UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD 9

Arquitectura para el Sistema de Gestión de Procesos para la Dirección de Televisión Universitaria.

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

Autor:

Yoel Falero Vento

Tutor:

Ing. Yuset Martínez Carballo

Ciudad de la Habana, Mayo, 2010
Año 52 de la Revolución

DATOS GENERALES.

GENERALES DE LA INVESTIGACIÓN

Título: Arquitectura para el Sistema de Gestión de Procesos para la Dirección de Televisión Universitaria.

Clasificación: Informe de investigación práctico del desarrollo de un rol de producción determinado.

DATOS DEL DIPLOMANTE

Nombre y apellidos: Yoel Falero Vento.

Sexo: M

Grupo: 9505

Correo electrónico: yvento@estudiantes.uci.cu

DATOS DEL TUTOR

Nombre y apellidos: Yuset Martínez Carballo

Sexo: M

Institución: Universidad de las Ciencias Informáticas

Correo electrónico: ycarballo@uci.cu

Teléfono del trabajo: 8352115

Teléfono particular: 547443

Categoría docente: Adiestrado**.**

Grado científico: Ingeniero**.**

Cargo del trabajador: Profesor**.**

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

Año de graduación: 2008

Institución donde se graduó: Universidad de las Ciencias Informáticas

DEDICATORIA

A mi "nenos" Nancyta y Valentín

A mis hermanos y toda mi familia

que es bastante grande.

A Ireydis, mi "bandía".

A mi otra familia Bebe, Negro,

Gordita, Cristian, y Salet.

AGRADECIMIENTOS

- *A mis padres Nancy y Vale por haber luchado tanto para lograr nuestro resultado y confiar en mí a pesar de los tantos obstáculos enfrentados durante estos 5 años.*
- *A todos mis hermanos y familia por darme el apoyo necesario para seguir adelante. En especial, a Yosbel por ser mi guía durante nuestro transcurso en la universidad.*
- *A mi novia Ireydis por darme el cariño que he necesitado para lograr las cosas tan lindas que hemos logrado y las que lograremos juntos. Por confiar en mi, comprenderme y ayudarme en momentos difíciles por los que he pasado.*
- *A mi amigo Reyalex(el tío), por ser mi amigo. Por las tantas cosas que hizo para posibilitar la realización de este trabajo. Por ser el profesor ejemplar que es. Por no flaquear en ningún momento cuando necesite su ayuda.*
- *A Yuset por ser un excelente tutor y ayudarme en todo momento.*
- *A "Osama" por ser lo suficiente critico como oponente de este trabajo.*
- *Al Yoa por ser un gran amigo y ayudarme a pesar de su reducido tiempo.*
- *A mi grupo musical "Los Intrusos" por pasar juntos esos grandes momentos en escena.*
- *A todos mis amigos de siempre que me llevan de verdad y se han portado conmigo estelares.*

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los __ días del mes de __ del año 2009.

_____ _____

Firma del Autor Firma del Tutor

OPINIÓN DEL TUTOR

RESUMEN

Debido la gran cantidad de servicios que brinda la Dirección de Televisión Universitaria perteneciente a la Universidad de las Ciencias Informáticas en Cuba, diariamente se efectúan muchos procesos en todos los departamentos de este centro. La información que se gestiona en el mismo, posee un gran volumen por lo que resulta tediosa la ejecución de muchos de sus procesos. Por estas razones se decide la realización del Sistema de Gestión de Procesos para la Dirección de Televisión Universitaria (SGPDTU), facilitando así el trabajo en dicho lugar.

La realización de este trabajo pretende conformar la propuesta de una arquitectura de software robusta y confiable que permita el desarrollo de un sistema modificable, interoperable, seguro, funcional y disponible, que asegure el mantenimiento, flexibilidad y reusabilidad del SGPDTU, apoyándose en el uso de herramientas libres.

Para esta propuesta pues se hizo necesario un análisis partiendo fundamentalmente de los principales conceptos, estilos, y patrones arquitectónicos para así definir un conjunto de herramientas y lenguajes necesarios para el desarrollo de este sistema. De acuerdo con los requerimientos del mismo y el nivel de complejidad, también se abordaron temas relacionados con las tecnologías de desarrollo del software definiéndose así la adecuada para su desarrollo.

Una vez definida la arquitectura pues se propusieron dos estrategias de evaluación de la arquitectura de software con el objetivo de identificar los riesgos y fortalezas de la propuesta y poder tomar medidas correctivas a tiempo.

