Universidad de las Ciencias Informáticas

Facultad 9

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

TÍTULO: Sistema para la Certificación de Arquitecturas de Software basado en los métodos ATAM y SAAM para los proyectos productivos de la Universidad de las Ciencias Informáticas.

AUTORES:

Michael Zapata Salazar

Naivis Rosa González Gamez

TUTOR:

Ingeniero en Ciencias Informáticas, Armando Ortiz Cabrera

Ciudad de la Habana, 29 Junio 2010.

Año 52 de la Revolución.

Dedico el presente trabajo de diploma antes que todo, a mis padres Irma Gamez y Roberto Ignacio González por su amor y confianza, por el apoyo incondicional, moral y económico que me han brindado a lo largo de todos estos años. A mis hermanos Andros Daniel y Roberto por ser siempre un ejemplo de lucha y superación. A madrina que ha sido siempre mi abuelita querida al igual que Mamá que aunque ya no está, sé que estaría orgullosa de mí.

Naivis Rosa.

Les dedico este trabajo de diploma a mis amistades y familiares, en especial a mis padres Lidia Idalys Salazar y Alfonso Zapata, y a mi querida abuela Ana América Figueroa por ser muy importantes en mi vida, por haberme brindado su amor, dedicación y apoyo incondicional.

Michael.

Este trabajo no es el producto de la labor individual, sino más bien el resultado del esfuerzo de muchas personas durante los años de formación; para ellos nuestro recuerdo y agradecimiento, en especial:

A mis padres y hermanos.

A mis compañeros, familiares y amigos por el tiempo compartido.

A Yenis, René, Marisleisdis, Jheisy, Lizandra y Yosbel.

A todas las personas que de una forma u otra han contribuido en mi formación como profesional.

A mi novio que es también mi compañero de tesis por todo el amor y apoyo brindado durante estos años, a ti mi amor un beso grande.

" A todos con amor y cariño los recordare" ♥

Naivis Rosa

Agradezco a todos los que me ayudaron en la realización del presente trabajo de diploma, ya sea de forma directa o indirecta.

A mis amistades, compañeros y profesores, por haber compartido a mi lado momentos importantes de nuestras vidas.

A mis familiares, en especial a mis padres por haberme educado y haber hecho de mí una mejor persona.

No quisiera terminar sin antes agradecer a mi novia y compañera de tesis por haber estado a mi lado, y por haberme dedicado gran parte de esta corta vida, por haberme ayudado a tener una mejor forma de pensar y por hacerme feliz durante estos cinco años de universidad.

Michael

Declaramos que somos los únicos autores de este trabajo, y autorizamos a los proyectos productivos de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los _29_ días del mes de _junio_ del año _2010_.

Autores:

Naivis Rosa González Gamez

_____

Michael Zapata Salazar

_____

Tutor:

Armando Ortiz Cabrera

_____

Tutor:

Nombre y apellidos: **Armando Ortiz Cabrera**

Sexo: M

Institución: **Universidad de las Ciencias Informáticas (UCI)**

Dirección de la institución: **Carretera San Antonio de Baños Km 2 ½, Boyero**

Correo electrónico: **aortizc@uci.cu**

Teléfono del trabajo: **837 2557** Teléfono particular: **835 8894**

Categoría docente: **Instructor**

Grado científico: Cargo del trabajador: **Jefe del Grupo de Asesoramiento Técnico a Proyectos, Polo Productivo PETROSOFT.**

Título de la especialidad de graduado: **Ingeniero en Ciencias Informáticas**

Año de graduación: **2007**

Institución donde se graduó: **Universidad de las Ciencias Informáticas.**

RESUMEN

La selección de una arquitectura de software (AS), define elementos arquitectónicos que influirán directamente en los requerimientos no funcionales de un sistema de software. Por tal razón la evaluación de la AS es un procedimiento de gran importancia que tiene como objetivo determinar en qué grado la arquitectura de un software, satisface los atributos de calidad que se especificaron en los requerimientos no funcionales de los clientes. En la Universidad de las Ciencias Informáticas (UCI) no existe un sistema que informatice el procedimiento de certificación de la arquitectura de un producto, que pueda ser utilizado por los proyectos de la universidad; es por ello que surge la necesidad de desarrollar un sistema que cumpla con tales características, basado en los métodos de evaluación que han sido identificados como factibles para evaluar la arquitectura en los proyectos productivos de la UCI. La esencia de este trabajo de diploma radica precisamente en el desarrollo de un sistema para la certificación de arquitecturas de software, basado en los métodos de evaluación ATAM y SAAM con el propósito de garantizar una mejora en la calidad de este procedimiento evaluativo.

PALABRAS CLAVES

- ✓ Arquitectura de software.
- ✓ Evaluación de arquitectura de software.
- ✓ Atributos de calidad.
- ✓ Métodos de evaluación.

ÍNDICE _TABLAS

ÍNDICE _FIGURAS

INTRODUCCIÓN

Con el avance de las Tecnologías de la Información y la Comunicación (TIC) y las necesidades actuales que tiene toda organización para el logro de sus objetivos, han surgido nuevas necesidades de construir grandes y complejos sistemas de software que requieren de la combinación de diferentes tecnologías y plataformas de hardware y software para alcanzar un funcionamiento acorde con dichas necesidades. Lo anterior, exige de los profesionales dedicados al desarrollo de software poner especial atención y cuidado al diseño de la arquitectura, bajo la cual estarán soportadas las funcionalidades del sistema.

Diseñar una AS es una de las actividades fundamentales de todo el procedimiento que se lleva a cabo en la construcción de un software, su selección debe estar en correspondencia con los requerimientos no funcionales del sistema. Es por ello que se dice que la AS incide de gran manera en la calidad de un sistema, por lo que es de suma importancia realizar su evaluación con el fin de encontrar o determinar los posibles riegos y sus respectivas soluciones antes de comenzar a programar; garantizando la calidad arquitectónica del sistema.

Situación Problemática

La UCI es una institución desarrolladora de software surgida como un proyecto de la Batalla de Ideas, con el objetivo de informatizar la sociedad y ampliar las posibilidades de competir en el mercado en la producción de software y así aportar ganancias al país e impulsar su desarrollo. Este centro está constituido por facultades cada una de ellas tiene polos productivos y a ellos se asocian una serie de proyectos. Estos polos son considerados célula fundamental del trabajo de formación, investigación y desarrollo para poder aportar a la producción en determinada área del conocimiento. Los proyectos asociados a los diferentes polos productivos han visto afectada su producción por numerosas deficiencias que presenta los productos finales que se han desarrollado; las cuales son provocadas, entre otras cosas, por la no realización de la evaluación de la arquitectura para poder detectar los errores a tiempo. Todo esto trae como consecuencia que la evaluación de las arquitecturas de los productos no sea una actividad lo suficientemente implementada, documentada y especificada durante el ciclo de desarrollo de software.

Estos elementos atentan directamente en contra del logro de los atributos de calidad exigidos por los clientes; gastos de recursos y tiempo en el desarrollo de sistemas, cuya arquitectura es incapaz de alcanzar los requerimientos funcionales; atrasos en los proyectos de desarrollo, con lo cual la calidad del producto final obtenido resulta afectada porque las deficiencias se descubren demasiado tarde y todo el proyecto es arrojado al caos. **(1)**

En la UCI no existe un sistema que informatice el procedimiento de certificación de la AS, que pueda ser utilizado por los proyectos productivos de la universidad, es por ello que surge la necesidad de desarrollar un sistema que cumpla con tales características, basado en los métodos de evaluación ATAM y SAAM que han sido identificados como factibles para evaluar la AS en estos proyectos.

Dada la situación problemática anteriormente planteada se identifica el siguiente **problema científico:**
¿Cómo mejorar el procedimiento de certificación de arquitectura en los proyectos productivos de la UCI?
Luego de ser analizados los antecedentes del problema existente se define como **objetivo general** de la investigación:
Desarrollar un sistema que informatice el procedimiento de certificación de AS basado en los métodos ATAM y SAAM.
Para darle cumplimiento a dicho objetivo se plantea como **objeto de estudio:**
El procedimiento de certificación de arquitecturas.
Para precisar el objeto de estudio se define como **campo de acción:**
Procedimiento de certificación de arquitectura según los métodos de evaluación ATAM y SAAM.
Para sustentar esta investigación, se formula la siguiente **hipótesis**:
El desarrollo de un sistema que informatice el procedimiento de certificación de la AS según los métodos de evaluación ATAM y SAAM, permitirá mejorar la calidad del proceso de certificación en los proyectos productivos de la UCI.
Como vías para cumplir el objetivo propuesto se determinaron las siguientes **tareas de investigación:**

1. Definir el marco teórico y metodológico en el que está enmarcado el sistema.
2. Seleccionar aspectos fundamentales de interés en los fundamentos teóricos de los métodos de evaluación de arquitecturas ATAM y SAAM.
3. Caracterizar las tecnologías a usar en la realización del sistema.

4. Diseñar la solución propuesta.
5. Implementar el producto de acuerdo con el diseño elaborado.
6. Validar la solución propuesta.

El **resultado** que se espera de la investigación es:
El diseño e implementación de un sistema que informatice la certificación de arquitecturas basado en los métodos de evaluación ATAM y SAAM para los proyectos productivos de la UCI y la documentación de los artefactos ingenieriles que se generen.
Para la obtención de información relacionada al problema planteado en el desarrollo de la investigación se utilizaron los siguientes **métodos científicos:**
Teóricos:
Histórico lógico: Se utiliza para determinar si actualmente existen sistemas informáticos que cumplan con tales características. Una vez realizada la investigación se llega a la conclusión de que no se cuenta con dicho sistema.

Análisis y síntesis: Se utiliza en el estudio y valoración de las técnicas y métodos de evaluación de AS que fueron seleccionadas como factibles para ser usadas en la UCI, permitiendo extraer de la fundamentación teórica de estos métodos todos aquellos elementos que se relacionan con el objeto de estudio. Se logra además analizar y determinar las partes que integran la investigación, unir los datos, definir cada elemento que está estrechamente relacionado con otro y darle solución al problema.

Empíricos:
Revisión de documentos: Se revisaron documentos relacionados con el objeto de estudio. Como documentos emitidos por el Instituto de Ingeniería de Software (SEI), estas siglas en inglés indican Software Engineering Institute, donde se exponen casos de estudio, que utilizan estos métodos de evaluación de arquitecturas.

Para una mejor organización y distribución el presente trabajo de diploma está estructurado en cuatro capítulos. Además, contiene las conclusiones generales, recomendaciones, bibliografía consultada y citada, un glosario de términos y por último los anexos; todos estos aspectos forman el cuerpo del trabajo y son imprescindibles para un mejor entendimiento. El contenido de los capítulos se muestra a continuación:

Capítulo 1. Fundamentación teórica:

Se expone el estado del arte del objeto de estudio de la presente investigación, definiéndose los elementos teóricos que la sustentan. Se enuncian conceptos que posibilitan un mejor entendimiento de lo planteado en la situación problemática y el marco del problema en general.

Capítulo 2. Tendencias y tecnologías actuales a desarrollar:

En este capítulo se describe el lenguaje que proporciona soporte a la modelación de la solución propuesta, la metodología que guía el proceso de desarrollo del software y se argumenta la causa de su utilización. Se exponen además las características de la herramienta CASE, estas siglas en inglés indican (Computer Assisted Software Engineering) y su traducción al español significa Ingeniería de Software Asistida por Computación, el lenguaje de programación, el framework de desarrollo y el sistema gestor de base de datos seleccionado para la implementación de la solución propuesta.

Capítulo 3. Presentación de la solución propuesta:

Se exponen los principales conceptos del entorno en el que se encuentra enmarcado el sistema, así como los artefactos generados de los flujos de trabajos: Modelo de Dominio, Diagrama de Caso de Uso, Descripción de los Casos de Uso y de los Actores del Sistema. Se especifican los Requerimientos Funcionales (RF) y los Requerimientos No Funcionales (RNF), que debe tener el sistema, con el objetivo de satisfacer las restricciones y necesidades del mismo.

Capítulo 4. Construcción de la solución propuesta:

Este capítulo presenta el diseño, la implementación y la prueba del sistema realizado, así como el diagrama de despliegue. Partiendo de los requisitos, se realiza el diagrama de clases del diseño y de los resultados obtenidos la implementación, para dar entrada finalmente a la etapa de pruebas; en esta última se realiza el diseño de los casos de pruebas para los casos de uso que se determinaron de gran importancia para el sistema en general.

Capítulo

Fundamentación Teórica

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se muestran los elementos teóricos que sustentan el trabajo científico y sus objetivos. Se sientan las bases de la AS, así como se analizan las principales corrientes teóricas y se examina la teoría que la origina. Se enuncian conceptos que posibilitan un mejor entendimiento de lo planteado en la situación problemática y el marco del problema en general.

1.2 Conceptos asociados al dominio del problema

Para un mejor entendimiento de lo que se plantea en la situación problemática es necesario enunciar algunos conceptos asociados al dominio del problema.

1.2.1 Arquitectura de software

Los antecedentes de la AS se remontan a la década de 1960 con las tempranas inspiraciones de Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, quien hacia 1968, aunque no utilizó el término arquitectura de software para describir el diseño conceptual del software, sentó las bases de esta disciplina al proponer que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera. **(2)**
Arquitectura Software = {Elementos, Formas, Fundamento o Restricciones} **(2)**

Diversas son las definiciones que los autores del campo han publicado hasta el momento de la arquitectura de software, aunque ninguna de estas, al parecer, han sido totalmente aceptadas. Al hacer un análisis detallado de cada uno de los conceptos disponibles, resulta interesante la existencia de ideas comunes entre los mismos, sin observarse planteamientos contradictorios, sino más bien complementarios. De aquí que la mayoría de los autores coinciden en que una arquitectura de software define la estructura del sistema.

Esta estructura se constituye de componentes o elementos arquitectónicos (estilos y patrones) que nacen de la noción de abstracción, cumpliendo funciones específicas, e interactuando entre sí con un comportamiento definido (ver anexos: imagen 1). **(1)**

La definición oficial de AS es la IEEE Std 1471-2000 que reza así: “La Arquitectura del Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución”; esta es la definición que se mantendrá a lo largo de esta investigación, por ser una definición básica adoptada por todo el mundo y por todas las organizaciones que tienen que ver con la AS.

Los autores de este trabajo coinciden con la idea de que la arquitectura de software es la estructura de un sistema. La misma debe ser seleccionada en los primeros momentos de la realización y antes de la implementación de un sistema, de forma tal que se cumplan con los RF y RNF. La misma recoge los elementos más importantes de un sistema como sus componentes y el vínculo o relación que existe entre ellos.
Sin dudas la década de 1990 fue el marco de tiempo del florecimiento de la AS y la consolidación y diseminación en una escala sin precedentes; en esta época surgen los estilos y patrones. **(2)**

1.2.2 Patrones

Desde la década de los 70 surgen las primeras definiciones de patrones con Christopher Alexander, quien incidentalmente fue arquitecto de edificios. Una de las ideas brillantes de la década de los 90 fue vincular las estructuras recurrentes, las soluciones recurrentes a problemas en determinados contextos con esta idea de Christopher Alexander, referida a la arquitectura real. **(2)**

"Cada patrón describe un problema que ocurre infinidad de veces en nuestro entorno, así como la solución al mismo, de tal modo que podemos utilizar esta solución un millón de veces más adelante sin tener que pensarla otra vez". Esta definición se refiere a patrones arquitectónicos de arquitecturas reales (arquitecturas de edificios y ciudades). **(2)**

Un patrón es la reutilización de la solución a un problema que se repite una y otra vez. Las clases de patrones que existen son muy variadas, es por ello que existen patrones que describen todo tipo de soluciones, que comprender desde el análisis hasta el diseño y desde la arquitectura hasta la implementación.

