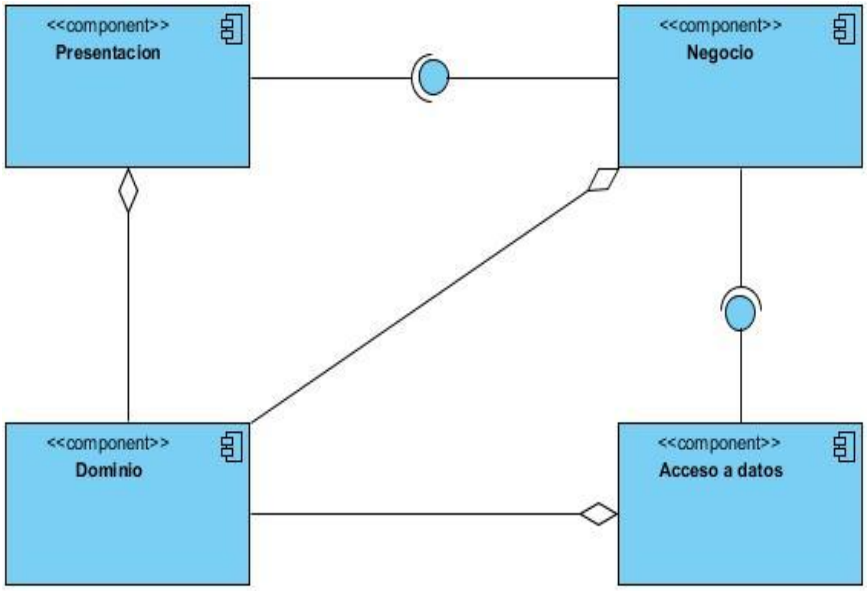

Figura 17. Diagrama de componentes**.**

En el diagrama de componentes de la **Figura 17**Figura 1 se puede ver como se establece un componente por cada capa de la aplicación, con el objetivo de hacer más fácil el mantenimiento de la

herramienta en un futuro; por ejemplo si se quisiera cambiar el gestor de base de datos, solamente se afectaría el componente de Acceso a datos y por lo tanto sería el único componente que sería necesario cambiar, disminuyendo así la complejidad del cambio y aumentado la rapidez de adopción del mismo. Además se puede ver como se relacionan los componentes Presentación, Negocio y Acceso a datos con el componente Dominio y como el acceso entre cada componente se realiza a través de interfaces.

3.3 Diagrama de despliegue

En la fase de construcción de la metodología AUP, una vez terminada la actividad de implementación se realiza el diagrama de despliegue. Este diagrama muestra las relaciones físicas de los distintos nodos[17] que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue, representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación.

El diagrama de despliegue de la herramienta a desarrollar se muestra en la **Figura 18**.

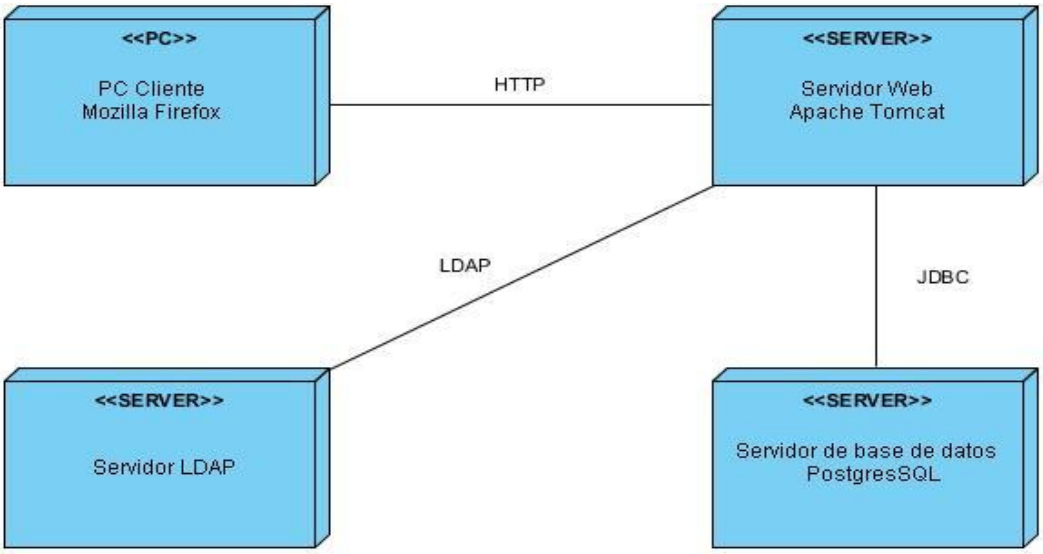

Figura 18. Diagrama de despliegue.

Como se puede apreciar en la **Figura 18** para poder utilizar la herramienta, se requiere una computadora que tenga instalado el navegador web Mozilla Firefox, un servidor donde se encontrará

[17] nodos: *Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria* (Marca, et al., 2000).

desplegada la herramienta en el servidor web Apache Tomcat y otra máquina server, que puede ser la misma del servidor web, donde se encuentre la base datos que utiliza la herramienta. Además para acceder a las funcionalidades de la herramienta es necesario autenticarse contra un servidor LDAP.

3.4 Pruebas de software

Después de concluida la actividad de implementación, el software debe ser probado, para detectar y corregir errores, antes de entregar el producto. Las pruebas aplicadas a la herramienta son las pruebas de caja negra, también denominadas prueba de comportamiento, las cuales se centran en los requisitos funcionales del software según (Pressman, 2002). Estas pruebas se llevan a cabo sobre la interfaz del software. O sea los casos de pruebas pretenden demostrar que las funciones de software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto. La misión de estas pruebas es detectar errores como:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en acceso a base de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

Los diseños de casos de prueba de los requisitos funcionales de la herramienta fueron entregados al cliente una vez que fueron probados satisfactoriamente en cuanto a la entrada y salida de los datos.

3.5 Validación de la herramienta

Una vez realizados y probados satisfactoriamente todos los requerimientos funcionales de la herramienta propuesta, se hace necesario validarla para comprobar que la misma arroja los resultados precisos en el momento de realizar las funcionalidades: Realizar priorización a un subsistema o sistema en general y Realizar análisis de interrelaciones de atributos de calidad, las cuales son las que muestran al usuario los resultados que servirán para la toma de decisiones arquitectónica. Para la validación fue tomado un juego de datos real correspondiente al sistema Producto Planificación y a los subsistemas que lo componen los cuales son: Estructura y composición, Configuración y Planificación. Una vez definido el juego de datos, fue realizado por parte del equipo de desarrollo del presente trabajo un levantamiento de requisitos no funcionales (RNF) asociados al sistema Producto Planificación. Para el levantamiento de RNF fueron entrevistados al analista, arquitecto y jefe de cada subsistema y sistema en general, con el objetivo

de combinar los criterios y crear una planilla general de RNF de cada subsistema y sistema en general. Para el caso del sistema Producto Planificación los RNF asociados a la planilla general, son los requisitos comunes en las tres planillas generales de cada uno de sus subsistemas. Los criterios para que los entrevistados pudieran establecer la prioridad de cada requisito no funcional de atributos de calidad, son los siguientes:

5: Muy importante

4: Importante

3: Medianamente importante

2: Poco importante

1: No importante

Con el objetivo de demostrar que los datos arrojados por la herramienta son los precisos, en el momento de darle solución a las funcionalidades basadas en métodos matemáticos y algoritmos de programación, se toma una pequeña muestra del juego de datos real, asociado al levantamiento de (RNF) de atributos de calidad realizado.