1.2.2.1 Patrones de Arquitectura.

Los patrones arquitectónicos representan los patrones de más alto nivel en nuestro sistema de patrón. Ellos le ayudan a especificar la estructura fundamental de una aplicación. Cada actividad de desarrollo que sigue se rige por esta estructura, por ejemplo, el diseño detallado de los subsistemas, la comunicación y colaboración entre las diferentes partes del sistema, y su posterior ampliación. Los patrones de arquitectura (ver anexos: imagen 3), que ayudan a apoyar propiedades similares, pueden ser agrupados en categorías: **(3)**

Del barro a la estructura: Ayudan a evitar un "mar" de componentes u objetos. Soportan una controlada descomposición de una tarea global del sistema en la cooperación de subtareas. La categoría incluye los siguientes patrones:

- Patrón de Capas
- Patrón Tubería y Filtros
- Patrón Pizarra
- Patrón Rompedor

Sistemas Interactivos: Esta categoría comprende dos patrones, el patrón Modelo-Vista-Controlador (MVC) y el patrón Presentación-Abstracción-Control (PAC). Ambos patrones soportan la estructuración de sistemas de software que presentan interacción persona-ordenador.

- Patrón MVC
- Patrón PAC

Sistemas Adaptables: El patrón Reflexión y el patrón Micronúcleo soportan fuertemente la extensión de las aplicaciones y su adaptación a la evolución de la tecnología y la evolución de las necesidades funcionales.

- Reflexión
- Micronúcleo

1.2.2.2 Patrones de Diseño.

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. **(4)**

Los patrones de diseño (ver anexos: imagen 4) se agrupan por categorías, según sus propósitos, entre las que podemos encontrar: **(3)**

Patrones de Descomposición Estructural: Trata de la descomposición de subsistemas y componentes complejos en partes cooperadoras.

- Todo-Parte
- Composición

Patrones de Organización de Trabajo: Definen cómo los componentes deben colaborar juntos para resolver un problema complejo.

- Maestro-Esclavo
- Cadena de responsabilidad
- Mediador

Patrones de Control de Acceso: Estos son patrones de vigilancia y control de acceso a los servicios de componentes.

- Proxy
- Fachada
- Iterador

Patrones de Administración: Definen las pautas para el manejo de colecciones homogéneas de objetos, servicios y componentes de su totalidad.

- Procesador de Instrucciones (Comandos)
- Manipulador de Vistas
- Memento
- Patrones de Comunicación
- Emisor-Receptor
- Cliente-Despachador-Servidor

Existe una analogía muy fuerte entre un patrón de arquitectura e incluso un patrón de diseño y un estilo arquitectónico. Una vez que se han decidido cuales son los estilos que ha de configurar una determinada solución, los patrones que existen relacionados a esos estilos ayudaran a derivar de esa formulación arquitectónica el diseño y posteriormente la programación correspondiente.

1.2.3 Estilos Arquitectónicos

Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. Los estilos conjugan elementos o “componentes”, conectores, configuraciones y restricciones. Los estilos sirven para sintetizar estructuras de soluciones. Pocos estilos abstractos encapsulan una enorme variedad de configuraciones concretas. Definen los patrones posibles de las aplicaciones. **(2)**

Estilos arquitectónicos más comunes

Dentro de los principales estilos arquitectónicos (ver anexos: imagen 2), se pueden identificar un conjunto de subgrupos que recogen sistemas muy específicos: **(2)**

Estilos de flujos de datos: Sucesión de transformaciones de los datos de entrada.

- Sistemas de filtros y tuberías.
- Sistemas de procesamiento por lotes.

Estilos basados en llamada y retorno: Son sistemas que reflejan la estructura del lenguaje de programación.

- Sistemas de programación principal y subrutina.
- Modelo – Vista – Controlador(MVC)
- Sistemas orientados a objetos.
- Sistemas organizados en capas.
- Sistemas basados en componentes

Estilos de componentes independientes: Grupo de sistemas cuyos componentes se comunican mediante el paso de mensajes.

- Sistemas cliente/servidor.
- Sistemas de eventos.
- Sistemas orientados a servicios.

Estilos centrados en los datos: Estos sistemas se caracterizan por el acceso compartido a un banco de datos central.

- Repositorios (datos pasivos).
- Pizarras (datos activos).

Estilos de máquinas virtuales: Los sistemas bajo estos estilos simulan una funcionalidad no nativa del entorno.

- Intérpretes.
- Sistemas basados en reglas.

Estilos heterogéneos: Son sistemas que combinan características de varios estilos.

- Sistemas de control de procesos
- Arquitecturas basadas en atributos.

1.2.4 Calidad del software

Según Roger Pressman (1992) la calidad de un software es:
"Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente".

Según la IEEE, Std. 610-1990
La calidad de un software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario.
Según los autores de este trabajo cuando se habla de calidad de un software se hace referencia a la capacidad o propiedad inherente a un producto software para satisfacer o cumplir con los requerimientos y expectativas del cliente o usuario.

1.2.5 Atributo de calidad

La calidad de software se define como el grado en el cual el software posee una combinación deseada de atributos. Tales atributos son requerimientos adicionales del sistema, que hacen referencia a características que este debe satisfacer, diferentes de los requerimientos funcionales. Estas características o atributos se conocen con el nombre de atributos de calidad, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios. **(1)**
Por otro lado, los atributos de calidad son los aspectos del sistema, que en general, no afectan directamente a la funcionalidad necesaria, sino que definen la calidad y las características que el sistema debe soportar.

Los atributos de calidad se clasifican en dos categorías:
Observables vía ejecución: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución. Ver anexo 5

No observables vía ejecución: aquellos atributos que se establecen durante el desarrollo del sistema. Ver anexo 6

1.3 Evaluación de la arquitectura

Evaluar una arquitectura consiste en verificar si la arquitectura diseñada cumple con los atributos de calidad requeridos. La evaluación de la arquitectura es una actividad que tiene como objetivo medir propiedades de un sistema. Además, analiza si la arquitectura cumple con los requerimientos no funcionales o atributos de calidad de un sistema y de esta forma verificar si el mismo satisface las necesidades de los clientes, de ahí la importancia y esencia de la evaluación. La evaluación de la AS, sirve para prevenir los posibles desastres de un diseño que no cumple los requerimientos de calidad aunque no dice si una arquitectura es buena o mala, solo identifica donde están los riesgos, las fortalezas y debilidades.

La evaluación de las arquitecturas de software puede ser realizada mediante el uso de diversas técnicas y métodos las cuales se muestran a continuación:

1.3.1Técnicas de evaluación

Las técnicas existentes en la actualidad para evaluar arquitecturas permiten hacer una evaluación cuantitativa sobre los atributos de calidad a nivel arquitectónico, sin embargo, debido al costo de realizar este tipo de evaluación en muchos casos los arquitectos del software evalúan cualitativamente. Para decidir entre las alternativas de diseño existen diferentes técnicas de evaluación: basada en escenarios, basada en simulación, basada en modelos matemáticos y basada en experiencia **(1)**

Escenarios:

Los escenarios son, al igual que los casos de usos, frases que describen un requerimiento. A diferencia de los casos de usos, los cuales tiene que ver con la funcionalidad, los escenarios se refieren a variables no funcionales. **(1)**

Técnica de evaluación a utilizar.

De las técnicas antes mencionadas se utiliza la basada en escenario, por las ventajas que esta técnica posee como el bajo costo, la efectividad y la facilidad para ser aplicada. Además, los escenarios permiten de una manera más fácil concretar y entender los atributos de calidad**. (1)** Los métodos ATAM y SAAM aplican esta técnica y por todas las ventajas que implica su aplicación son los métodos de evaluación escogidos para guiar el proceso evaluativo que se desea automatizar.

1.3.2 Métodos de evaluación

Un método de evaluación sirve de guía a los involucrados en el desarrollo de un sistema para la búsqueda de conflictos que pueden presentar una arquitectura y sus soluciones. **(5)**

Método de Análisis de Acuerdos de Arquitectura (ATAM)

Es un método de evaluación de arquitectura basada en escenarios para evaluar los atributos de calidad tales como: modificabilidad, portabilidad, extensibilidad, integrabilidad y rendimiento. Además, de la evaluación de atributos de calidad, ATAM explora interacción de los atributos de calidad y sus

interdependencias destacando el comercio fuera de los mecanismos y oportunidades entre diferentes calidades. **(1)**

El método consta de nueve pasos principales los cuales son:

Paso 1-Presentar el ATAM.

Paso 2-Controladores de negocios actual.

Paso 3-Presentar arquitectura.

Paso 4-Identificar los enfoques de arquitectura.

Paso 5-Generar árbol de utilidad de los atributos de calidad.

Paso 6-Analizar enfoques de arquitectura.

Paso 7-LLuvia de ideas y la prioridad de escenarios.

Paso 8-Analizar enfoques de arquitectura.

Paso 9-Presentar los resultados.

Método de Análisis de Arquitectura de Software (SAAM)

El método SAAM es aplicable en los proyectos donde el atributo de calidad modificabilidad es el de mayor interés, ya que este método evalúa varios atributos de calidad como la flexibilidad, mantenimiento, portabilidad, extensibilidad, integridad y funcionalidad pero es la modificabilidad su objetivo principal y es el mejor para evaluar este atributo ya que provee escenarios de cambios para realizar la evaluación. **(1)**

El método consta de seis pasos principales los cuales son:

Paso 1 - Desarrollar escenarios.

Paso 2 - Describir la arquitectura.

Paso 3 - Clasificar y priorizar escenarios.

Paso 4 - Evaluar individualmente escenarios indirectos.

Paso 5 - Evaluar la interacción de los escenarios.

Paso 6 - Crear una evaluación global.

1.4 Conclusiones del capítulo

En este capítulo se ha visto diferentes aspectos del objeto de estudio y se mostraron algunos conceptos básicos para la comprensión del problema. Se exponen un conjunto de técnicas de evaluación de AS y se selecciona con la que se trabajará. Por el análisis realizado se ha llegado a la conclusión que a pesar de existir muchas técnicas y métodos que permiten evaluar AS no existe un sistema con tales características en la Universidad de las Ciencias Informáticas. Por todo este es necesario el desarrollo de una aplicación capaz de realizar este trabajo de forma automatizada y rápida, para así lograr mejores resultados en cuanto a la calidad del procedimiento evaluativo referido.

Capítulo

Tendencias y tecnologías actuales a desarrollar

CAPÍTULO II: TENDENCIAS Y TECNOLOGIAS ACTUALES A DESARROLLAR.

2.1 Introducción

El presente capítulo está conformado por varios epígrafes para dar respuesta a las tendencias y tecnologías actuales a desarrollar. Se describe el lenguaje que proporciona soporte a la modelación de la solución propuesta, la metodología que guía el proceso de desarrollo del software y se argumenta la causa de su utilización. Se exponen además las características de la herramienta CASE, el lenguaje de programación y el sistema gestor de base de datos seleccionado para la implementación de la solución propuesta.

2.2 El Lenguaje Unificado de Modelado (UML)

UML es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Se define como un lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software **(6)**. Ofrece soporte para clases, clases abstractas, relaciones, comportamiento por interacción, empaquetamiento, entre otros. Estos elementos se pueden representar mediante tres tipos de diagramas diferentes en los que agrupa UML todos los diagramas, los cuales muestran diferentes aspectos de las entidades representadas.

La figura 1 muestra como UML agrupa todos los diagramas en tres tipos diferentes. Los diagramas de estructura estática, dentro de los que se encuentran los diagramas de clases, los diagramas de objeto y los diagramas de caso de uso. También se encuentran los diagramas de implementación, donde se enmarcan los diagramas de componente y los de despliegue. Por último, se muestran los diagramas de comportamiento que son los diagramas de actividades, de estado y de interacción.

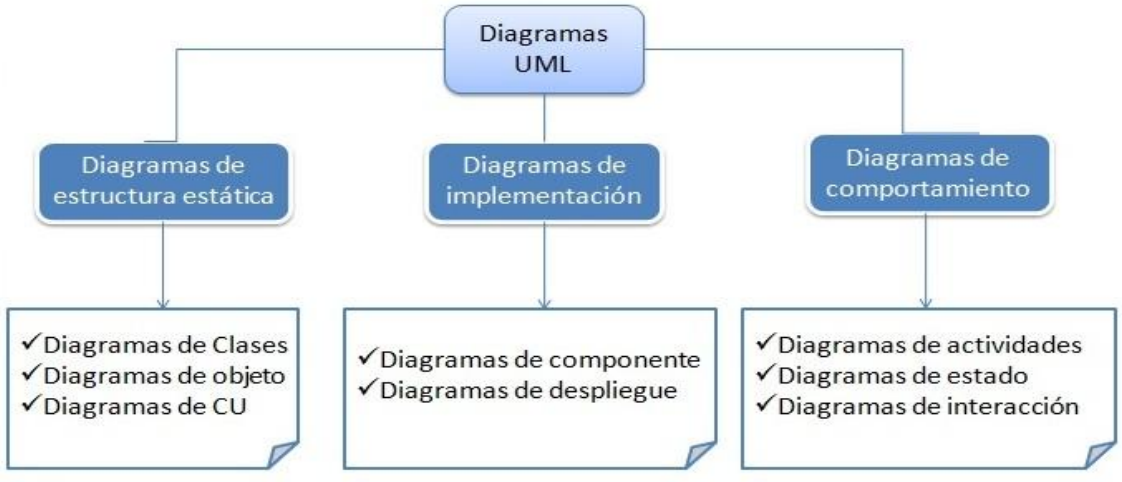

Figura 1 Diagramas UML.

Lo fundamental de una herramienta UML es la capacidad de diagramación, y los diferentes tipos de diagramas que soporta. Sus esquemas de apoyo de diseño, documentación, construcción e implantación de sistema. Así mismo, su flexibilidad para admitir cambios no previstos durante el diseño o el rediseño. En resumen, la herramienta ideal, es aquella que admite diseño desde inicio a fin, diseño inverso (o rediseño) y diseño vise-versa, con esquemas amplios para documentar detalladamente los procesos **(7)**. Por todas estas razones, es que en el presente trabajo, se usa este lenguaje de modelado como soporte de la modelación de la solución propuesta.