Muestra del juego de datos a probar para la funcionalidad: Priorizar Subsistema

Para probar esta funcionalidad se toman algunos fragmentos del levantamiento de RNF general realizado al subsistema Configuración, el cual se muestra en el siguiente árbol de calidad, compuesto en (orden descendente), por los atributos de calidad, subatributos y RNF siguientes:

1. **Confiabilidad 4.50**
   - 1.1. Recuperabilidad 4.50
     - 1.1.1. 4 El sistema tendrá un servidor de datos de respaldo que se actualizará diario.
     - 1.1.2. 5 El sistema será transaccional.

2. **Funcionalidad 3**
   - 2.1. Interoperabilidad 2
     - 2.1.1. 2 El sistema permitirá generar reportes estándares en formato XML.
   - 2.2. Seguridad4
     - 1.2.2. 4 El sistema concederá acceso a cada usuario autenticado solo a las funciones que le estén permitidas, de acuerdo a la configuración del sistema.

3. **Eficiencia 4**
   - 3.1. Rendimiento 4
     - 3.1.1. 4 El sistema no excederá los 3 segundos de respuesta al efectuar acciones de

cargar un registro (esta cifra no incluye los retardos por concepto de tráfico de red).

4. **Mantenibilidad 5**
   4.1. Diagnosticabilidad 5
      4.1.1. 5 El sistema deberá poseer un mecanismo de almacenamiento, detección y tratamiento de errores.

5. **Usabilidad 3**
   5.1. Operabilidad 3
      5.1.1. 3 La aplicación debe ser uso de las teclas calientes.

6. **Portabilidad 4**
   6.1. Reemplazabilidad 4
      6.1.1. 4 El producto deberá ser capaz de reemplazar a los sistemas en el Documento Visión en el mismo ambiente en que estos de ejecutaban.
   6.2. Coexistencia
      6.2.1. 4 El sistema interactuará con herramientas ofimáticas y visualizador de ficheros PDF para la presentación de los reportes que genere y los ficheros que importe.

Después de conocido el valor de importancia de cada RNF de subatributos de calidad, es necesario conocer la prioridad de los atributos a que pertenecen estos, para esto se emplea el criterio de la media aritmética, expresado mediante la fórmula: $M = (\sum Xi) / fi$

Xi: *Valores de importancia asociados a cada RNF.*
fi: *Cantidad de requisitos asociados a cada valor.*

En la **Tabla 5** se exponen los resultados de aplicarle la fórmula de la media aritmética a los atributos de calidad.

Atributos	**Fórmula** $M = (\sum Xi) / fi$	**Resultado**
Confiabilidad	4+5 / 2	4.50
Funcionalidad	2+4/2	3
Eficiencia	4/1	4
Mantenibilidad	5/1	5
Usabilidad	3/1	3

Portabilidad	4+4/2	4

Tabla 5. Valores de priorización para los atributos del subsistema Configuración para la muestra del juego de datos.

En la **Tabla 5** se muestra el resultado para priorizar el subsistema Configuración para la muestra tomada del juego de datos real. Teniendo en cuenta que a mayor valor de media mayor prioridad, obtendríamos para la muestra tomada el siguiente orden de prioridad: Mantenibilidad 5, Confiabilidad 4.50, Eficiencia-Portabilidad 4, y Funcionalidad- Usabilidad 3.

En el caso de que la priorización de los atributos de valores iguales, nótese, en el ejemplo anterior, que los atributos señalados en rojo Eficiencia y Portabilidad tienen la misma prioridad con valor 4, y los atributos Funcionalidad y Usabilidad tienen la misma prioridad con valor 3. En este caso que dos o más atributos de calidad tienen igual valor, se propone emplear la desviación estándar para distinguir que atributos de calidad con igual valor de media debe tener mayor prioridad, en este caso sería el de menor desviación de sus valores de origen. Siendo la desviación estándar igual a la raíz cuadrada positiva de la varianza, se calcula entonces el valor de la última mediante la fórmula $S2 = \sum^{n} (Xi - M)^2 / N$, donde:

S2 = *Varianza.*
Xi = *Valor del atributo de calidad en cada subsistema y en el sistema si se esta priorizando el sistema.*
M= *Valor de la media del conjunto de valores asociados al atributo de calidad.*
N= *Cantidad total de valores analizados.*

Los resultados de aplicar la fórmula de la desviación estándar a los atributos con igual valor de media resaltados en rojo en la **Tabla 5**, se muestran en la **Tabla 6**.

Atributos	**Formula** **$\sum$n (Xi - M)2 / N**	**Resultado**
Eficiencia	$((4 -2)^2) / 2$	$\sqrt{2} = 1.41$
Portabilidad	$((4-2)^2 + (4 – 2)^2) /2$	$\sqrt{4} = 2$
Funcionalidad	$((2-3^2) + (4-3)^2)/2$	$\sqrt{1} = 1$
Usabilidad	$((3-3^2)/1$	$\sqrt{0} = 0$

Tabla 6. Resultados al aplicar la desviación estándar para los atributos con igual valor de media.

Tras los resultados arrojados en la **Tabla 6**, el subsistema Configuración para la muestra tomada del juego de datos real tiene el siguiente orden de prioridad: Mantenibilidad 5, Confiabilidad 4.50, Eficiencia 4, Portabilidad 4, Usabilidad 3 y Funcionalidad con 3.

Los resultados arrojados por la herramienta desarrollada para la muestra tomada del juego de datos real, se muestra en la **Figura 19**.

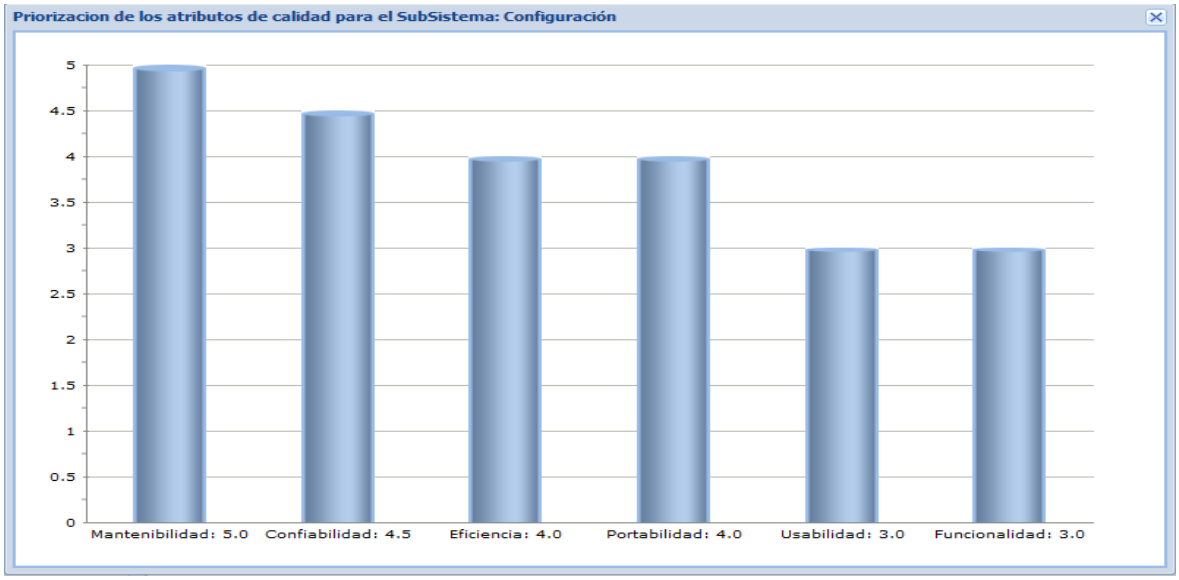

Figura 19. Priorización de atributos para el subsistema Configuración de la muestra tomada del juego de datos real.

Como se puede observar la **Figura 19** arroja los resultados que se obtienen al realizar el proceso manualmente, por lo que se puede concluir que el sistema muestra los datos precisos en cuanto a todo lo que engloba el proceso de realizar la priorización de subsistemas. Se infiere además que con esta herramienta el proceso de priorización sea mucho más rápido, en cuanto a la obtención de los valores y centralización de los datos necesarios para realizar este proceso.

Muestra del juego de datos a probar para la funcionalidad: Priorizar Sistema

Una vez establecida la prioridad de los atributos de calidad por cada subsistema es posible determinar la prioridad general del sistema. Tomando para el cálculo de esta dos grupo de valores:

1. Los valores de prioridad calculados para cada subsistema.
2. Los valores de prioridad de atributos y subatributos en base a los requisitos no funcionales generales del sistema.

Es importante destacar que al sistema también se le puede gestionar RNF de atributos de calidad en caso de que sea necesario, para esto se emplea el mismo procedimiento que el explicado anteriormente para el subsistema. En caso de que no tenga asociado ningún RNF el sistema, la prioridad a seguir será la misma que la calculada para el subsistema asociado a este en caso de ser uno solo. En caso de que el sistema tenga más de un subsistema asociado, se toman los valores de los atributos para cada subsistema asociado a este y se emplea el mismo criterio de la media aritmética para los atributos de calidad explicado anteriormente.

Para probar esta funcionalidad se toma como muestra del juego de datos real algunos fragmentos del levantamiento de RNF general realizado al sistema Producto Planificación, el cuál se muestra en el siguiente árbol de calidad, compuesto en (orden descendente), por los atributos de calidad, subatributos y RNF siguientes:

1. **Confiabilidad 4**
   1.1. Madurez 4
      1.1.1. 4 El sistema no permitirá la entrada de datos incorrectos.

2. **Funcionalidad 2**
   2.1. Idoneidad 2
      2.1.1. 2 El sistema permitirá generar reportes estándares en formato PDF, XLS, HTML.

3. **Eficiencia 4**
   3.1. Utilización de recursos 4
      3.1.1. 4 En los casos en que el sistema provea información a otro sistema, cualquier cambio en el estado de dicha información deberá ser notificado al sistema consumidor.

4. **Mantenibilidad 3**
   4.1. Flexibilidad 3
      4.1.1. 3 El sistema permitirá agregar nuevas funcionalidades o modificar alguna existente sin romper la estructura y consistencia de los componentes.

5. **Usabilidad 3.50**
   5.1. Comprensibilidad 3.50
      5.1.1. 3 Todos los mensajes de error del sistema deberán incluir una descripción textual del error.
      5.1.2. 4 El orden de las etiquetas de funcionalidades en el menú responderá a las dependencias de ejecución de negocio.

6. **Portabilidad 5**
   6.1. Reemplazabilidad 5
      6.1.1. 5 El producto deberá ser capaz de reemplazar a los sistemas especificados en el Documento Visión en el mismo ambiente en que estos se ejecutaban.

Es decir la priorización del sistema para la muestra tomada del juego de datos real será: Portabilidad 5, Confiabilidad y Eficiencia 4, Usabilidad 3.50, Mantenibilidad 3, Funcionalidad con 2.

En la **Tabla 7,** se muestra el grupo de valores de los atributos priorizados para el subsistema y sistema en general de las muestras tomadas del juego de datos real.

Sistemas	**Funcionalidad**	**Confiabilidad**	**Eficiencia**	**Usabilidad**	**Portabilidad**	**Mantenibilidad**
Subsistema Configuración	3	4.50	4	3	4	5
Sistema Producto Planificación	2	4	4	3.50	5	3

Tabla 7. Valores de prioridad de los atributos del Subsistema Configuración y Sistema Producto Planificación para la muestra tomada del juego de datos real.

Tomándose como bases los valores de la priorización de la **Tabla 7,** es posible calcular la media aritmética de cada atributo y de ahí determinar las prioridades de cada uno, es decir aplicando la fórmula M = (∑Xi) / fi. Los resultados al aplicar la fórmula se muestran en la **Tabla 8**.

Atributos	**Fórmula** **M = (∑Xi) / fi**	**Resultado**
Confiabilidad	4.50+4/ 2	4.25
Funcionalidad	3+2/2	2.50
Eficiencia	4+4/2	4
Mantenibilidad	5+3/2	4
Usabilidad	3+3.50/2	3.25

Portabilidad	4+5/2	4.50

Tabla 8. Valores de la priorización para el sistema Producto Planificación de la muestra tomada del juego de datos real.

Tras los resultados arrojados por la **Tabla 8** los valores de prioridad del sistema Producto Planificación para la muestra tomada del juego de datos real tiene el siguiente orden de prioridad: Portabilidad 4.50, Confiabilidad 4.25, Eficiencia-Mantenibilidad 4, Usabilidad con 3.25, Funcionalidad 2.50.

En la **Tabla 8** podemos observar que los atributos señalados en rojo Mantenibilidad y Eficiencia tienen el mismo resultado y no se sabe con exactitud quien debe tener mayor prioridad. Para saber esto se emplea el mismo procedimiento que cuando se tienen valores iguales en el subsistema, es decir mediante la fórmula de la varianza.

En la **Tabla 9** se muestra el resultado de la desviación estándar de los atributos con igual valor de media.

Atributo	**Fórmula** $S2 = \sum^{n} (Xi - M)^2 / N$	**Resultado**
Eficiencia	$((4 -4)^2 + (4 -4)^2) / 2$	√0
Mantenibilidad	$((5 -4)^2 + (3 -4)^2) / 2$	√1

Tabla 9. Resultados al aplicar la desviación estándar para los atributos con igual valor de media para el sistema Producto Planificación de la muestra tomada del juego de datos real.

Tras los resultados arrojados por la **Tabla 9** los valores de prioridad del sistema Producto Planificación para la muestra tomada del juego de datos real tiene el siguiente orden de prioridad: Portabilidad 4.50, Confiabilidad 4.25, Eficiencia 4 y Mantenibilidad 4, Usabilidad con 3.25, Funcionalidad 2.50.

Es importante aclarar que cuando los valores de desviación dan igual, el primer valor que va a tener mayor prioridad a seguir es el asociado al primer atributo que aparezca en el árbol de calidad.