2.3. Metodología de desarrollo de software

En el proceso de construcción de un software se necesita contar con una metodología que guíe dicho proceso. En el momento de seleccionar una metodología para aplicar en la construcción de un sistema es necesario tener en cuenta las características del proyecto y del equipo y ser adaptada al contexto del mismo. Una de las características principales a tener en cuenta es la complejidad del sistema a desarrollar, es decir, es necesario valorar la complejidad del proceso a automatizar, la cantidad de requisitos que deben ser implementados en el sistema y la cantidad de información que se maneja en el proceso **(8)**. A la hora de elegir una metodología para desarrollar exitosamente los sistemas, normalmente, las personas se inclinan por aplicar la metodología estudiada, aunque existen otras que se adaptan mejor al proyecto.

2.3.1 Proceso Unificado Ágil (AUP)

AUP es una versión simplificada del Proceso Unificado de Rational (RUP). La AUP es la adopción de muchas de las técnicas ágiles de XP y otros procesos ágiles que mantiene de las RUP. Si se compara con la metodología XP, el AUP resulta ser un proceso muy pesado y en relación con RUP, resulta ser un proceso muy simplificado **(9)**, pues este último es considerado un proceso altamente ceremonioso que especifica muchas actividades y artefactos involucrados en el desarrollo de un proyecto software.

El proceso AUP establece un Modelo más simple que el que aparece en RUP por lo que reúne en una única disciplina las disciplinas de Modelado de Negocio, Requisitos y Análisis y Diseño. El resto de disciplinas (Implementación, Pruebas, Despliegue, Gestión de Configuración, Gestión y Entorno) coinciden con las restantes de RUP **(9)**.

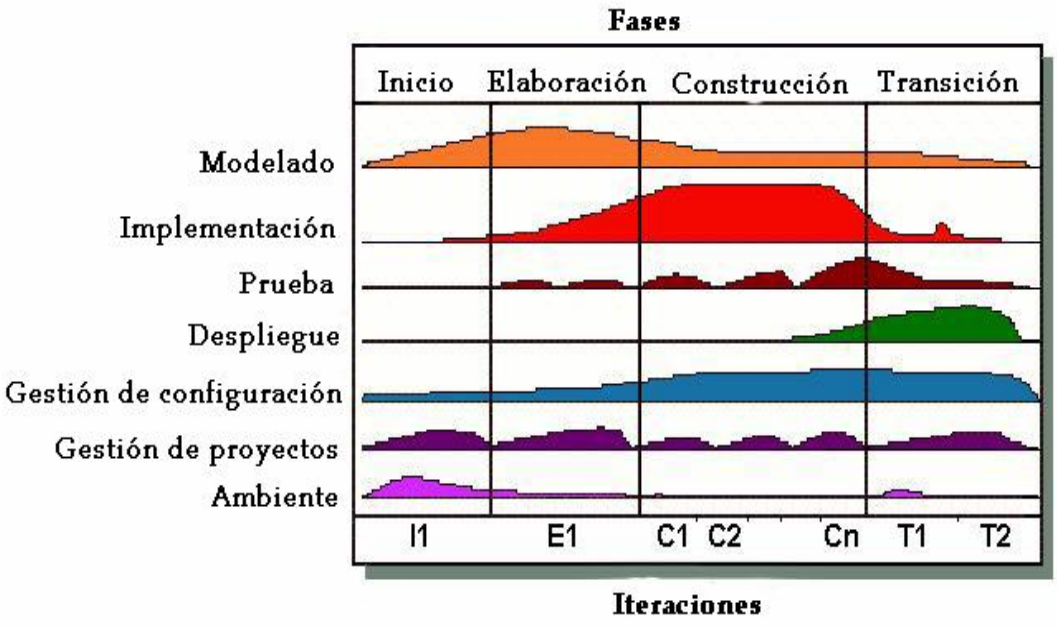

Figura 2 Disciplina, fases, iteraciones del AUP.

Ciclo de vida de (AUP).

Al igual que en RUP, en AUP se establecen cuatro fases que transcurren de manera consecutiva y que acaban con hitos claros alcanzados:

- Concepción: El objetivo de esta fase es obtener una comprensión común entre cliente y equipo de desarrollo sobre el alcance del nuevo sistema, para definir una o varias arquitecturas candidatas.
- Elaboración: El objetivo es que el equipo de desarrollo profundice en la comprensión de los requisitos del sistema y en validar la arquitectura.
- Construcción: Durante la fase de construcción el sistema es desarrollado y probado al completo en el ambiente de desarrollo.
- Transición: El sistema se lleva a los entornos de preproducción donde se somete a pruebas de validación y aceptación y finalmente se despliega en los sistemas de producción. **(9)**

Las disciplinas se llevan a cabo de manera sistemática, a la definición de las actividades que realizan los miembros del equipo de desarrollo a fin de desarrollar, validar, y entregar el software de trabajo que responda a las necesidades de sus interlocutores. Las disciplinas son:

1. Modelo. El objetivo de esta disciplina es entender el negocio de la organización, el problema de dominio que se abordan en el proyecto, y determinar una solución viable para resolver el problema de dominio.
2. Aplicación (Implementación). El objetivo de esta disciplina es transformar su modelo(s) en código ejecutable y realizar un nivel básico de las pruebas, en particular, la unidad de pruebas.
3. Prueba. El objetivo de esta disciplina consiste en realizar una evaluación objetiva para garantizar la calidad. Esto incluye la búsqueda de defectos, validar que el sistema funciona tal como está establecido, y verificando que se cumplan los requisitos.
4. Despliegue. El objetivo de esta disciplina es la prestación y ejecución del sistema y que el mismo este a disposición de los usuarios finales.
5. Gestión de configuración. El objetivo de esta disciplina es la gestión de acceso a herramientas de su proyecto. Esto incluye no sólo el seguimiento de las versiones con el tiempo, sino también el control y gestión del cambio para ellos.
6. Gestión de proyectos. El objetivo de esta disciplina es dirigir las actividades que se llevan a cabo en el proyecto. Esto incluye la gestión de riesgos, la dirección de personas (la asignación de tareas, el

seguimiento de los progresos, entre otros.), coordinación con el personal y los sistemas fuera del alcance del proyecto para asegurarse de que es entregado a tiempo y dentro del presupuesto.
7. Entorno (Ambiente). El objetivo de esta disciplina es apoyar el resto de los esfuerzos por garantizar que el proceso sea el adecuado, la orientación (normas y directrices), y herramientas (hardware, software, entre otros.) estén disponibles para el equipo según sea necesario **(9)**.

La AUP es ágil, porque está basada en los siguientes principios:
1. El personal sabe lo que está haciendo. La gente no va a leer detallado el proceso de documentación, pero algunos quieren una orientación de alto nivel y / o formación de vez en cuando. La AUP producto proporciona enlaces a muchos de los detalles, si usted está interesado, pero no obliga a aquellos que no lo deseen.
2. Simplicidad. Todo se describe concisamente utilizando un puñado de páginas, no miles de ellos.
3. Agilidad. El ajuste a los valores y principios de la Alianza Ágil.
4. Centrarse en actividades de alto valor. La atención se centra en las actividades que se ve que son esenciales para el de desarrollo, no todas las actividades que suceden forman parte del proyecto.
5. Herramienta de la independencia. Se puede usar cualquier conjunto de herramientas pero lo aconsejable es utilizar las más adecuadas para el trabajo, que a menudo son las herramientas simples o incluso de código abierto **(9)**.

2.3.2 Proceso Unificado Ágil como base en el desarrollo de la propuesta de solución.

Se ha seleccionado a AUP como la metodología indicada para la realización del sistema de certificación de arquitecturas de software por ser la que más se adapta al proyecto a desarrollar así como a las condiciones de trabajo. Se ajusta a las características de este proyecto porque evita los tortuosos y burocráticos caminos de las metodologías tradicionales enfocándose en la gente y los resultados.

Es muy apropiada para guiar proyectos de una complejidad y volumen no muy altos y que necesiten una rápida implementación por límite de tiempo, los cuales son los aspectos fundamentales a tener en cuenta para el desarrollo del sistema. En este caso se necesitaba una metodología liviana para el desarrollo de un software de poca complejidad, en corto tiempo y con poca documentación técnica. Esta metodología describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. Además, AUP como metodología ágil tiene entre sus reglas "no producir documentos a menos que sean necesarios"; estos documentos deben ser cortos y centrarse en lo fundamental. Por estar especialmente orientadas para proyectos pequeños, las Metodologías Ágiles constituyen una solución a medida para ese entorno, aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto **(10)**. AUP proporciona un desarrollo del software más rápido y eficiente, con una generación de artefactos media que satisface los requerimientos para la construcción del sistema, a la vez que permite un ahorro de tiempo considerable.

2.4 Herramienta CASE

Una herramienta CASE representa un instrumento para modelar los procesos de negocio de las empresas y desarrollar sistemas de información. Además, permiten organizar y manejar la información de un proyecto informático y su uso está en dependencia de la metodología escogida para el análisis y diseño de dicho proyecto.

2.4.1 Visual Paradigm

Visual Paradigm es una herramienta CASE que utiliza UML como lenguaje de modelado, soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Visual Paradigm ofrece:

• Entorno de creación de diagramas para UML 2.0

• Diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.

• Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.

• Capacidades de ingeniería directa (versión profesional) e inversa.

• Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.

• Disponibilidad de múltiples versiones, para cada necesidad.

• Disponibilidad de integrarse en los principales IDEs, estas siglas indican en inglés (Integrated Development Environment) y su traducción al español significa Entorno de Desarrollo Integrado.

• Disponibilidad en múltiples plataformas. **(11)**

2.4.2 Visual Paradigm como herramienta CASE para el modelado de la solución propuesta.

Visual Paradigm se ha escogido como herramienta para el modelado de la solución propuesta, para realizar esta selección se tuvo en cuenta que es una herramienta multiplataforma, permite modelado orientado a objeto, y a la vez, orientado UML, el cual fue escogido para la realización del presente trabajo. Además, tiene algunas versiones gratuitas. Soporta aplicaciones web, exportación/importación XML y exporta en formato HTML. Cuenta además con gran cantidad de tutoriales, es fácil de instalar y de actualizar, tiene características gráficas muy cómodas y se encuentra disponible en varios idiomas.

2.5 Lenguajes de Programación.

Un lenguaje de programación es un conjunto de símbolos y reglas utilizados para controlar el comportamiento físico y lógico de una máquina. Es mediante este que los seres humanos pueden dar instrucciones a un equipo, particularmente una computadora.

¿Aplicación web o Escritorio?

Aplicación de escritorio: Son aplicaciones que se ejecutan sobre un sistema operativo como Windows o Linux (de interfaz visual) en un ordenador de escritorio.

Aplicación web: Son todas las aplicaciones software que se codifican en un lenguaje soportado por los navegadores web (HTML, Java Script, Java entre otros.) Los usuarios los pueden utilizar accediendo a un servidor web a través de Internet o de una intranet.

Ventajas de las aplicaciones web:

- Con las aplicaciones web no hay necesidad de instalar y distribuir el software a miles de usuarios.
- No ocupan espacio en el disco duro.
- Como la aplicación no se encuentra en el ordenador donde se utiliza, las tareas que realiza el software no consumen recursos.
- Permiten que el usuario acceda a los datos de modo interactivo garantizando que exista una comunicación activa entre el usuario y la información.
- Los virus no dañan los datos por medio del ordenador donde se utiliza porque éstos están guardados en el servidor de la aplicación.
- Se pueden realizar tareas sencillas sin necesidad de descargar ni instalar ningún programa.
- Se puede usar desde cualquier sistema operativo, solo es necesario tener un navegador por lo tanto es multiplataforma.
- Son portables porque no dependen del ordenador donde se utilice (un ordenador personal de sobremesa, un portátil, un móvil) porque se accede a través de una página web sólo es necesario disponer de acceso a Internet o intranet. Para su ejecución desde cualquier sitio en cualquier navegador web simplemente basta con teclear su dirección URL, estas siglas indican en inglés (Uniform Resource Locator) y su traducción en español localizador uniforme de recurso.
- Es fácil la administración de la información desde cualquier parte.
- Sus actualizaciones se hacen sin la necesidad de comprar el producto físicamente, sin descargas.

Las aplicaciones web son populares y ofrecen muchas ventajas sobre las aplicaciones de escritorio en cuanto a movilidad, facilidad de soporte y mantenimiento. Por todas las ventajas antes mencionadas para la solución propuesta se ha optado por una aplicación de tipo web.

2.5.1 Lenguaje PHP

PHP es un lenguaje interpretado de programación ampliamente usado que está orientado al desarrollo de aplicaciones web. Es un lenguaje sencillo, de sintaxis cómoda y similar a la de otros lenguajes como C o C++.

PHP es un lenguaje para programar scripts del lado del servidor, que se incrustan dentro del código HTML. Se escribe dentro del código HTML, lo que lo hace realmente fácil de utilizar y con algunas ventajas como su gratuidad, independencia de plataforma, rapidez y seguridad. Tiene además una gran librería de funciones y mucha documentación. Es independiente de plataforma, puesto que existe un módulo de PHP para casi cualquier servidor web. Esto hace que cualquier sistema pueda ser compatible con el lenguaje y significa una ventaja importante, ya que permite portar el sitio desarrollado en PHP de un sistema a otro sin prácticamente ningún trabajo. Este lenguaje de programación está preparado para realizar muchos tipos de aplicaciones web gracias a la extensa librería de funciones con la que está dotado **(12).**

Al ser PHP un lenguaje que se ejecuta en el servidor no es necesario que su navegador lo soporte, es independiente del navegador, pero sin embargo para que sus páginas PHP funcionen, el servidor donde están alojadas debe soportar PHP **(13)**.

Figura 3 Funcionamiento de PHP.

La figura 3 muestra que PHP se ejecuta en el servidor, lo que permite acceder a los recursos que tenga el servidor como por ejemplo podría ser una base de datos. El programa PHP es ejecutado en el servidor y el resultado enviado al navegador. El resultado es normalmente una página HTML.

2.5.2 PHP como lenguaje utilizado en la programación de la aplicación web desarrollada.

Después de un estudio de los lenguajes de programación y analizadas sus características y en aras de potenciar la utilización de recursos libres, se propone la utilización del lenguaje de programación PHP en su versión 5.0 para el desarrollo de la aplicación que se desea realizar.

La razón por la cual ha sido seleccionado es por la gran cantidad de ventajas que presenta sobre otros lenguajes de programación para el desarrollo de aplicaciones web.

Entre sus ventajas figuran:

- Es un lenguaje multiplataforma
- Tiene capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad.
- Posee una amplia documentación en su página oficial, entre la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite las técnicas de Programación Orientada a Objetos.
- Tiene una biblioteca nativa de funciones sumamente amplia e incluida.
- No requiere definición de tipos de variables y tiene manejo de excepciones (desde PHP5).

Además de este lenguaje se utilizarán otros lenguajes del lado cliente (entre los cuales no sólo se encuentra el HTML sino también JavaScript y CSS los cuales son simplemente incluidos en el código HTML) son aquellos que pueden ser directamente "digeridos" por el navegador y no necesitan un pre-tratamiento.

HTML, estas siglas en inglés indican (Hypertext Markup Language) y su traducción en español significa lenguaje de marcas hipertexto.