Los resultados arrojados por la aplicación para la muestra tomada del juego de datos real se muestran en la **Figura 20. Priorización del sistema Producto Planificación para la muestra tomada del juego de datos real.**

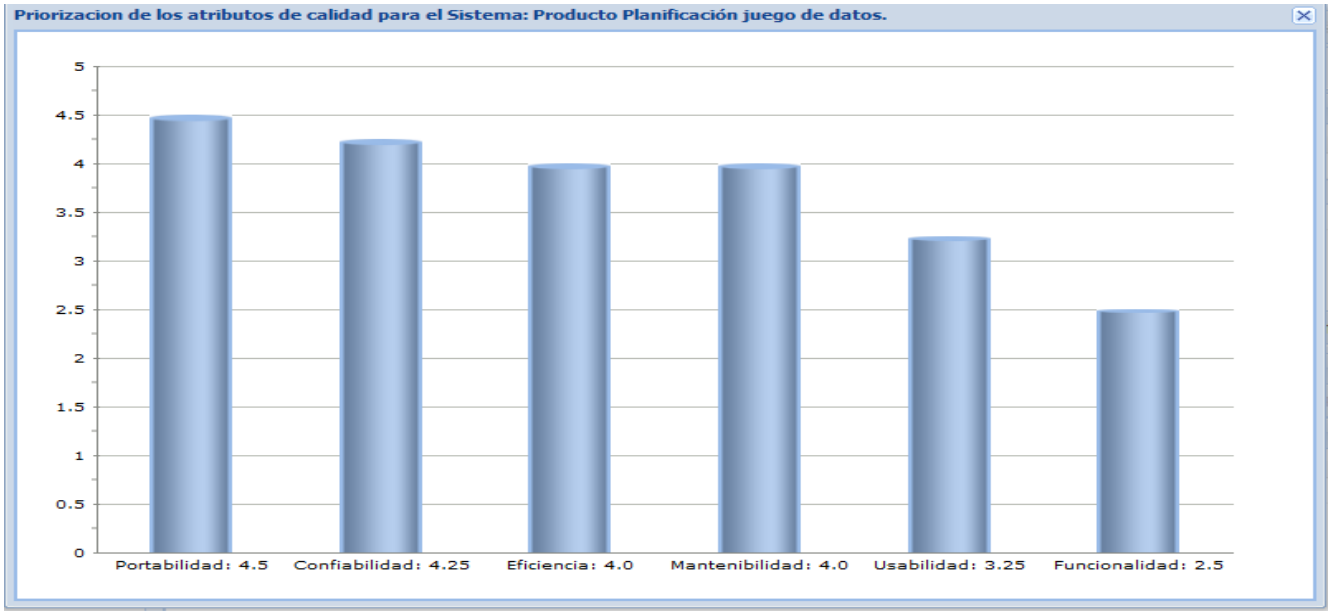

Figura 20. Priorización del sistema Producto Planificación para la muestra tomada del juego de datos real.

Como se puede observar la **Figura 20** arroja los mismos resultados que se obtienen al realizar el proceso manualmente, por lo que se puede concluir que el sistema muestra los datos precisos en cuanto a todo lo que engloba el proceso de realizar la priorización de sistema. Se infiere además que con este sistema el proceso de priorización sea mucho más rápido en cuanto a la obtención de los valores y centralización de los datos necesarios para realizar este proceso.

Una vez explicado el procedimiento a seguir en el momento de realizar la priorización para el subsistema y sistema en general a través de la muestra tomada del juego de datos real, se muestra en la **Figura 21** el resultado real de la priorización para el sistema Producto Planificación, con todos los datos existentes en las planillas generales, tanto para el sistema como para los subsistemas asociados a este.

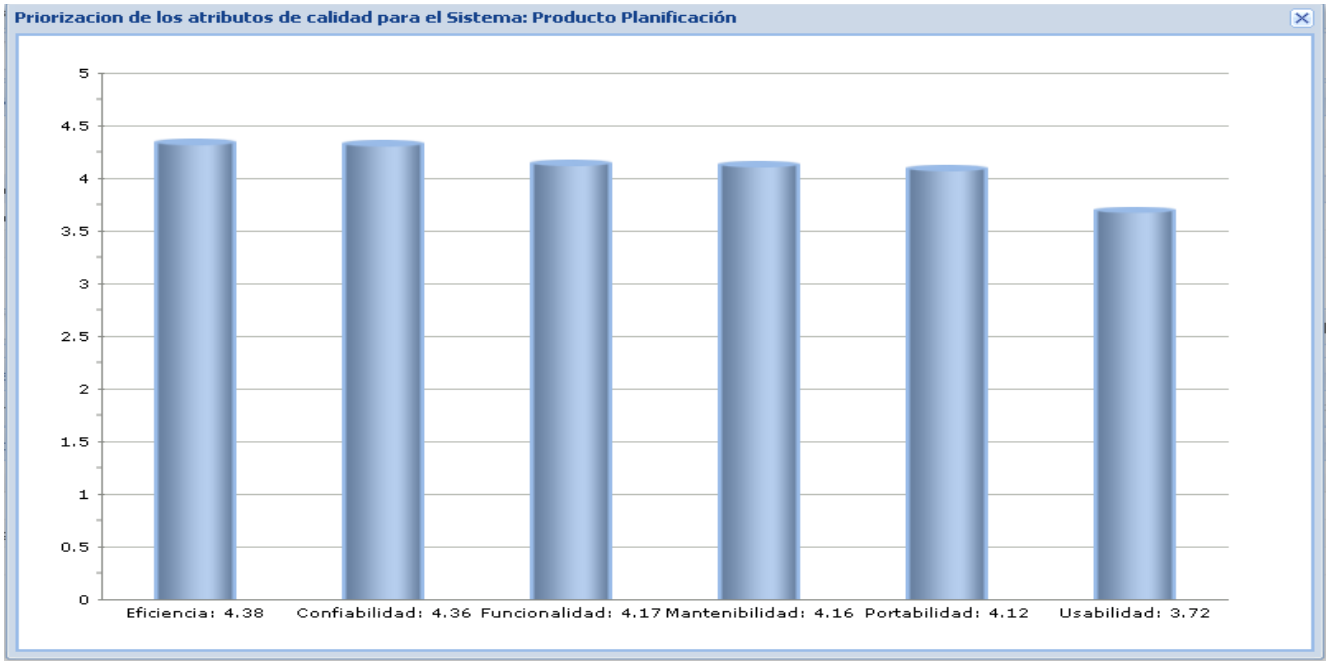

Figura 21. Priorización de los atributos de calidad para el sistema Producto Planificación del juego de datos real.

Después de realizados todos los cálculos internos del sistema, para mostrar los resultados finales es que se tiene en cuenta los siguientes criterios:

- ✓ Si el resultado arrojado finalmente es un número entero se conservan además los 2 primeros dígitos decimales de este, ejemplo 4 sería 4.00.
- ✓ Si el resultado no es un número entero, ejemplo 3.4761 se conservan los primeros 2 dígitos decimales, siendo redondeado por exceso si el dígito que sigue a continuación de estos se encuentra del 5-9, en este caso quedaría 3.48. En caso de que le siga un número del 1-4 se redondea por defecto, en este caso quedaría 3.47.

Juego de datos a probar para la funcionalidad: Realizar Análisis de interrelaciones

Las entradas necesarias para este algoritmo son:

- Matriz de interrelaciones
- La lista de prioridades de subatributos para el sistema.