Se utiliza para crear documentos que muestren una estructura de hipertexto. Un documento de hipertexto es aquel que contiene información cruzada con otros documentos, lo cual nos permite pasar de un documento al referenciado desde la misma aplicación con la que lo estamos visualizando. HTML permite, además, crear documentos de tipo multimedia, es decir, que contengan información más allá de la simplemente textual como pueden ser imágenes, videos o sonido. **(14)**

JAVASCRIPT

Es un lenguaje de programación interpretado, ampliamente utilizado en el mundo del desarrollo web por ser muy versátil y potente, tanto para la realización de pequeñas tareas como para la gestión de complejas aplicaciones, es ejecutado por el navegador que utilizamos para ver las páginas; lo que hace que podemos desarrollar aplicaciones de diversos tipos, desde generadores de HTML, comprobadores de formularios, páginas web dinámicas, intercambiar información entre páginas web en distintas ventanas, manipulación de gráficos, texto, programas que gestionan las capas de una página **(14)**. Es un lenguaje débilmente tipiado, basado en objetos y guiado por eventos lo que permite el dinamismo de las páginas que incluyan este tipo de código, útil para el desarrollo de aplicaciones cliente-servidor dentro del ámbito de Internet. El código script tiene capacidades limitadas, por razones de seguridad, por lo que es necesario usarlo conjuntamente con HTML **(15)**.

CSS, **estas siglas en inglés indican (Cascading Style Sheets) y su traducción en español significa Hojas de Estilo en Cascada**.

Representan un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). Se utiliza para dar estilo a documentos HTML y XML, separando el contenido de la presentación. Los estilos definen la forma de mostrar los elementos HTML y XML. CSS permite a los desarrolladores web controlar el estilo y el formato de múltiples páginas web al mismo tiempo. Cualquier cambio en el estilo marcado para un elemento en la CSS afectará a todas las páginas vinculadas a esa CSS en las que aparezca ese elemento.

2.6 Servidor Web Apache 2.0

Apache, es un servidor web hecho por excelencia, su configurabilidad, robustez y estabilidad hacen que cada vez millones de servidores reiteren su confianza en este programa. Es un servidor que posee disímiles de características como:

- Corre en una multitud de Sistemas Operativos, lo que lo hace prácticamente universal.
- Es una tecnología gratuita de código abierto. El hecho de ser gratuita es importante pero no tanto como que se trate de código fuente abierto. Esto le da una transparencia a este software de manera que si queremos ver que es lo que estamos instalando como servidor, lo podemos saber, sin ningún secreto.
- Es un servidor altamente configurable de diseño modular. Es muy sencillo ampliar las capacidades del servidor Web Apache.
- Trabaja con gran cantidad de lenguajes de script como PHP teniendo todo el soporte que se necesita para tener páginas dinámicas.
- Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor. Es posible configurar Apache para que ejecute un determinado script cuando ocurra un error en concreto **(16)**.

2.7 Symfony como framework para el desarrollo de la aplicación propuesta.

Un framework simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, proporciona estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener. Por último, un framework facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas **(17)**. En la mayoría de los casos un framework implementa uno o más patrones de diseño de software que aseguran la escalabilidad del producto aunque muchas veces su uso añade código innecesario.

Symfony es un completo framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es que no se debe reinventar la rueda cada vez que se crea una nueva aplicación web **(17).**

Symfony está desarrollado completamente con PHP 5. Symfony es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, entre otros.) como en plataformas Windows **(17).**

2.8 Sistema Gestor de Base Datos (SGBD)

Un Sistema Gestión de Bases de Datos consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a esos datos. Es un software que sirve de interfaz entre la base de datos (BD), el usuario y las aplicaciones que la utilizan. Es también un sistema de gestión que les permite a los usuarios manejar de forma ordenada el conjunto de datos que conforman la BD.

2.8.1 Sistema gestor de base datos PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, con su código fuente disponible libremente sin costo, utiliza un modelo cliente/servidor y usa multiprocesos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando. Sus características técnicas la hacen una de las bases de datos más potentes y robustas del mercado. Durante todo su desarrollo, estabilidad, potencia, robustez, facilidad de administración e implementación de estándares han sido las características que más se han tenido en cuenta. PostgreSQL funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema. **(18)**

Posee otras características dentro de las cuales se encuentran:

- PostgreSQL acerca los datos a un modelo objeto-relacional, y es capaz de manejar complejas rutinas y reglas.
- Soporta casi toda la sintaxis SQL (incluyendo sub-consultas, transacciones, y tipos y funciones definidas por el usuario).
- Soporta una gran parte de SQL estándar y ofrece muchas características modernas:
  - Consultas complejas.
  - Claves foráneas.
  - Triggers o disparador.
  - Vistas.
  - La integridad transaccional.
  - Control de concurrencia multi-versión **(19)**.
- Puede ser instalado en una cantidad de servidores ilimitada porque con PostgreSQL no se violan acuerdos de licencia, puesto que no hay costo asociado a la licencia del software. Está disponible en casi todos los principales sistemas operativos: Linux, Unix, BSDs, Mac OS, Beos, Windows, entre otros. Si se necesita extender o personalizar PostgreSQL se puede hacer sin costos adicionales.

2.8.2 Gestor de BD seleccionado

Luego de analizada las características de algunos de los Sistemas Gestores de Base de Datos más conocidos y utilizados en la actualidad se llegó a la conclusión de que el más idóneo a utilizar en el presente trabajo de diploma es el PostgreSQL. Se escogió el gestor de BD PostgreSQL para manejar los datos del sistema porque es multiplataforma y no tiene costo asociado a la licencia de software, es una propuesta gratis para poder gestionar fácilmente una base de datos de gran volumen. Este permite que mientras un usuario este haciendo alguna modificación en las tablas otro pueda realizar cambios sin que se bloquee el gestor. Además, brinda gran consistencia y seguridad a los datos. Este gestor de base de datos implementa el uso de subconsultas y transacciones, lo que proporciona una mayor eficacia a su funcionamiento y ofrece soluciones en campos en los que el MySQL no podría. Tiene la capacidad de comprobar la integridad referencial, así como la de almacenar procedimientos en la propia base de datos, equiparándolo con los gestores de bases de datos de alto nivel. PostgreSQL está disponible sin costo alguno.

2.9 Conclusiones del capítulo

En este capítulo después de analizadas las diferentes herramientas, metodologías y lenguajes existentes para la realización de la aplicación, se obtuvieron los datos necesarios para la selección final en aras de dar solución al problema en cuestión. Se definió utilizar como metodología de desarrollo AUP, así como Visual Paradigm como herramienta case por UML como herramienta de modelado, se escogió a PHP como lenguaje de programación web a utilizar, el framework de desarrollo symfony y como gestor de base de datos PostgreSQL. Al realizar esta selección se ha tenido en cuenta dos aspectos importantes, uno de estos es el soporte sobre diferentes sistemas operativos y por otro lado el cumplimiento con los principios de software libre.

Capítulo

Presentación de la solución propuesta

CAPÍTULO III: PRESENTACIÓN DE LA SOLUCIÓN PROPUESTA.

3.1 Introducción.

En el presente capítulo se especifican cuales son las funcionalidades del software, definiéndose los RF y los RNF; partiendo de estos requerimientos se obtienen y describen los casos de uso, los cuales guían todo el proceso llevado a cabo para la solución del problema. Se presenta un modelo de dominio en aras de lograr la comprensión de los conceptos asociados al entorno donde trabajará el sistema y se describen los actores con los que se cuentan.

3.2 Entorno donde trabajará el sistema.

Como no existe un negocio bien definido, y no se puede precisar la estructura de los procesos del negocio, se empleó un modelo de dominio. El modelo de dominio es un medio para comprender el sector del negocio al cual el sistema va a servir, con el objetivo de mostrar al usuario los principales conceptos que se tratan en el entorno, logrando un mejor entendimiento del sistema.

3.2.1 Conceptos principales del entorno.

Proyecto: Un proyecto es una planificación que consiste en un conjunto de actividades que se encuentran interrelacionadas y coordinadas; la razón de un proyecto es alcanzar objetivos específicos dentro de los límites que imponen un presupuesto, calidades establecidas previamente y un lapso de tiempo previamente definido.

Arquitecto: Es el responsable de la arquitectura del software, así como de la selección, gestión y obtención de las herramientas que se utilizarán en el proyecto. Define los patrones de diseño a utilizarse en el proyecto, la metodología de organización y desarrollo del proyecto, así como la estrategia de comunicación para lograr la integración de las diferentes partes y con otros sistemas. Uso de estándares.

Documento Arquitectura: Es un documento breve que tiene información esencial, tal como:

1. Elementos y los principales componentes del sistema.
2. Capas del sistema.
3. Mecanismos arquitecturales necesarios para el sistema.
4. Tecnologías elegidas para hacer frente a cada capa y el mecanismo y la motivación detrás de estas opciones.
5. Componentes comprados o de código abierto que se utilizarán y la forma en que se tomó la decisión.
6. Descripción de los principales patrones de diseño utilizados y por qué la elección.

Equipo Evaluación: Grupo de personas responsables de la realización de una evaluación. La composición del equipo debe ser equilibrada, y estar formado por expertos(as) independientes y representantes de las contrapartes y / o de la población beneficiaria, expertos (as).

3.2.2 Diagrama de clases del Modelo de Dominio.

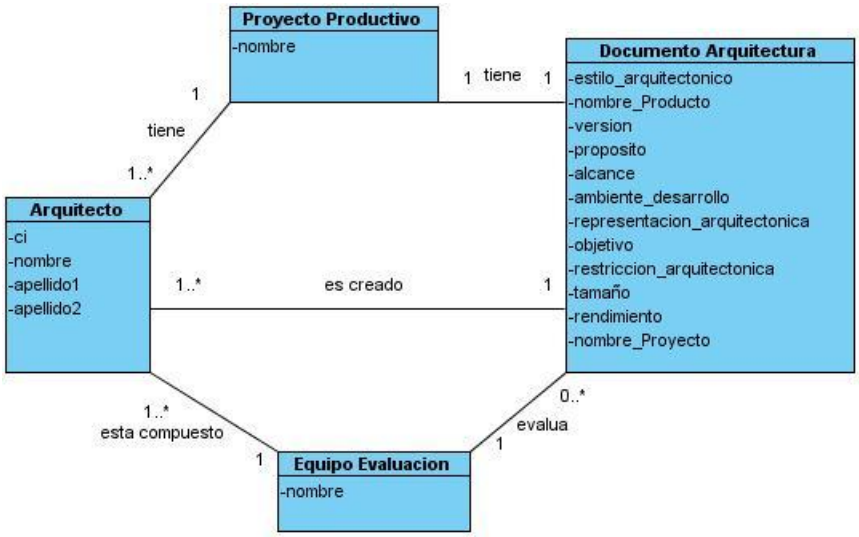

Figura 4 Modelo de dominio.

Descripción del modelo de dominio:

Cada facultad de la UCI cuenta con varios proyectos productivos, cada uno conformado por uno o varios arquitectos, encargado(s) de realizar el documento de arquitectura, el cual tiene información esencial de la arquitectura del software que se encuentra en construcción. Este documento es certificado por un equipo de evaluación conformado por un grupo de arquitectos expertos en arquitectura de software.

3.3 Requerimientos del sistema.

3.3.1 Requerimientos Funcionales

El sistema debe cumplir con una serie de requisitos funcionales (capacidades o condiciones) para satisfacer las funcionalidades requeridas o las restricciones que se imponen al sistema.

RF1 Autenticar usuario

1.1 Permitir que el arquitecto introduzca su usuario y contraseña.
1.2 Validar los datos introducidos.

RF2 Gestionar arquitectura

El sistema debe permitir que el arquitecto gestione la arquitectura, correspondiente al proyecto al que él pertenece.

2.1 Listar estilos arquitectónicos.

2.2 Listar patrones arquitectónicos.

2.3 Listar patrones de diseño.

2.4 Registrar estilos arquitectónicos**.**

2.5 Registrar patrones arquitectónicos.

2.6 Registrar patrones de diseño.

2.7 Actualizar estilos arquitectónicos.

2.8 Actualizar patrones arquitectónicos.

2.9 Actualizar patrones de diseño.

2.10 Eliminar estilos arquitectónicos.

2.11 Eliminar los patrones arquitectónicos.

2.12 Eliminar los patrones de diseño.

RF3 Gestionar escenarios.

El sistema debe permitir que el arquitecto gestione los escenarios, correspondientes al proyecto al que él pertenece.

3.1 Listar escenarios.

3.2 Registrar escenarios.

3.3 Actualizar escenarios.

3.4 Eliminar escenarios.

RF4 Gestionar arquitecto.

El sistema debe permitir que el administrador gestione arquitecto que tendrá acceso al sistema.

4.1 Listar arquitectos.

4.2 Registrar arquitectos.

4.3 Actualizar arquitectos.

4.4 Eliminar arquitectos.

RF5 Gestionar conocimiento.

El sistema debe permitir que el administrador gestione el conocimiento con que contará el sistema.

5.1 Listar conocimientos.

5.2 Registrar conocimientos.

5.3 Actualizar conocimientos.

5.4 Eliminar conocimientos.

RF6 Evaluar.

El sistema debe permitir que el arquitecto evalúe la arquitectura, correspondiente al proyecto al que él pertenece.

6.1 Listar escenarios.

6.2 Evaluar.

RF7 Gestionar Patrón

El sistema debe permitir que el administrador gestione los patrones arquitectónicos con que contará el sistema.

7.1 Listar patrones

7.2 Registrar patrones

7.3 Actualizar patrones

7.4 Eliminar patrones

RF8 Gestionar Relación

El sistema debe permitir que el administrador gestione las relaciones que existen entre los patrones arquitectónicos con que contara el sistema.

8.1 Listar relaciones

8.2 Registrar relaciones

8.3 Actualizar relaciones

8.4 Eliminar relaciones.

3.3.2 Requerimientos No Funcionales

El sistema debe tener una serie requisitos no funcionales (propiedades o cualidades) que representan las características del producto.

Requisitos de usabilidad

- Será usable por cualquier tipo de arquitecto con experiencia básica, media o avanzada en arquitectura de software.
- Se mostrará la información de forma lógica y correctamente estructurada.
- La aplicación contará con íconos representativos.
- Usa botones y formularios de un tamaño adecuado para la vista del usuario.
- Proporciona un diseño ordenado.
- Se brindan mensajes al usuario cada vez que realiza una acción.

RNF2 Fiabilidad

El factor fiabilidad del sistema se maneja mediante una atención rápida y esmerada de los errores producidos en tiempo de ejecución.

Requisitos de fiabilidad

- Se controlará ofreciendo oportunamente los mensajes e indicaciones pertinentes al usuario.
- Se informa la presencia de algún error de forma inmediata luego de haberse producido.

RNF3 Eficiencia

- Se debe garantizar que el tiempo de respuesta a las solicitudes se produzca en la mayor brevedad posible. Mostrar en pantalla la mayor cantidad de información posible.
- La eficiencia del producto estará determinada en gran medida por el aprovechamiento de los recursos que se disponen en el modelo Cliente/Servidor, y la velocidad de las consultas en la Base de Datos.
- Para el funcionamiento óptimo de la aplicación se siguieron las diferentes técnicas de elaboración en la web, que facilitan el rápido acceso a sus páginas.

RNF4 Soporte

El sistema tiene garantía de prueba e instalación del mismo. Se le realizaron diferentes pruebas para comprobar su correcto funcionamiento. El mismo debe correr sobre los sistemas operativos Windows 2000/XP/Vista principalmente y sobre el servidor web Apache 2.0

RNF5 Restricciones de diseño

- La base de dato será desarrollada en PostgreSQL.
- Para garantizar el desarrollo de la aplicación se utilizó como metodología el Proceso Unificado Ágil (AUP).
- Se utilizó para realizar los modelos del sistema, UML y como herramienta de apoyo a este Lenguaje de Modelación Visual Paradigm.

RNF6 Restricciones de implementación

Para la implementación del sistema se utilizará el lenguaje de programación PH5 y como IDE de desarrollo phpDesigner 7.

RNF7 Seguridad

- La seguridad de la aplicación estará respaldada por el lenguaje php5.
- Se aplicarán las reglas de la "programación segura", mediante el tratamiento de excepciones.
- El sistema permitirá además la verificación sobre acciones irreversibles; es decir, se le solicitará al arquitecto la confirmación al realizar operaciones como la eliminación.
- El sistema garantizará la confidencialidad, integridad y disponibilidad de la información almacenada en la base de datos.
- Cada usuario del sistema contará con una contraseña para poder acceder al mismo, la cual será guardada en la base datos.
- Se le asignará a cada usuario un nivel de acceso determinado que le permitirá interactuar con el sistema según los permisos que tenga.
- Para evitar las inyecciones de SQL se utilizará el lenguaje de programación Java Script, el cual permitirá la validación de los campos que tendrán los formularios, a la hora de almacenar la información en la base de datos.

RNF8 Interfaces de usuario

- El sistema tendrá un menú, para que el arquitecto pueda disponer de él en la medida de sus necesidades, visible todo el tiempo para propiciar el fácil acceso a las funcionalidades del software.
- La interfaz gráfica de la aplicación tendrá un ambiente agradable, sencilla y de navegación fácil, combinada con colores y una estructura amigable que permita que el arquitecto se adapte con facilidad.
- La interfaz principal contará con el logotipo del sistema.

RNF9 Hardware

Máquina servidora

- Memoria RAM: 1GB, Disco Duro: 160 GB, Tarjeta de Red: Ethernet 10/100 MB.

Máquina Cliente

- Memoria RAM: 256 MB, Disco Duro: 2.5 GB, Tarjeta de Red: Ethernet 10/100 MB.

RNF10 Software

Máquina servidora

- Sistema Operativo Windows 2000/XP/Vista, Servidor Web Apache 2.0, Gestor de Base de Datos PostgreSQL.

Máquina Cliente

- Sistema Operativo Windows 2000/XP/Vista, navegador Mozilla Firefox o Internet Explorer.

3.4 Descripción del Sistema Propuesto.

Tabla 1 Descripción de los actores sistema.

Actor	Descripción
Arquitecto	Este actor después de haberse autenticado puede gestionar información de la arquitectura del software que está realizando el proyecto al cual pertenece, así como los requerimientos no funcionales que debe cumplir el sistema (en forma de escenario), para finalmente realizar la evaluación de dicha arquitectura, después de haber clasificado y priorizado dichos escenarios.
Administrador	Es un arquitecto que puede gestionar usuarios y roles, además de los datos con que contará el sistema para poder realizar la evaluación de la arquitectura del software.

Casos de Uso del Sistema

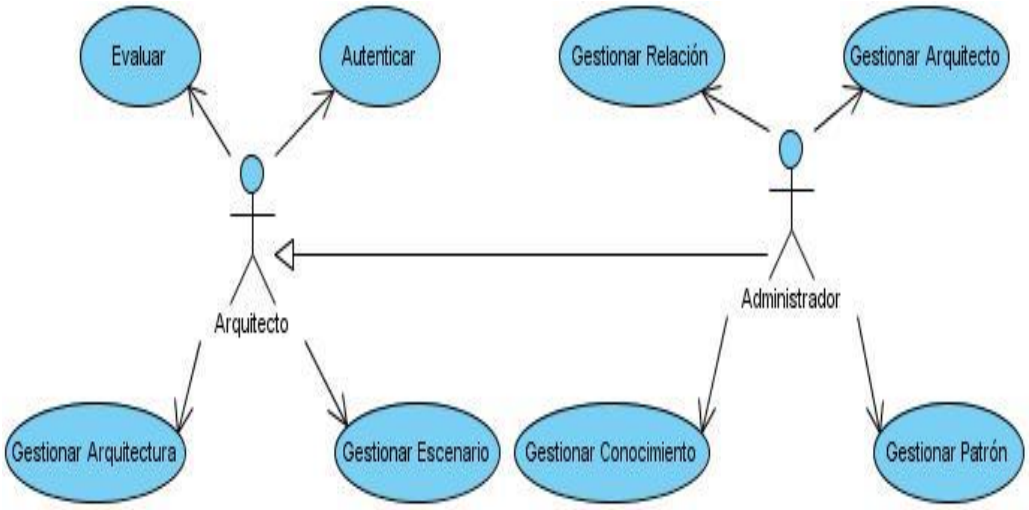

Figura 5 Diagrama de caso de uso del sistema.

Tabla 2 Descripción del Casos de Uso Evaluar.

Caso de Uso:	Evaluar
Actores:	Arquitecto
Resumen:	El caso de uso comienza cuando el arquitecto selecciona la opción de evaluar, donde podrá conocer si la arquitectura seleccionada satisface los escenarios seleccionados por el arquitecto, correspondientes al proyecto al que pertenece, y termina con la respuesta del sistema.
Precondiciones:	El arquitecto debe de estar registrado en la base de datos, la contraseña debe ser la correspondiente al mismo y debe de existir escenarios, correspondientes al proyecto al que pertenece el arquitecto, todos priorizados.

Referencias	RF6
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Negocio**
1 Selecciona la opción evaluar.	
	2 Busca todos los datos de los escenarios existentes, correspondientes al proyecto al cual pertenece el arquitecto.
	3 Muestra un listado con todos los datos de los escenarios que tiene el sistema correspondiente al proyecto al que pertenece el arquitecto.
4 Selecciona escenarios a evaluar.	
5 Selecciona la opción “evaluar”.	
	6 Verifica que los escenarios estén priorizados.
	7 Crea una lista con todos los escenarios ordenados según su prioridad, de forma descendente.
	8 Crea una lista de debilidades.
	9 Selecciona del listado de escenario ordenado el primero, el de mayor prioridad.

	10 Busca dentro de los patrones, correspondientes al proyecto al que pertenece el arquitecto, si hay alguno que satisfaga al escenario seleccionado; para esto se apoya de los conocimientos existentes. Ir a la acción 9.
Flujo Alterno	
Acción del Actor	**Respuesta del Sistema**
6a Escenarios no Priorizados	6a 1 Muestra un mensaje que indica que existen escenarios no priorizados y termina el caso de uso.
9a Listado de Escenarios Ordenados Vacío.	9a 1 Muestra un listado de debilidades que presenta la arquitectura seleccionada y termina el caso de uso.
10a No existe patrones que satisfagan el escenario seleccionado.	10a 1 Adiciona a la lista de debilidades que la arquitectura seleccionada no satisface el escenario seleccionado.
Poscondiciones	Se obtiene un listado de las debilidades de la arquitectura seleccionada.

Tabla 3 Descripción del Casos de Uso Gestionar Escenario.

Caso de Uso:	Gestionar Escenario
Actores:	Arquitecto
Resumen:	El caso de uso comienza cuando el arquitecto selecciona la opción de gestionar escenario, donde podrá registrar, actualizar o eliminar escenario correspondiente al proyecto al que pertenece el arquitecto, y termina cuando acepta la opción seleccionada.
Precondiciones:	El arquitecto debe de estar registrado en la base de datos y la contraseña debe ser la correspondiente al mismo.
Referencias	RF3
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Negocio**
1 Selecciona la opción gestionar escenario.	
	2 Busca todos los datos de los escenarios existentes correspondiente al proyecto al que pertenece el arquitecto.
	3 Muestra un formulario con los campos necesarios para registrar un escenario; así como un listado con todos los datos de los escenarios que tienen el sistema correspondiente al proyecto al que pertenece el arquitecto, dando la

	posibilidad de actualizar o eliminar el o los escenarios que el arquitecto desee.
4 Selecciona la opción que desea realizar.	
	5 Ejecuta la opción seleccionada. • Si selecciona registrar, ir a la sección “Registrar”. • Si selecciona actualizar, ir a la sección “Actualizar”. • Si selecciona eliminar, ir a la sección “Eliminar”. Ir a la acción 4.
Flujo Normal de Eventos	
Sección “Registrar”	
Acción del Actor	**Respuesta del Sistema**
1. Introduce los datos.	
2 Selecciona la opción “Registrar”.	
	3 Verifica que los datos estén correctos.

	4 Verifica que los datos no existan.
	5 Registra los datos.
	6 Muestra un mensaje que indica que la operación se realizó con éxito.
Flujo Alterno	
Acción del Actor	**Respuesta del Sistema**
3a Datos Introducidos Incorrectos	3a 1 Muestra un mensaje que indica que los datos están incorrectos.
4a Datos Introducidos Existentes	4a 1 Muestra un mensaje que indica que los datos ya existen.
Sección "Actualizar"	
Acción del Actor	**Respuesta del Sistema**
1. Introduce los nuevos datos.	
2 Selecciona la opción "Actualizar".	
	3 Verifica los datos.
	4 Verifica que los datos no existan.

	5 Actualiza los datos.
	6 Muestra un mensaje que indica que la operación se realizó con éxito.
Flujo Alterno	
3a Datos Introducidos Incorrectos	3a 1 Muestra un mensaje que indica que los datos están incorrectos.
4a Datos Introducidos Existentes	4a 1 Muestra un mensaje que indica que los datos ya existen.
Sección "Eliminar"	
Acción del Actor	**Respuesta del Sistema**
1. Selecciona los datos que desea eliminar.	
2 Selecciona la opción "Eliminar".	
	3 Muestra un mensaje de confirmación.
	4 Elimina los datos.
	5 Muestra un mensaje que indica que la operación se realizó con éxito.
Flujo Alterno	

Acción del Actor	Respuesta del Sistema
3a Acción Cancelada	3a 1 Cancela la acción.
Poscondiciones	Se obtiene la realización de cada funcionalidad seleccionada.

3.5 Conclusiones del capítulo

En este capítulo quedaron expuestos los principales conceptos del entorno en el que se encuentra enmarcado el sistema. Realizándose un modelo de domino para una mejor comprensión del entorno donde coexiste el problema, definiéndose a partir de este modelo, conceptos, entidades y sus relaciones. Se identificaron los requerimientos que el sistema deberá cumplir, clasificándolo en RF y RNF con el objetivo de satisfacer las restricciones y necesidades del mismo.

Capítulo

Construcción de la solución propuesta

CAPÍTULO IV: IMPLEMENTACIÓN DE LA SOLUCIÓN PROPUESTA.

4.1 Introducción

Este capítulo presenta el diseño, la implementación y la prueba del sistema realizado, así como el diagrama de despliegue. Describiéndose la construcción de la aplicación, las principales características del patrón arquitectónico a utilizar y los patrones de diseño empleados. Los requisitos se traducen a una especificación que describen cómo implementar el sistema. Partiendo de los requisitos, se realiza el diagrama de clases del diseño y de los resultados obtenidos se comienza con la implementación para dar entrada a la etapa de pruebas; en esta última se realiza el diseño de los casos de pruebas para los casos de uso que se determinaron de gran importancia para el sistema en general. Además, se muestra el diseño de la base de datos.

4.2 Descripción de la arquitectura

Dado que se opta por utilizar el framework Symfony por las características y ventajas antes expuestas, se utiliza la arquitectura Modelo-Vista-Controlador (MVC) que propone dicho framework para el desarrollo la presente aplicación web.

El Patrón clásico del diseño web conocido como arquitectura MVC, que está formado por tres niveles:

• El modelo representa la información con la que trabaja la aplicación, es decir, su lógica de negocio.

• La vista transforma el modelo en una página web que permite al usuario interactuar con ella.

• El controlador se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista. **(10)**

La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. Por ejemplo, si una misma aplicación debe ejecutarse tanto en un navegador estándar como en un navegador de un dispositivo móvil, solamente es

necesario crear una vista nueva para cada dispositivo; manteniendo el controlador y el modelo original. El controlador se encarga de aislar al modelo y la vista de los detalles del protocolo utilizado para las peticiones. El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación. **(10)**

Este modelo de arquitectura presenta varias ventajas:

• Hay una clara separación entre los componentes de un programa; lo cual nos permite implementarlos por separado.

• Hay un API muy bien definido; cualquiera que use el API, podrá reemplazar el Modelo, la Vista o el Controlador, sin aparente dificultad.

• La conexión entre el Modelo y sus Vistas es dinámica; se produce en tiempo de ejecución, no en tiempo de compilación.

• Al incorporar el modelo de arquitectura MVC a un diseño, las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Posteriormente, si uno de los componentes, se observa que funciona mal puede reemplazarse sin que las otras piezas se vean afectadas. **(10)**

Patrones de diseño empleados.

Symfony sigue la mayoría de mejores prácticas y patrones de diseño para la web **(17)**. Para dar solución a la implementación de los casos de uso planteados anteriormente se aplican los patrones de diseño y los patrones GRASP que utiliza el framework.

Experto: Expresa que los objetos hacen cosas de acuerdo a la información con que cuentan. Esto trae como beneficio que se conserve el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Además, favorece el bajo acoplamiento. El comportamiento se distribuye entre las clases que cuentan con la información requerida alentando con ello definiciones de clases sencillas que son fáciles de comprender y mantener.

Creador: Permite asignar quien debería ser el responsable de la creación de una nueva instancia de alguna clase. Guía la asignación de responsabilidades relacionadas con la creación de objetos.

El propósito general de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento.

Alta Cohesión: El objetivo de este patrón es asignar responsabilidades de tal forma que la cohesión siga siendo alta. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.

Controlador: El propósito de este patrón es asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. El controlador no realiza estas actividades, las delegas en otras clases con las que mantiene un modelo de alta cohesión. Se utiliza un controlador frontal donde todas las peticiones web son manejadas por dicho controlador, que es el punto único de entrada de toda la aplicación.

Bajo Acoplamiento: El bajo acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios. Este patrón especifica cómo dar soporte a una dependencia escasa y a un aumento de la reutilización.

Patrones GOF

Decorador (Envoltorio): Añade funcionalidad a una clase, dinámicamente. El archivo layout.php, almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página.

Singleton (Instancia única): Garantizar que una clase sólo tiene una única instancia, proporcionando un punto de acceso global a la misma.

Abstract Factory (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.