Matriz de interrelaciones para el sistema Producto Planificación: La matriz con la que se realiza el análisis de interrelaciones para probar el juego de datos se muestra en la **Figura 22,** la cual solamente tiene algunas relaciones de los subatributos de calidad del sistema Producto Planificación, con el objetivo de que el grafo asociado a esta matriz sea lo más entendible posible.

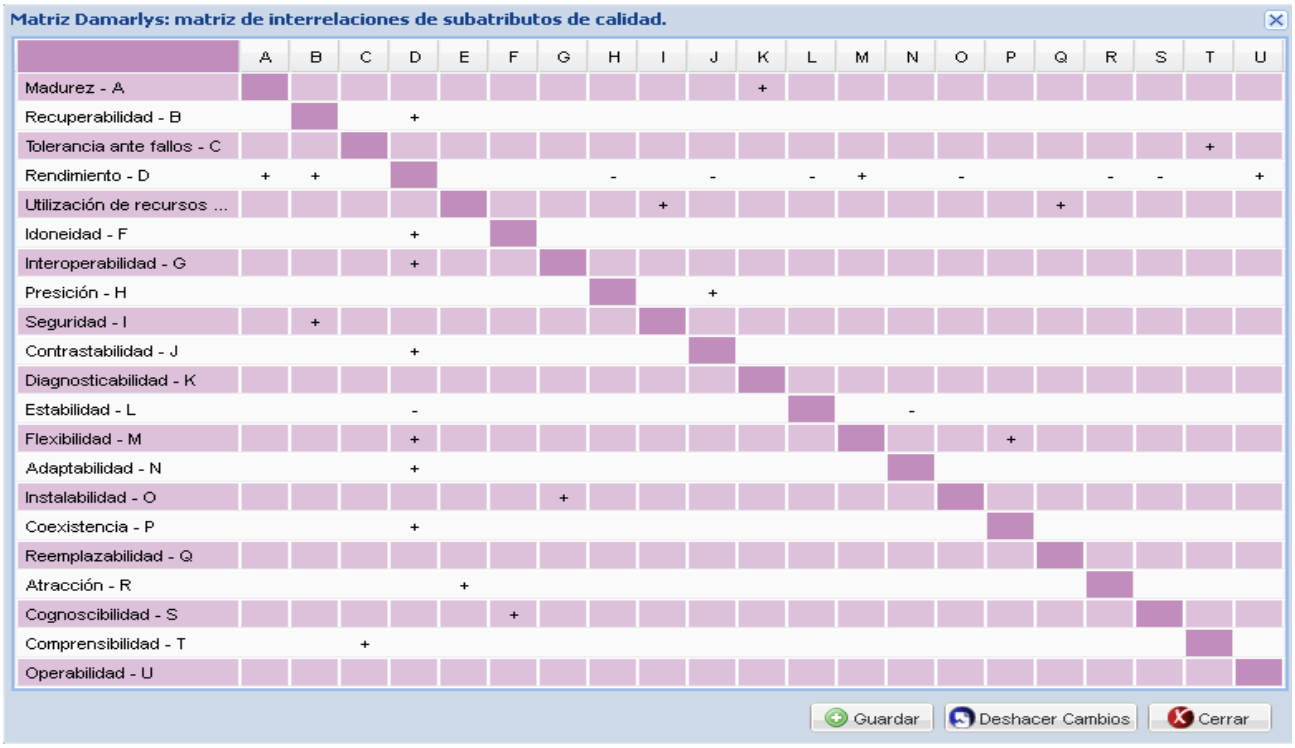

Figura 22. Matriz de interrelaciones de atributos de calidad para el juego de datos.

Lista de prioridades de subatributos para el sistema Producto Planificación: Esta lista se obtiene a partir del cálculo de las prioridades de los subatributos del sistema en general del jugo de datos real, el cual arrojó el siguiente orden de prioridad: O,B,H,K,C,Q,E,I,D,A,M,U,L,G,R,T,S,P,F,N,J. Con este orden de prioridad se construye la lista de prioridades que se le pasa al algoritmo de análisis de interrelaciones, la cual quedaría de la siguiente manera: (A=12, B=20, C=17, D=13, E=15, F=3, G=8, H=19, I=14, J=1, K=18, L=9, M=11, N=2, O=21, P=4, Q=16, R=7, S=5, T=6, U=10).

Una vez definida la matriz de interrelaciones se pasa a construir un grafo dirigido y ponderado que tiene tantos nodos como filas tiene la matriz, los pesos del camino de este grafo se establecen en función de la forma de afectación como se muestra en la **Tabla 10** y las aristas del grafo y sus direcciones se construyen a partir de las filas, o sea el origen lo determina la fila y el fin la columna. El grafo asociado a la matriz se muestra en la **Figura 23.**

Valor en la matriz	Peso en el grafo
+	**1**
-	**-1**
Sin valor	**0**

Tabla 10. Transformación de los valores de las celdas de la matriz para poner pesos en el grafo.

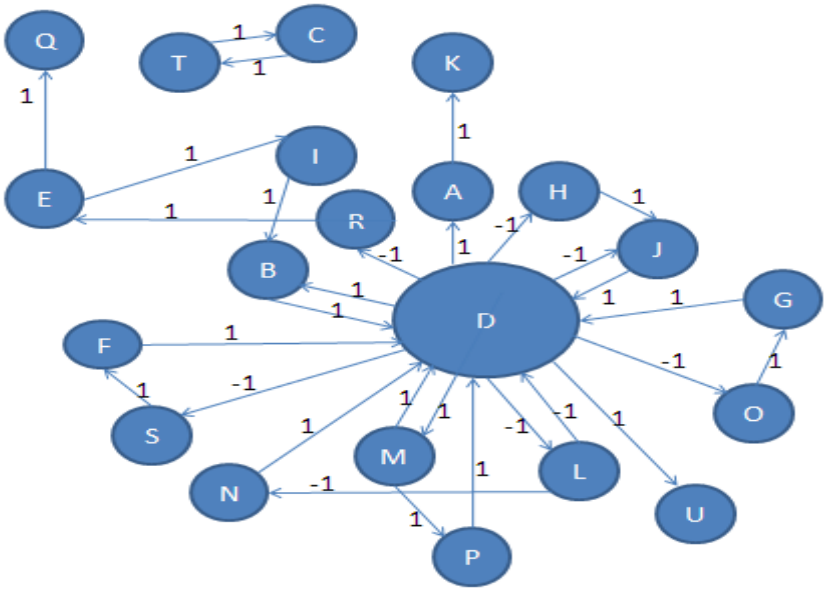

Figura 23. Grafo asociado a la matriz de interrelaciones del juego de datos.