Fachada: Symfony, mediante el sistema de configuración de archivos .yml, implementa este patrón permitiendo acceder a diferentes configuraciones del framework desde un único lugar. Además de que el acceso a la aplicación es a través del controlador frontal que implementa este patrón.

Adapter (Adaptador): Symfony da la posibilidad de cambiar a otro sistema de base de datos completamente diferente a mitad de desarrollo, solo basta con configurar un archivo .yml. La capa de abstracción utilizada encapsula toda la lógica de los datos. El resto de la aplicación no tiene que preocuparse por las consultas SQL y el código SQL que se encarga del acceso a la base de datos es fácil de encontrar.

4.3 Diseño del sistema.

El diagrama de clases de diseño de un sistema refleja los detalles que tienen que ver más concretamente con la implementación, mostrando la estructura interna del sistema, en cuanto a la información concerniente a cada una de las clases que forman el mismo, atributos y sus tipos de datos, métodos y sus tipos de datos de retorno y además brinda una representación gráfica de las relaciones entre todas las clases del sistema. Seguidamente se muestran los diagramas de clases del diseño de los casos de usos críticos del sistema. En ellos se pueden observar, el uso del estilo arquitectónico MVC.

Diagrama de clase del diseño del caso de uso Evaluar.

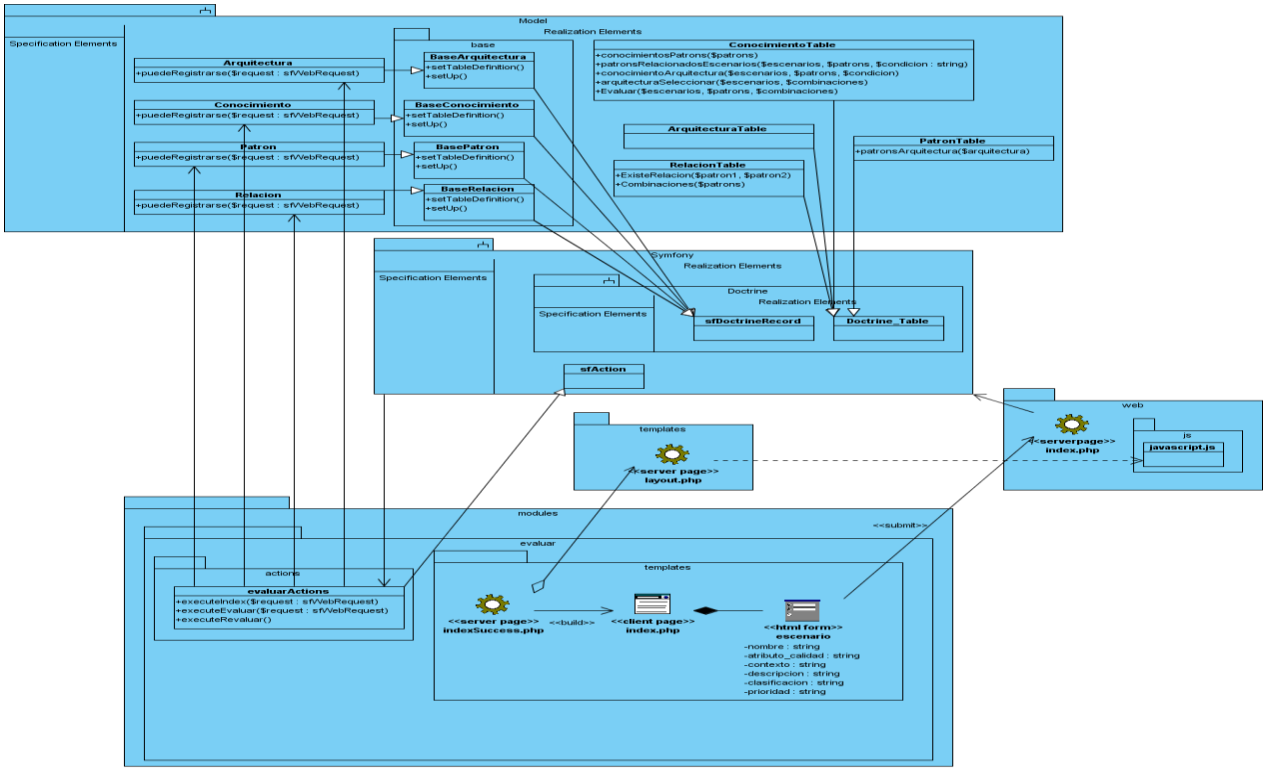

Figura 6 Diagrama de clase del diseño del CU Evaluar.

Diagrama de clase del diseño del caso de uso Gestionar Escenario.

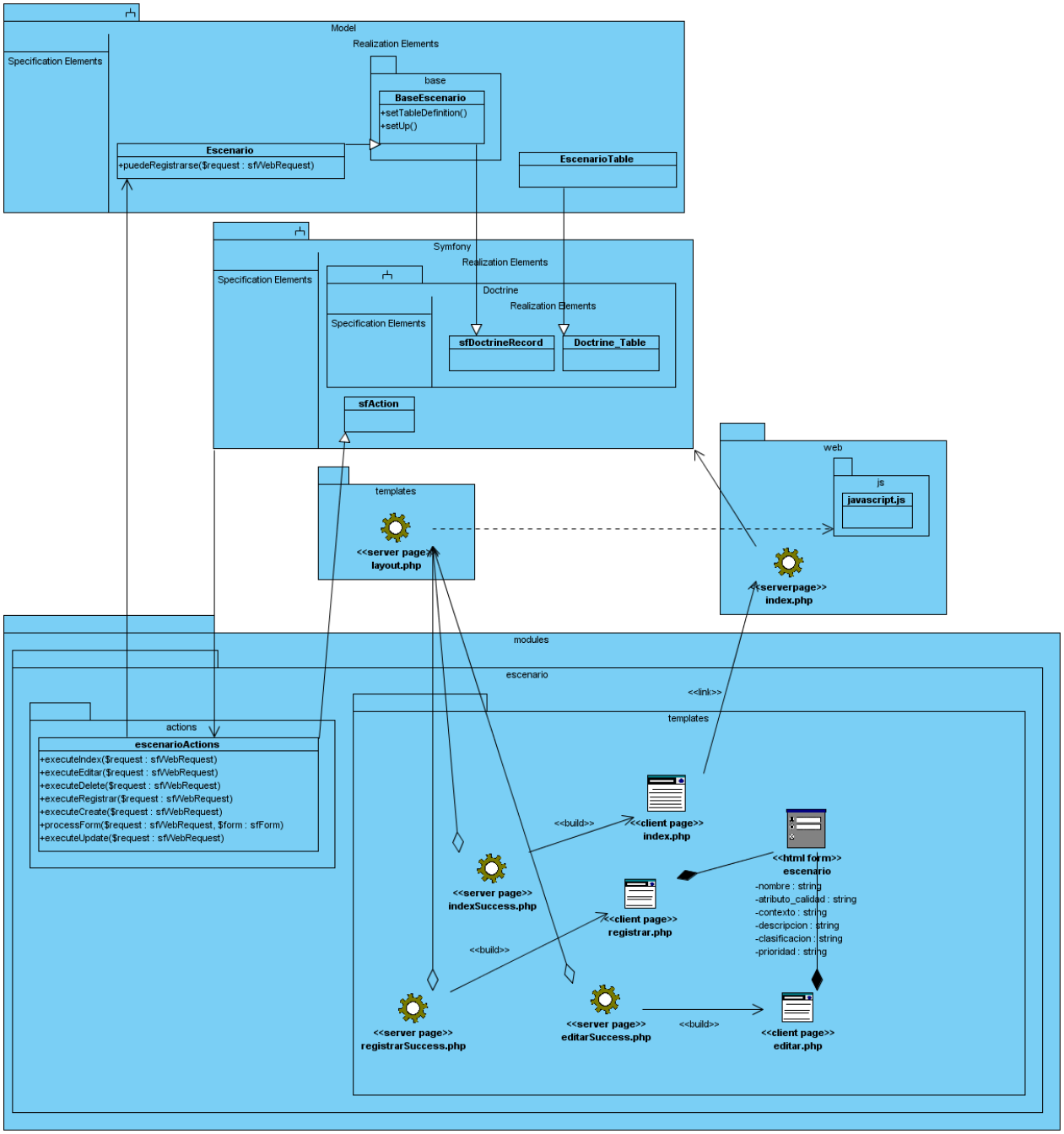

Figura 7 Diagrama de clase del diseño del CU Gestionar Escenario.

4.4 Diseño de la Base de Datos

Diagrama de clases persistentes de la base de datos del sistema.

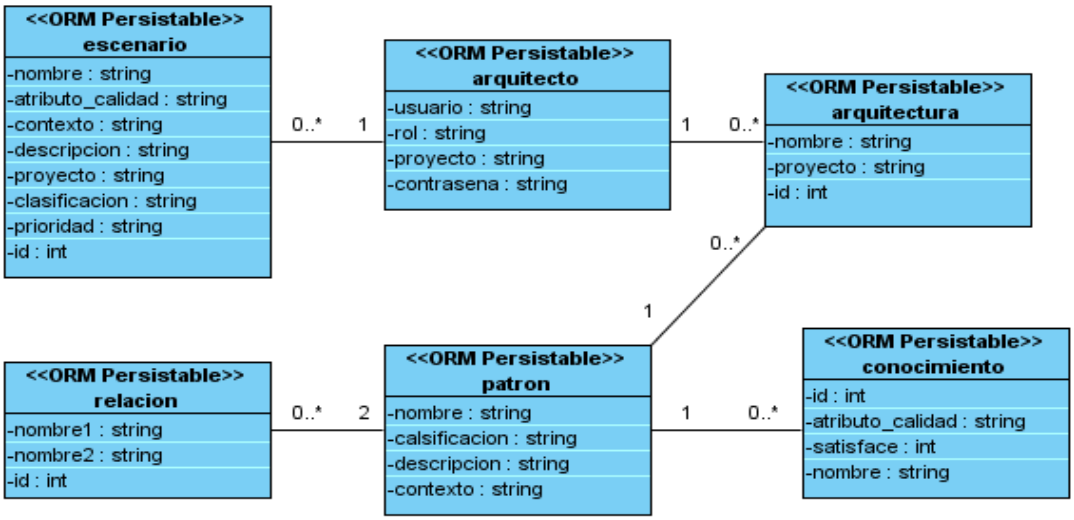

Figura 8 Diagrama clases persistentes de la base de datos del sistema.

Diagrama entidad-relación de la base de datos del sistema.

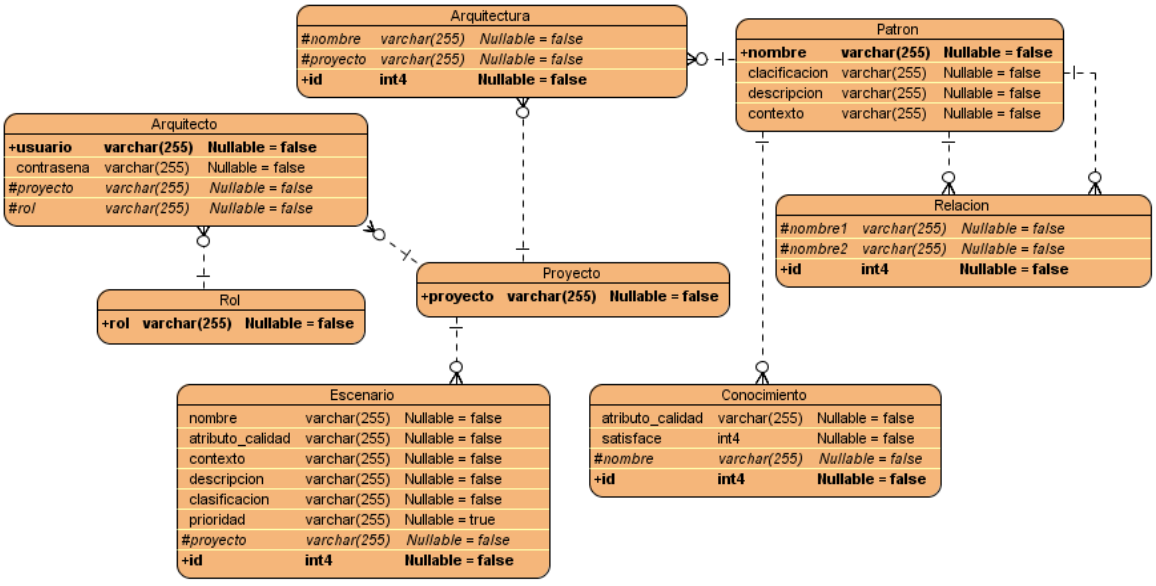

Figura 9 Diagrama entidad-relación de la base de datos del sistema.

4.5 Modelo de implementación

La implementación empieza con el resultado del diseño y se implementa en términos de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares. El flujo de trabajo de implementación detalla cómo los elementos del modelo del diseño se implementan en términos de componentes y representa cómo se organizan en el modelo de despliegue. Permite obtener los diagramas de despliegue y componentes los cuales son artefactos que conforman lo que se conoce como un modelo de implementación. A continuación se realizará el diagrama de despliegue y diagrama de componentes correspondiente a la lógica de presentación y negocio del software informático para uso de los arquitectos de información en la UCI.

4.5.1 Diagramas de componentes

Los diagramas de componentes son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre los elementos de implementación. El Diagrama de Componente muestra la relación entre los componentes del software, sus dependencias, comunicaciones, localización y otras condiciones.

Diagrama de Componente del CU Evaluar.

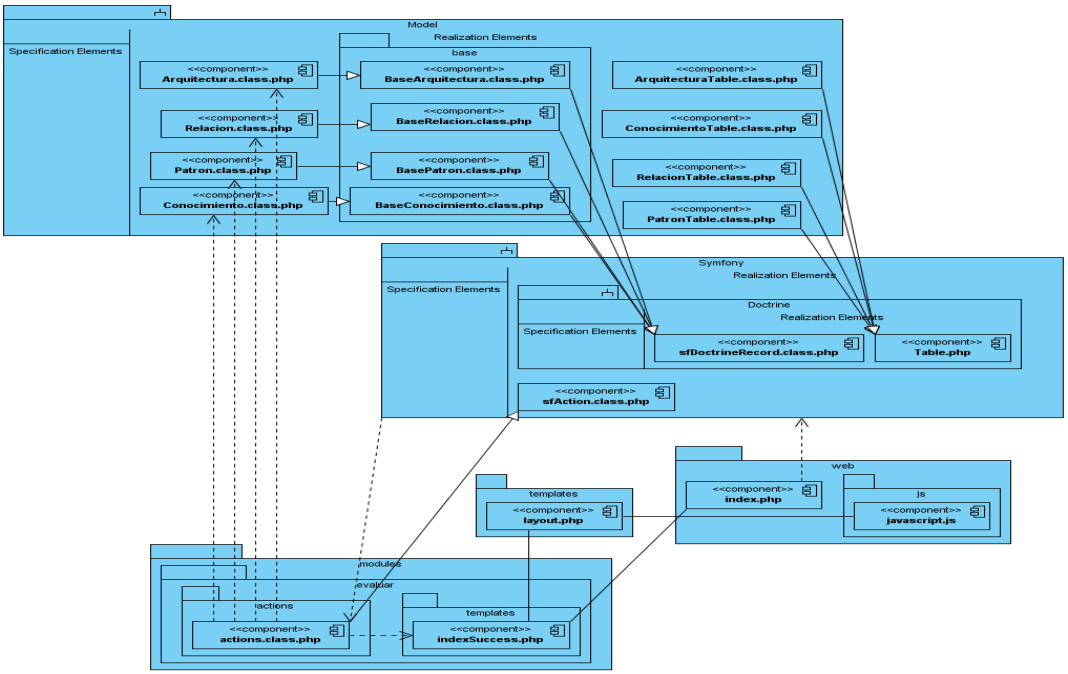

Figura 10 Diagrama de componente del CU Evaluar.