Una vez obtenido el grafo se debe recorrer para obtener la mayor importancia posible de cada atributo según orden de prioridad, para esto se hace uso de la siguiente fórmula:

$$I_x = C_x * \sum_{i=1}^{C_x} R_i * P_i + A_x * \sum_{i=1}^{A_x} R_i * P_i$$

Donde:

I: *Importancia de un nodo*.
Cx: *Cantidad de nodos que afecta el nodo x*.
Ax: *Cantidad de nodos que afectan al nodo x.*
R: *Prioridad de un nodo*.
P: *Peso.*
X: *Nodo.*

Siguiendo el ejemplo de la **Figura 23** y tomando como prioridades de los nodos las calculadas en el ejemplo de la priorización anterior (A=12, B=20, C=17, D=13, E=15, F=3, G=8, H=19, I=14, J=1, K=18, L=9, M=11, N=2, O=21, P=4, Q=16, R=7, S=5, T=6, U=10), el valor de importancia, por ejemplo, para el nodo M quedaría:

$I_M = C_M*((R_D * P_D) + (R_{P*} P_P)) + A_M* ((R_D*P_D))$

$I_M = 2* ((13 * 1) + (4 * 1))+ 1 * ((13 * 1))$

$I_M = 2* ((17))+ 1* ((13))$

$I_M = 47$

Análogamente se calculan el resto de los valores de importancia para cada nodo del grafo. Hay que señalar que todo lo realizado hasta ahora en este algoritmo es solamente para el grafo inicial de la **Figura 23**. Pero se hace necesario determinar todas las posibles combinaciones de grafos que se puedan formar a partir de ese grafo inicial, esto se logra manteniendo o no, determinadas aristas del grafo en cada iteración del cálculo. Para poder generar todos los grafos posibles a partir de uno inicial es que se empleó un algoritmo recursivo que utiliza la técnica de Vuelta Atrás. La cantidad de grafos generados a partir del inicial se puede determinar mediante la fórmula:

$$\sum_{r=0}^{r-n} \frac{n!}{r!(n-r)!}$$

Donde **n** es la cantidad de aristas del grafo original y **r** es la cantidad en cada caso restante.

Aplicando la fórmula anterior al ejemplo de la **Figura 23**, donde n = 30 y r = {0, 1, 2, 3...30}, la cantidad de grafos generados es de 1073741824.

El próximo paso después de calculadas todas las importancias asociadas a un grafo, es determinar si el mismo puede ser factible a una solución, para esto el grafo debe cumplir con las siguientes condiciones:

1. $I_x > I_y$ cuando $R_x < R_y$
2. $I_x \neq 0$

Donde las **I** son las importancias de los nodos asociados a ese grafo y las **R** son las prioridades asociadas a dichos nodos.

Si el grafo es factible se calcula la desviación estándar de las importancias del mismo. La desviación estándar no es más que la raíz cuadrada de la varianza, cuya fórmula es:

$$\mathbf{S^2 = \sum^n (X_i - M)^2 / N}$$

Donde:

X_i=*Valor de importancia de un nodo del grafo.*
M =*Media de los valores de importancia del grafo.*
N: *Cantidad de nodos del grafo.*

Después de calculada la desviación estándar se trata de insertar el grafo en una lista donde están los primeros 25 grafos factibles ordenados de menor a mayor valor de desviación estándar. Si el grafo posee menor desviación que alguno de los existentes en la lista, este se inserta en la misma y los demás elementos se corren desechándose el último elemento de la lista. En el caso que el grafo que se quiera insertar no tenga la desviación estándar menor que ninguno de los 25 que conforman la lista, este se desecha. Para el caso en que tenga desviación estándar igual a alguno de la lista, entonces se pasa a comparar la importancia del primer nodo y si esta es mayor, entonces se inserta el grafo, corriéndose los demás elementos de la lista.

Una vez analizados todos los grafos que se generaron a partir del original, se procede a identificar el caso óptimo, para esto se selecciona de la lista que contiene los 25 grafos con menor desviación estándar, el grafo que tenga mayor primer valor de importancia. En el caso de que existiera más de un grafo con igual primer valor de importancia se tomaría de ellos el que tenga menor valor de desviación. Para el ejemplo que se viene desarrollando el grafo óptimo es el que tiene los siguientes valores de importancias: (I_A = 31.0, I_B = 0.0, I_C = 0.0, I_D = 10.0, I_E = 7.0, I_F = 5.0, I_G = 0.0, I_H= 0.0, I_I= 0.0, I_J = 0.0, I_K = 12.0, I_L = 0.0, I_M = 0.0, I_N = 0.0, I_O = 0.0, I_P = 0.0, I_Q = 0.0, I_R = 2.0, I_S = 3.0, I_T = 0.0, I_U = 0.0) y la matriz de interrelaciones asociada a este grafo óptimo sería la de la **Figura 24.**

Figura 24. Matriz de Interrelaciones del caso óptimo.

En la ejecución del algoritmo de análisis de interrelaciones explicado anteriormente surge un inconveniente asociado al tiempo de ejecución del mismo, debido a que mientras más interrelaciones tenga la matriz de entrada, más aristas tendrá el grafo inicial asociado a esta, y por ende más grafos será necesario generar. Es por esto que surge el problema de que mientras más grafos se tengan que generar, más tiempo se demorará la ejecución del algoritmo. Con el objetivo de tener una idea exacta del tiempo de ejecución del algoritmo, se probó el mismo para un grafo inicial de 20 aristas, donde es necesario generar 1 048 576 grafos, el cual se demoró aproximadamente 7 segundos. Posteriormente se hizo otra prueba esta vez para un grafo inicial de 21 aristas donde es necesario generar 2 097 152 grafos y este se demoró aproximadamente 15 segundos. En las pruebas realizadas anteriormente se puede apreciar que si el tiempo que demora para calcular una cantidad de grafos X es aproximadamente Y, entonces el tiempo que demorará para calcular una cantidad de grafos 2X es aproximadamente 2Y.

De lo anterior se puede concluir que cuando se necesita calcular una gran cantidad de grafos debe demorarse una gran cantidad de tiempo. Es por esto que se hace necesario aplicarle técnicas de programación paralela al algoritmo, con el objetivo de que se analicen varios grafos al mismo tiempo, y en consecuencia el tiempo de ejecución del algoritmo disminuya considerablemente.

El algoritmo utilizado para generar los grafos a partir de un grafo inicial se puede apreciar en la **¡Error! o se encuentra el origen de la referencia.**.