Diagrama de Componente del CU Gestionar Escenario.

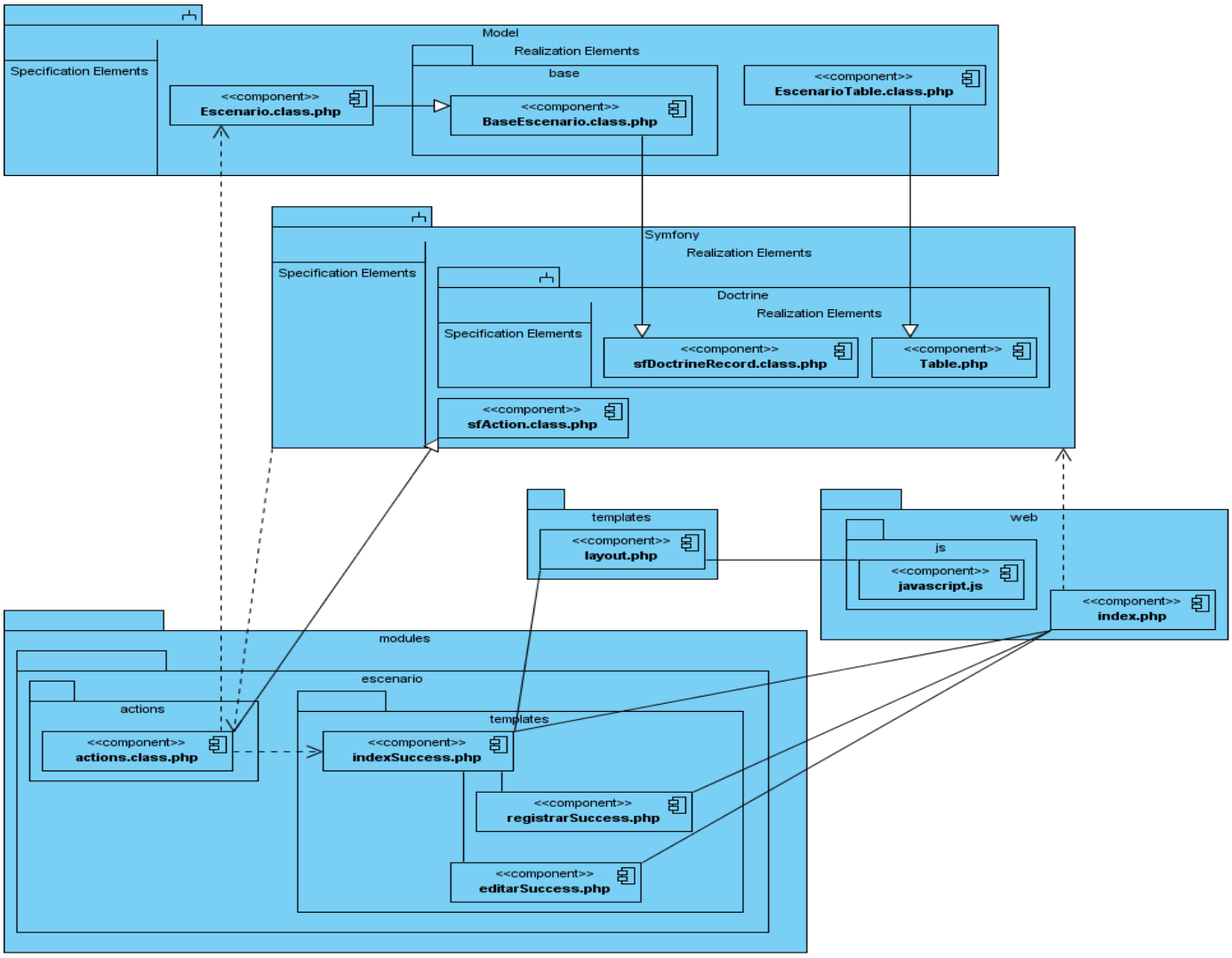

Figura 11 Diagrama de componente del CU Gestionar Escenario.

4.6 Modelo de Despliegue

El modelo de despliegue es un modelo de objetos que describe la distribución física de un sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. En esta vista se realiza una representación gráfica de los nodos físicos en los que estará desplegado el sistema propuesto y la comunicación entre ellos.

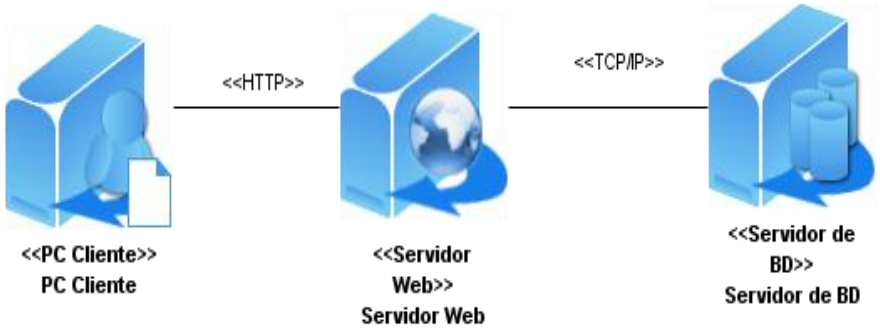

Figura 12 Diagrama de despliegue.

4.7 Prueba del sistema propuesto

El desarrollo del software ha de ir acompañado de alguna actividad que garantice la calidad del software, la prueba es un elemento crítico para ello.

4.7.1 Prueba de Caja negra

Objetivo: El objetivo de realizarle este tipo de prueba al sistema, es para detectar el incorrecto o incompleto funcionamiento de este, así como los errores de interfaces, rendimiento y errores de inicialización y terminación.

Alcance: El proceso de pruebas de caja negra se va a centrar principalmente en los requisitos funcionales del software para verificar el comportamiento de la interfaz gráfica, su interacción con el usuario y la calidad funcional.

Casos de Prueba: Para verificar que se cumpliera el requerimiento funcional más importante de los establecidos anteriormente, se le realizó la prueba al caso de uso evaluar escenario, que es considerado el caso de uso más crítico. Esta se realizó con el objetivo de evaluar la interacción del usuario con el sistema en lo que respecta a esta funcionalidad.

Tabla 4 Diseño de prueba del caso de uso Evaluar Escenario.

Entrada	Resultados	Condiciones
El arquitecto desea evaluar sin haber creado escenarios.	El sistema notifica al arquitecto que debe crear escenarios para poder realizar la evaluación.	No se crean escenarios para realizar la evaluación.
El arquitecto desea evaluar sin haber seleccionado los escenarios.	El sistema notifica al arquitecto que debe seleccionar escenarios.	No se seleccionan escenarios para realizar la evaluación.
El arquitecto desea evaluar escenarios sin haber entrado al sistema la arquitectura correspondiente a su proyecto.	El sistema notifica al arquitecto que debe entrar una arquitectura para poder evaluar los escenarios directos.	No se tiene la arquitectura correspondiente al proyecto al que pertenece el arquitecto.
El arquitecto para realizar una evaluación selecciona los escenarios por los que será evaluada la arquitectura entrada.	El sistema notifica las debilidades y las fortalezas de la arquitectura entrada; así como la arquitectura más conveniente a usar para satisfacer los escenarios seleccionados.	Se seleccionan los escenarios y se entra al sistema la arquitectura a evaluar.

4.8 Conclusiones del capítulo.

En este capítulo se obtuvo como resultado los diagramas de clases de diseño y de igual forma el diseño de la base de datos construido por medio del diagrama de clases persistentes, el modelo de datos y el diagrama entidad-relación. Se muestra el modelo de despliegue y el diagrama de componentes del sistema propuesto, permitiendo un mejor entendimiento de la distribución física y lógica del sistema. Se explican los diferentes patrones de arquitectura y de diseño utilizado en el desarrollo de la aplicación. También se aplica la prueba de Caja Negra al sistema, con el objetivo de evaluar el cumplimiento de los requisitos funcionales definidos anteriormente.

CONCLUSIONES GENERALES.

Una vez finalizado el desarrollo de este trabajo de diploma, se tiene como resultado un sistema informático que certifica o evalúa la arquitectura del software realizado en cualquier proyecto de la Universidad de las Ciencias Informáticas y permite conocer que tan adecuada es la AS diseñada para el sistema. El sistema informa donde está el riesgo, es decir, fortalezas y debilidades identificadas de la AS. Esto permitirá una mejora en la realización de este proceso evaluativo así como en la calidad de los productos realizados.

Por otra parte, se arriban a las siguientes conclusiones:

- Se capturaron los requisitos funcionales y no funcionales que debía tener el sistema. Esto sentó las bases para desarrollar todo el trabajo de manera organizada y poder dar cumplimiento a los objetivos perseguidos por el sistema.
- Se obtuvo una aplicación web, desarrollada en entornos libres y que da respuesta a la necesidad que la originó. Además, de ser un sistema capaz de informar al usuario los errores, inmediatamente que son producidos.
- Se realizó el diseño y la construcción del sistema teniendo en cuenta el patrón arquitectónico Modelo Vista Controlador, combinado con patrones de diseño.

Por todo lo antes expuesto se concluye que el objetivo y las tareas propuestos para el presente trabajo de diploma, se han alcanzado satisfactoriamente. Además, se incluyen una serie de recomendaciones que pueden ser útiles en el desarrollo de nuevas versiones.

RECOMENDACIONES.

Luego de haber analizado los resultados del presente trabajo de diploma se recomienda lo siguiente:

- Mejorar el diseño de la presentación del sistema.
- Perfeccionar y ampliar los servicios que brinda.
- Valorar por parte de la Universidad de las Ciencias Informáticas la utilización del sistema para evaluar las arquitecturas de los diferentes sistemas realizados por los proyectos de la universidad.

TRABAJOS CITADOS

1. Martín, Dayana Calvo San. ***Factibilidad del uso de métodos de evaluación de arquitectura en los proyectos productivos de la UCI.*** **Ciudad de la Habana : s.n., 2009.**

2. Reynoso, Carlos Billy. ***Introducción a la arquitectura de software.*** **Buenos Aires : s.n., 2004.**

3. Frank Buschmann, Regine Meunier , Hans Rohnert, Peter Sommerlad, Michael Stal. ***Pattern-Oriented Software Architecture.*** **New York : s.n., 1996.**

4. Tedeschi, Nicolás. **msdn.microsoft.com.** ***msdn.microsoft.com.*** **[En línea] [Citado el: 7 de Marzo de 2010.] http://msdn.microsoft.com/es-es/library/bb972240.aspx#XSLTsection122121120120.**

5. S.Tenenbaum, Andrew. ***Computer networks. Upper Saddle Rive.*** **2003.**

6. Laman, Craig. ***UML y Patrones.*** **México : s.n., 1999.**

7. José Enrique González Cornejo. **www.docirs.cl.** ***www.docirs.cl.*** **[En línea] [Citado el: 26 de Febrero de 2010.] http://www.docirs.cl/caracteristica_herramienta_uml.htm.**

8. Acuña, Karenny Brito. ***Selección de metodologías de desarrollo para aplicaciones web en la facultad de informática de la universidad de Cienfuegos.*** **Cienfuegos : s.n., 2009.**

9. Cordero., Jorge Luis. ***Metodologías Ágiles " Proceso Unificado Ágil (AUP)".*** **La Paz, EL Alto – Bolivia : s.n.**

10. Calderón Amaro, Dámaris Sarah. ***Metodologías Ágiles.*** **Trujillo, Perú : s.n., 2007.**

11. Alberto Hidalgo Reyes, Josué Vazquez Sorí. ***Modelación de Arquitecturas para Aplicaciones Empresariales en PHP.*** **Ciudad de la Habana : s.n., 2008.**

12. Alvarez, Miguel Angel. **desarrolloweb.com.** ***desarrolloweb.com.*** **[En línea] [Citado el: 2 de Junio de 2010.] http://www.desarrolloweb.com/articulos/392.php.**

13. Murugarren, Joaquín Gracia. **Web Estilo.** ***Web Estilo.*** **[En línea] 21 de Octubre de 2008. [Citado el: 13 de Marzo de 2010.] http://www.webestilo.com/mysql/intro.phtml.**

14. Mastrapa, Henry Yordan López. ***Propuesta de Arquitectura del Sistema de Registro de Visitante en la UCI.*** **Ciudad de La Habana : s.n., 2008.**

15. Dominguez, Rolando Ramiro Barrientos. ***Propuesta de Arquitectura del Sistema Gestión de Gestión de Fotos del Proyecto Control de Acceso.*** **Ciudad de la Habana : s.n., 2008.**

16. Ciberaula. ***Ciberaula.*** **[En línea] 2010. [Citado el: 5 de Junio de 2010.] http://linux.ciberaula.com/articulo/linux_apache_intro.**

17. Fabien Potencier, Francois Zaninotto. ***Symfony 1.2, la guia definitiva.*** **2008.**

18. www.postgresql-es.org. ***www.postgresql-es.org.*** **[En línea] 22 de Marzo de 2009. [Citado el: 26 de Marzo de 2010.] http://www.postgresql-es.org/sobre_postgresql.**

19. PostgreSQL. ***PostgreSQL.*** **[En línea] 2010. [Citado el: 6 de Junio de 2010.] http://www.postgresql.org/docs/8.1/interactive/preface.html#INTRO-WHATIS.**

BIBLIOGRAFÍA

1. Martín, Dayana Calvo San. ***Factibilidad del uso de métodos de evaluación de arquitectura en los proyectos productivos de la UCI.*** **Ciudad de la Habana : s.n., 2009.**

2. Reynoso, Carlos Billy. ***Introducción a la arquitectura de software.*** **Buenos Aires : s.n., 2004.**

3. Frank Buschmann, Regine Meunier , Hans Rohnert, Peter Sommerlad, Michael Stal. ***Pattern-Oriented Software Architecture.*** **New York : s.n., 1996.**

4. Tedeschi, Nicolás. **msdn.microsoft.com.** ***msdn.microsoft.com.*** **[En línea] [Citado el: 7 de Marzo de 2010.] http://msdn.microsoft.com/es-es/library/bb972240.aspx#XSLTsection122121120120.**

5. S.Tenenbaum, Andrew. ***Computer networks. Upper Saddle Rive.*** **2003.**

6. Laman, Craig. ***UML y Patrones.*** **México : s.n., 1999.**

7. José Enrique González Cornejo. **www.docirs.cl.** ***www.docirs.cl.*** **[En línea] [Citado el: 26 de Febrero de 2010.] http://www.docirs.cl/caracteristica_herramienta_uml.htm.**