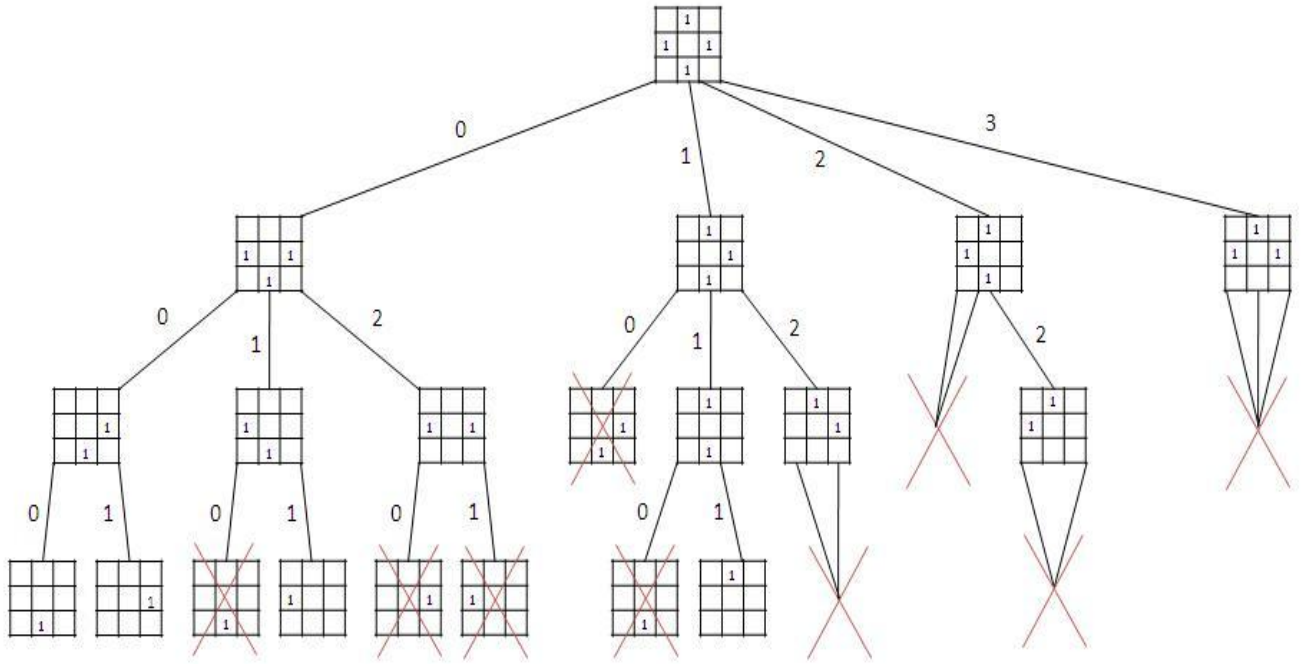

Figura 25. Algoritmo utilizado para generar lo grafos.

En la **¡Error! No se encuentra el origen de la referencia.** se puede observar que se toma el grafo original se le va quitando una arista a la vez, esto genera nuevos grafos que van ser parecidos al inicial, de manera que a todos estos primeros que se generaron a partir del inicial nada más les va a faltar una arista del grafo inicial, siendo diferente la arista que les va a faltar en cada grafo generado. Además el proceso realizado para generar los primeros grafos a partir del grafo inicial se vuelve a aplicar nuevamente a cada grafo generado hasta que se llegue a un grafo que contenga una sola arista. Todo este proceso de generación de grafos se lleva a cabo utilizando un algoritmo recursivo que utiliza la técnica de Vuelta Atrás. También en esta figura se puede evidenciar que en este proceso de generación de grafos pueden generarse grafos que ya habían sido generados y por tanto es necesario que no se vuelvan a generar. Esto se resolvió asignándole un número a cada grafo que se genere, el cuál va indicar el orden de generación que tiene el grafo con respecto al grafo que lo generó y a la vez va a servir para indicarle a ese grafo que no genere a partir de él, esos primeros grafos que representa ese número. De esta manera se lograr evitar que se generen grafos repetidos.

Una vez representada la forma en que se generan los grafos en el algoritmo se hizo necesario estudiar la manera en que se podía aplicar programación paralela al mismo, esto se logró creando un

hilo de ejecución por cada grafo que se genere a partir del grafo original, como se puede observar en la **Figura 26.**

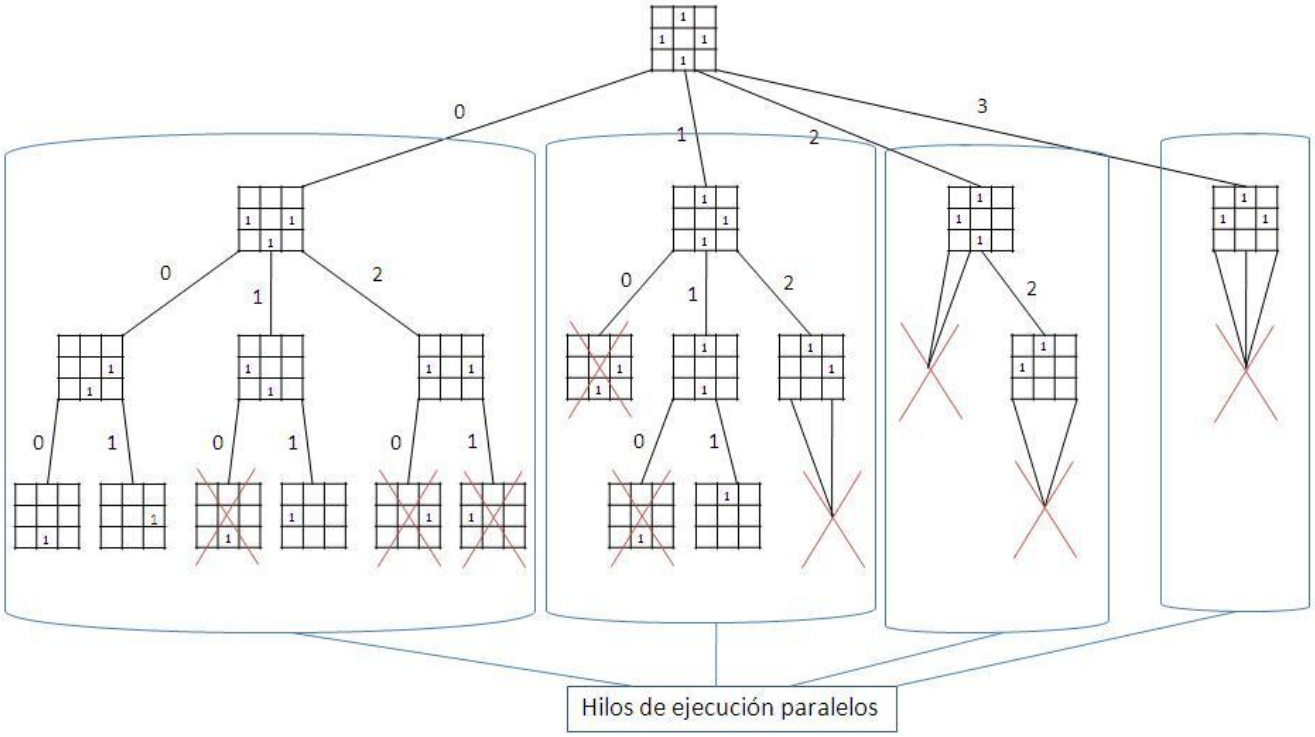

Figura 26. Algoritmo de generación de grafos utilizando programación paralela.

En la **Figura 26** se puede observar que cada grafo generado a partir del original se encuentra separado por diferentes hilos de ejecución. Para poder aplicar el paralelismo se hizo uso de la clase Thread de java, la cual es la que va a permitir que se pueda ejecutar la generación de los grafos paralelamente. De esta forma se logró reducir el tiempo considerablemente, aunque esta reducción esta en dependencia del servidor donde se ejecute el algoritmo debido a que mientras más procesadores tenga el mismo, más rápido será la ejecución del algoritmo y menos tiempo demorará.

Para comprobar la reducción del tiempo de ejecución del algoritmo utilizando la programación paralela, se volvieron a realizar las pruebas de tiempo ejecutadas anteriormente, en la cual para un grafo inicial de 20 aristas es necesario generar 1 048 576 grafos, se demoró aproximadamente 4 segundos y para un grafo inicial de 21 aristas donde es necesario generar 2 097 152 grafos, se demoró aproximadamente 8 segundos. Con la prueba realizada queda demostrado que utilizando la programación paralela el tiempo de ejecución se reduce casi a la mitad que la de las pruebas realizadas anteriormente sin utilizar programación paralela.