8. Acuña, Karenny Brito. ***Selección de metodologías de desarrollo para aplicaciones web en la facultad de informática de la universidad de Cienfuegos.*** **Cienfuegos : s.n., 2009.**

9. Cordero., Jorge Luis. ***Metodologías Ágiles " Proceso Unificado Ágil (AUP)".*** **La Paz, EL Alto – Bolivia : s.n.**

10. Calderón Amaro, Dámaris Sarah. ***Metodologías Ágiles.*** **Trujillo, Perú : s.n., 2007.**

11. Alberto Hidalgo Reyes, Josué Vazquez Sorí. ***Modelación de Arquitecturas para Aplicaciones Empresariales en PHP.*** **Ciudad de la Habana : s.n., 2008.**

12. Alvarez, Miguel Angel. **desarrolloweb.com.** ***desarrolloweb.com.*** **[En línea] [Citado el: 2 de Junio de 2010.] http://www.desarrolloweb.com/articulos/392.php.**

13. Murugarren, Joaquín Gracia. **Web Estilo.** ***Web Estilo.*** **[En línea] 21 de Octubre de 2008. [Citado el: 13 de Marzo de 2010.] http://www.webestilo.com/mysql/intro.phtml.**

14. Mastrapa, Henry Yordan López. ***Propuesta de Arquitectura del Sistema de Registro de Visitante en la UCI.*** **Ciudad de La Habana : s.n., 2008.**

15. Dominguez, Rolando Ramiro Barrientos. ***Propuesta de Arquitectura del Sistema Gestión de Gestión de Fotos del Proyecto Control de Acceso.*** **Ciudad de la Habana : s.n., 2008.**

16. Ciberaula. ***Ciberaula.*** **[En línea] 2010. [Citado el: 5 de Junio de 2010.] http://linux.ciberaula.com/articulo/linux_apache_intro.**

17. Fabien Potencier, Francois Zaninotto. ***Symfony 1.2, la guia definitiva.*** **2008.**

18. www.postgresql-es.org. ***www.postgresql-es.org.*** **[En línea] 22 de Marzo de 2009. [Citado el: 26 de Marzo de 2010.] http://www.postgresql-es.org/sobre_postgresql.**

19. PostgreSQL. ***PostgreSQL.*** **[En línea] 2010. [Citado el: 6 de Junio de 2010.] http://www.postgresql.org/docs/8.1/interactive/preface.html#INTRO-WHATIS.**

20. Pressman, Roges S. ***Ingeniería de Software un Enfoque Práctico.*** **2005.**

21. Diana Delgado González, Emily Mejías Domínguez. ***Propuesta de Arquitectura para el Simulador del sistema Inmune J-IMMSIM 2.0.*** **Ciudad de la Habana : s.n., 2008.**

22. Torres, Patricio Letelier. **dsic.upv.es.** ***dsic.upv.es.*** **[En línea] 2 de Abril de 2004. http://users.dsic.upv.es/~letelier/pub/.**

23. Chacón, Julio César Rueda. ***Aplicación de la metodología RUP para el estándar J2EE.*** **Guatemala : s.n., 2006.**

24. Valdés, Damián Pérez. **Maestros del Web.** ***Maestros del Web.*** **[En línea] 2 de Noviembre de 2007. http://www.maestrosdelweb.com/principiantes/los-diferentes-lenguajes-de-programacion-para-la-web.**

25. James Rumbaugh, Ivar Jacobson , Grady Booch. ***El Lenguaje Unificado de Modelado.Manual de Referencia.*** **Madrid : s.n., 2005.**

GLOSARIO DE TÉRMINOS

API (Interfaz de Programación de Aplicaciones): Es un conjunto de funciones residentes en bibliotecas que permiten que una aplicación corra bajo un determinado sistema operativo.

Aplicación web: Sitio web que contiene páginas con contenido sin determinar parcialmente o en su totalidad. El contenido final de estas páginas se determina sólo cuando un visitante solicita una página del servidor web.

Framework: Término usado en programación orientada a objetos para definir un conjunto de clases que definen un diseño abstracto para solucionar un conjunto de problemas relacionados.

GRASP (Responsivility Asignment Software Patterns): Patrones de asignación de responsabilidades.

HTTP (Hypertext Transfer Protocol): Protocolo de Transferencia de Hipertexto.

Layout: Archivo, que también se denomina plantilla global, almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página.

Repositorio: Es un sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos**.**

RUP (Proceso de Desarrollo Unificado): Proceso de desarrollo de software, que constituye la metodología más usada para el análisis, implementación y documentación de sistemas con tecnología orientada a objetos.

Servidor web: Software que suministra páginas web en respuesta a las peticiones de los navegadores web.

Software: Equipamiento o soporte lógico de una computadora. Conjunto de componentes lógicos necesarios para hacer posible la realización de una tarea específica.

XP (Programación Extrema): Metodología ágil para el desarrollo de un software.

Triggers: En una Base de datos, es un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación de inserción (INSERT), actualización (UPDATE) o borrado (DELETE).

GOF (Gang of Four): Estas siglas indican “Banda de los cuatro” y es el nombre con el que se conoce a los patrones de diseño publicados en el libro Design Patterns (Patrones de Diseño).

ANEXOS

Anexo1

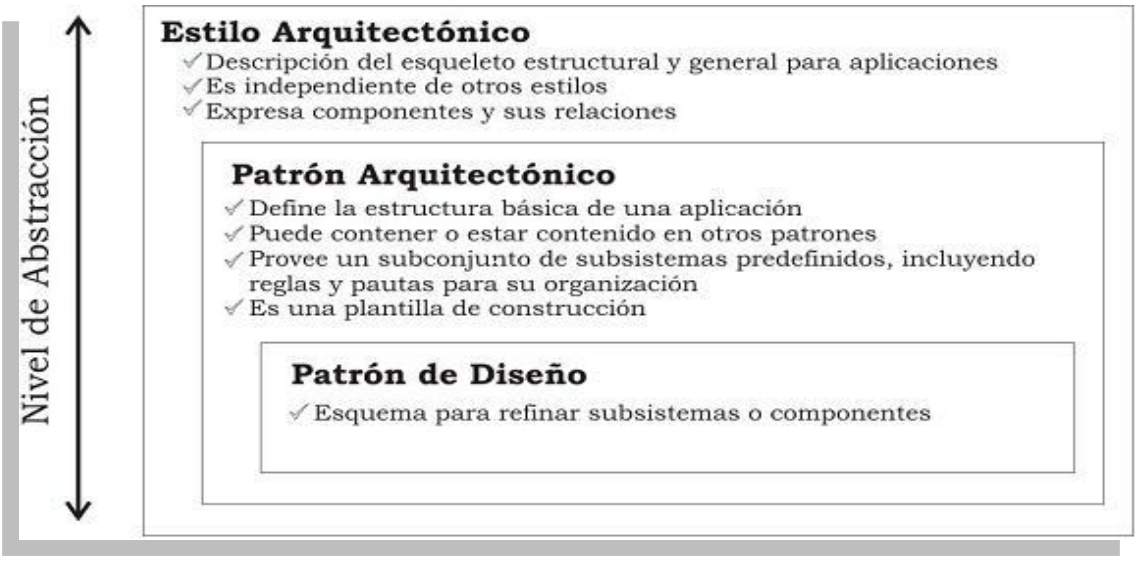

Figura 13 Nivel de abstracción.

Anexo 2

Estilo	Descripción	Atributos asociados	Atributos en conflicto
Datos Centralizados	Sistemas en los cuales cierto número de clientes accede y actualiza datos compartidos de un repositorio de manera frecuente.	Integrabilidad Escalabilidad Modificabilidad	Desempeño
Flujo de datos	El sistema es visto como una serie de transformaciones sobre piezas sucesivas de datos de entrada. El dato ingresa en el sistema, y fluye entre los componentes, de uno en uno, hasta que se le asigne un destino final (salida o repositorio).	Reusabilidad Modificabilidad Mantenibilidad	Desempeño
Maquinas Virtuales	Simulan alguna funcionalidad que no es nativa al hardware o software sobre el que está implementado.	Portabilidad	Desempeño
Llamada y Retorno	El sistema se constituye de un programa principal que tiene el control del sistema y varios subprogramas que se comunican con éste mediante el uso de programas.	Modificabilidad Escalabilidad Desempeño	Mantenibilidad Desempeño
Componentes Independientes	Consiste en un número de procesos u objetos independientes que se comunican a través de mensajes.	Modificabilidad Escalabilidad	Desempeño Integrabilidad

Figura 14 Estilos Arquitectónicos y Atributos de Calidad.

Anexo 3

Patrón Arquitectónico	Descripción	Atributos asociados	Atributos en conflicto
Capas	Consiste en estructurar aplicaciones que pueden ser descompuestas en grupos de subtareas, las cuales se clasifican de acuerdo a un nivel particular de abstracción.	Reusabilidad Portabilidad Facilidad de Prueba	Desempeño Mantenibilidad
Tuberías y Filtros	Provee una estructura para los sistemas que procesan un flujo de datos. Cada paso de procesamiento está encapsulado en un componente filtro (filter). El dato pasa a través de conexiones (pipes), entre filtros adyacentes.	Reusabilidad Mantenibilidad	Desempeño
Pizarrón	Aplica para problemas cuya solución utiliza estrategias no determinísticas. Varios subsistemas ensamblan su conocimiento para construir una posible solución parcial ó aproximada.	Modificabilidad Mantenibilidad Reusabilidad Integridad	Desempeño Facilidad de Prueba
Rompedor	Puede ser usado para estructurar sistemas de software distribuido con componentes desacoplados que interactúan por invocaciones a servicios remotos. Un componente broker es responsable de coordinar la comunicación, como el reenvío de solicitudes, así como también la transmisión de resultados y excepciones.	Modificabilidad Portabilidad Reusabilidad Escalabilidad Interoperabilidad	Desempeño
Modelo-Vista-Controlador	Divide una aplicación interactiva en tres componentes. El modelo (model) contiene la información central y los datos. Las vistas (view) despliegan información al usuario. Los controladores (controlers) capturan la entrada del usuario. Las vistas y los controladores constituyen la interfaz del usuario.	Funcionalidad Mantenibilidad	Desempeño Portabilidad
Presentación-Abstracción-Control	Define una estructura para sistemas de software interactivos de agentes de cooperación organizados de forma jerárquica. Cada agente es responsable de un aspecto específico de la funcionalidad de la aplicación y consiste de tres componentes: presentación, abstracción y control.	Modificabilidad Escalabilidad Integrabilidad	Desempeño Mantenibilidad
Micronúcleo	Aplica para sistemas de software que deben estar en capacidad de adaptar los requerimientos de cambio del sistema. Separa un núcleo funcional mínimo del resto de la funcionalidad y de partes específicas pertenecientes al cliente.	Portabilidad Escalabilidad Confiabilidad Disponibilidad	Desempeño
Reflexión	Provee un mecanismo para sistemas cuya estructura y comportamiento cambia dinámicamente. Soporta la modificación de aspectos fundamentales como estructuras tipo y mecanismos de llamadas a funciones.	Modificabilidad	Desempeño

Figura 15 Descripción de los patrones arquitectónicos y atributos de calidad.

Anexo 4

Patrón de Diseño	Descripción	Atributos asociados	Atributos en conflicto
Todo-Parte	Ayuda a constituir una colección de objetos que juntos conforman una unidad semántica.	Reusabilidad Modificabilidad	Desempeño
Maestro-Esclavo	Un componente maestro (master) distribuye el trabajo a los componentes esclavos (slaves). El componente maestro calcula un resultado final a partir de los resultados arrojados por los componentes esclavos.	Escalabilidad Desempeño	Portabilidad
Proxy	Los clientes asociados a un componente se comunican con un representante de éste, en lugar del componente en sí mismo.	Desempeño Reusabilidad	Desempeño
Command Procesor	Separa las solicitudes de un servicio de su ejecución. Maneja las solicitudes como objetos separados, programa sus ejecuciones y provee servicios adicionales como el almacenamiento de los objetos solicitados, para permitir que el usuario pueda deshacer alguna solicitud.	Funcionalidad Modificabilidad Facilidad de Prueba	Desempeño
Manejador de Vista	Ayuda a manejar todas las vistas que provee un sistema de software. Permite a los clientes abrir, manipular y eliminar vistas. También coordina dependencias entre vistas y organiza su actualización.	Escalabilidad Modificabilidad	Desempeño
Forwarder-Receiver	Provee una comunicación transparente entre procesos de un sistema de software con un modelo de interacción punto a punto (peer to peer).	Mantenibilidad Modificabilidad Desempeño	Configurabilidad
Client-Dispatcher-Server	Introduce una capa intermedia entre clientes y servidores, es el componente despachador (dispatcher). Provee una ubicación transparente por medio de un nombre de servicio, y esconde los detalles del establecimiento de una conexión de comunicación entre clientes y servidores.	Configurabilidad Portabilidad Escalabilidad Disponibilidad	Desempeño Modificabilidad
Publisher Subscriber	Ayuda a mantener sincronizados los componentes en cooperación. Para ello, habilita una vía de propagación de cambios: un editor (publisher) notifica a los suscriptores (suscribers) sobre los cambios en su estado.	Escalabilidad	Desempeño

Figura 16 Patrones de diseño y atributos de calidad.

Anexo 5

Atributo de Calidad	Descripción
Disponibilidad	Es la medida de disponibilidad del sistema para el uso (Barbacci et al, 1995).
Confidencialidad	Es la ausencia de acceso no autorizado a la información (Barbacci et al, 1995).
Funcionalidad	Habilidad del sistema para realizar el trabajo para el cual fue concebido (Kazman et al., 2001).
Desempeño	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria. (IEEE 610.12).
Confiabilidad	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo (Barbacci et al., 1995).
Seguridad externa	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información (Barbacci et al, 1995).
Confiabilidad	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo (Barbacci et al., 1995).
Seguridad interna	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos (Kazman et al., 2001).

Figura 17 Descripción de atributos de calidad observables vía ejecución.

Anexo 6

Atributo de Calidad	Descripción
Configurabilidad	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema (Bosch et al., 1999).
Integrabilidad	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados. (Bass et al. 1998)
Integridad	Es la ausencia de alteraciones inapropiadas de la información (Barbacci et al., 1995).
Interoperabilidad	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de integrabilidad (Bass et al. 1998)
Modificabilidad	Es la habilidad de realizar cambios futuros al sistema. (Bosch et al. 1999).
Mantenibilidad	Es la capacidad de someter a un sistema a reparaciones y evolución (Barbacci et al., 1995). Capacidad de modificar el sistema de manera rápida y a bajo costo (Bosch et al. 1999).
Portabilidad	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos (Kazman et al., 2001).
Reusabilidad	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones (Bass et al. 1998).
Escalabilidad	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental (Pressman, 2002).
Capacidad de Prueba	Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba (Bass et al. 1998).

Figura 18 Descripción de atributos de calidad no observables vía ejecución.