3.6 Conclusiones parciales

De las actividades de implementación y pruebas realizadas en este capítulo se obtuvieron los siguientes resultados:

- ✓ Se realizó el diagrama de componentes, el cual permite tener una mejor comprensión acerca de cómo estará compuesta la aplicación y tener conocimiento de que componentes se pueden reutilizar en futuras mejoras realizadas a la aplicación.
- ✓ Se realizó el diagrama de despliegue, con el objetivo de conocer los componentes necesarios para poder poner en explotación la herramienta desarrollada.
- ✓ Se le realizaron pruebas de caja negra a la aplicación, mediante casos de prueba, lo que permitió comprobar que la herramienta cumple con todas las funcionalidades requeridas y que responde correctamente en cuanto a los datos de entrada y de salida y en caso de errores.
- ✓ Se realizó una prueba en ambiente real, para determinar si la precisión de los algoritmos matemáticos que ejecuta la aplicación son los esperados para la toma de decisiones de tipo arquitectónico, obteniéndose resultados satisfactorios además de una mayor confiabilidad y rapidez en la ejecución de dichos algoritmos.
- ✓ Se utilizó programación paralela en la implementación del algoritmo para realizar el análisis de interrelaciones de atributos de calidad, lo que permitió disminuir considerablemente el tiempo de respuesta de dicho algoritmo.

Conclusiones generales

Una vez finalizado este trabajo de diploma, se puede arribar a las siguientes conclusiones:

- ✓ El método de evaluación QUASAR, fue propuesto para realizar el proceso de evaluación arquitectónica de Cedrux, debido a sus amplias ventajas, sin embargo por las deficiencias que presenta, se hizo necesario realizarle un grupo de mejoras que van encaminadas a erradicar las mismas.

- ✓ Se realizó un estudio acerca de las herramientas existentes para evaluar arquitecturas, determinándose que ninguna de las existentes brindaba los indicadores necesarios para darle cumplimiento a las mejoras propuestas para eliminar las deficiencias del QUASAR, por lo que se decidió la creación de una herramienta que brinde estas funcionalidades.

- ✓ Se definió la metodología y herramientas que se utilizarán para el desarrollo de la nueva herramienta, con el objetivo de que esta sea web, además de especificarse todas las funcionalidades que debe brindar.

- ✓ Se realizó el diseño de la herramienta a implementar, validando el mismo teniendo en cuenta las métricas TOC y RC, las cuales demostraron por los resultados arrojados que el diseño de la herramienta tiene una buena calidad.

- ✓ Se realizó la implementación de la herramienta, y para comprobar que la misma cumple con los requisitos funcionales propuestos se realizaron las pruebas de caja negra, demostrándose que la misma cumple con todas las funcionalidades definidas.

- ✓ Se demostró a través del juego de datos utilizado, que las funcionalidades basadas en métodos matemáticos y algoritmos clásicos de programación de la herramienta arrojan los datos precisos, contribuyendo así a erradicar los problemas que trae la aplicación manual de estos métodos y algoritmos en el proceso de evaluación de Cedrux.

Como conclusión de estos resultados, se logró dar respuesta al objetivo general de la presente investigación, el cual consistía en desarrollar una herramienta para el análisis de interrelaciones de atributos de calidad que contribuya a la precisión de los datos para la toma de decisiones en la evaluación de la arquitectura de Cedrux.

Recomendaciones

Aunque con la realización de la nueva herramienta, se cumplió el objetivo general de esta investigación, se recomienda:

- ✓ Si es posible, optimizar los algoritmos para un mejor funcionamiento interno del sistema.
- ✓ Continuar el desarrollo de la aplicación agregándole nuevas funcionalidades, que ayuden a mejorar y agilizar el proceso de evaluación arquitectónica de Cedrux o de cualquier otro sistema.

Bibliografía

Barbacci, M, y otros. *Quality Attributes. Carnegie Mellon University Technical Report.*1995**.**

Bass, L, Clements, P y Kazman, R. *Software Architecture in practice.Addison-Wesley.*1998.

Bosch, Jan. *Software Architecture Assessment.* [ppt] Netherlands : University of Groningen.2001.

Camacho, Erika, Cardeso, Fabio y Nuñez, Gabriel. *Arquitecturas de software. Guía de estudio.* abril de 2004.

Carrillo, Isaías., Perez, Rodrigo., Rodriguez, Aureliano. *Metodologias de desarrollo de software.* 2008.

CASELLES, J. [En línea] **2008**. http://www.coninteres.es/sql/material/Proceso_de_Normalizacion.pdf.

Date, J.C. *Introducción a los Sistemas de Base de Datos*. **2003.**

Escalona, M.J y Koch, N. *Ingeniería de Requisitos en Aplicaciones para la Web*: *Un estudio comparativo.*2008.

Firesmith, Donald. *QUASAR: A Method for the Quality Assessment of Software-Intensive System Architectures.* [pdf] Pittsburgh : Carnegie Mellon. Software Engineering Institute. , 2006. CMU/SEI-2006-HB-001.

—.*Quality Assessment of System Architectures and their Requirements (QUASAR) Version 3.0.* [ppt] Pittsburgh : Carnegie Mellon University. Software Engineering Institute, 2008. PA 15213.

Garlan, D yAllen, R. *The Wright Architectural Description Language*. Carnegie Mellon University: Technical Report. 2006

Garriazo, Luna y Stefany, Janet. *Trabajo de investigación "Navegadores".* Perú: Lima. 2011.

Gómez, Omar. *Evaluando la Arquitectura de Software. Parte 1. Panorama General.* México : Brainworx S.A. de C.V., enero-febrero de 2007.

Kazman, R, Clements, P y Klein, M. *Evaluating Software Architectures. Addison Wesley.* 2001.

López, Charl., Villanueva, Sergio., Rodríguez, Gerson., Ganoza, Juan., Taboada, Orlando. *PostgreSQL: Papel sobre el gestor de datos PostgreSQL, uso e instalación.*2011.

Losavio, F., Chirinos, L., Lévy, N., Ramdane-Cherif, A. *Quality Characteristics for Software Architecture*.2003

Bibliografía

Marca, Hugo y Quisberti, Nancy. *ANALISIS Y DISEÑO DE SISTEMAS II, Diagrama de despliegue.* 2000.

Morales, José Raúl Perera. *Arquitectura de software para sistema gestión de inventarios.* Ciudad de la Habana: s.n., 2007.

Montesino Afonso, Yasmany y Parra Lubín, Orestes. *Implementación del submódulo de Recuperaciones de Información del Módulo de Recuperaciones Dinámicas. [Digital]*Ciudad de la Habana: Universidad de las Ciencias Informáticas, 2008.

Panovski, Gregor. *Master's Thesis. Product Software Quality.* [pdf] Eindhoven : Department of Mathematics and Computing Science.Technische Universiteit Eindhoven, February de 2008.

Pressman R. *Ingeniería de Software. Un Enfoque Práctico*. Quinta Edición. McGraw Hill. 2002

Reynoso, C. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.*s.l.: Universidad de buenos aires,2004.

Salvador Gómez, Omar.*Evaluando Arquitecturas de Software. Parte 1. Panorama General.* México: Brainworx S.A, 2007**.** 1870-0888.

Vera, Angelo Benvenuto. *Implementación de Sistemas ERP, su impacto en la gestión de la empresa e integración con otras TIC.* 2006.

Vigil, Yamila Regalado. *Metodología de evaluación de arquitectura de software.* [pdf] Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2009.