ÍNDICE DE CONTENIDO

INTRODUCCIÓN....1
CAPÍTULO 1: Fundamentación Teórica....3
1.1 Introducción....3
1.2 Arquitectura de Software....3
1.3 Patrones....5
1.3.1 Clasificación de los Patrones para la Ingeniería de Software....5
1.4 Estilos y Patrones Arquitectónicos....6
1.4.2 Patrones Arquitectónicos....8
1.5 Conclusiones del Capítulo....13
CAPÍTULO 2: Herramientas para la Solución....14
2.1 Introducción....14
2.2 Metodología de Desarrollo de Software....14
2.2.1 Extreme Programing (XP)....15
2.2.2 Metodología Rational Unified Process (RUP)....16
2.3 Lenguaje Unificado de Modelado (UML)....19
2.4 Herramienta CASE....20
2.4.1 Rational Rose Enterprise....20
2.4.2 Visual Paradigm....21
2.5 Lenguajes de Programación del Lado del Servidor....22
2.5.1 PHP....22
2.6 Frameworks....22
2.6.1 Symfony....23
2.7 Sistema Gestor de Base de Datos....25
2.7.1 Oracle....26
2.7.2 MySQL....26
2.7.3 PostgreSQL....27
2.8 Conclusiones del Capítulo....28
Capitulo 3: Propuesta de la Arquitectura del Sistema....29
3.1 Introducción....29
3.2 Estructura del Equipo de Desarrollo....29
3.2.1 Configuración de los puestos de trabajos por roles....29
3.3 Subsistemas del Sistema....30
3.4 Requerimientos no funcionales del sistema....30
3.4.1 Requerimientos de Software....31
3.4.2 Requerimientos de Hardware....31
3.4.3 Restricciones en el diseño y la implementación....31
3.4.4 Apariencia o Interfaz Externa....32

3.4.5 Requerimientos de confidencialidad. 32
3.4.6 Requerimientos de disponibilidad. 32
3.4.7 Requerimientos de Seguridad. 32
3.4.8 Rendimiento. 33
3.4.9 Eficiencia. 33
3.5 Vistas de la Arquitectura de Software. 33
3.5.1 Vista de Caso de Uso. 34
3.5.2 Vista Lógica. 37
3.5.3 Vista de procesos. 38
3.5.4 Vista de Implementación. 38
3.5.5 Vista de Despliegue. 39
3.6 Conclusiones del Capítulo. 40
CAPÍTULO 4: Evaluación de la Arquitectura del Sistema. 41
4.1 Introducción. 41
4.2 ¿Por qué evaluar una Arquitectura? 41
4.3 ¿Cuándo evaluar una Arquitectura? 41
4.4 Cualidades a evaluar en la Arquitectura de Software. 42
4.5 Técnicas de Evaluación. 44
4.6 Métodos de prueba de Arquitectura de Software. 45
4.6.1 SAAM. 46
4.6.2 ADR. 47
4.6.3 ATAM. 47
4.6.3 ARID (Active Review Intermediate Designs). 47
4.7 Propuesta de Estrategia de Evaluación de la Arquitectura. 48
4.7.1 Estrategia de evaluación temprana (ARID). 48
4.8 Conclusiones del Capítulo. 53
CONCLUSIONES GENERALES. 54
RECOMENDACIONES 55
Trabajos Citados 56
BIBLIOGRAFÍA CONSULTADA 58
GLOSARIO DE TÉRMINO 60
ANEXO 1 61
ANEXO 2 62

Índice de Tablas y Figuras

Figura 1: Patrones de Capas.....9
Figura 2 : Arquitectura Basada en Servicios (SOA). 10
Figura 3: Arquitectura Orientada en Objetos. 13
Figura 4: Flujos de Trabajo, Fases y Esfuerzo por Iteración.....18
Figura 5: Vistas de la AS. 34
Figura 6: Vista de Casos de Uso del Sistema.....36
Figura 7: Vista Lógica.....37
Figura 8: Vista de Implementación.....39
Figura 9: Diagrama de Despliegue. 40
Figura 10: Clasificación de las Técnicas de Evaluación.....45
Figura 11: Diagrama de Clases para Gestionar Plan de Producción. 62
Figura 12: Fases de la Metodología XP.....63

INTRODUCCIÓN

La información es un conjunto organizado de datos procesados, que forman una idea o mensaje sobre un determinado fenómeno. Los datos se perciben, se integran y generan la información necesaria para producir el conocimiento, siendo este último quien permite finalmente tomar decisiones para realizar las acciones cotidianas que aseguran la existencia. Actualmente en el mundo, el manejo de la información ha formando parte de muchas esferas de la sociedad, pues su gestión ha sido las bases de muchos logros obtenidos. Para la gestión de información se fue haciendo necesario automatizar sus procesos mediante las Tecnologías de Información y las Comunicaciones (TIC) facilitando así un mejor funcionamiento.

Cuba a pesar de ser un país subdesarrollado no se queda atrás pues poco a poco ha ido trabajando en la informatización de esferas como la economía, y educación fundamentalmente, mejorando con esto el trabajo en las mismas.

Dando un gran aporte a lo antes mencionado se encuentra la Universidad de las Ciencias Informáticas (UCI), la cual a pesar de formar ingenieros informáticos se dedica a la producción de software para el consumo nacional e internacional, colaborando así con la economía del país. Debido a su gran cantidad de estudiantes y trabajadores, y con el objetivo de mejorar el sistema de aprendizaje e información a los mismos, se crea la Dirección de Televisión Universitaria (DTU).

La DTU debido a la gran cantidad de servicios que brinda, diariamente se efectúan muchos procesos en todos sus departamentos. La información que se gestiona posee un gran volumen por lo que resulta tediosa la ejecución de algunos procesos, siendo esto una de las causas que conllevan que en ocasiones no se efectúen con la eficiencia que requieren, viéndose afectados con esto los servicios correspondientes que ofrece este centro.

Por estas razones se hizo necesaria la automatización de los procesos mediante un sistema de gestión que facilitara el trabajo de las personas involucradas en estas funciones, haciendo así más eficiente sus actividades correspondientes. Una vez aceptada y confirmada la realización de este sistema pues se le asigna dicha tarea a la Facultad 9 y específicamente al Departamento de Señales Digitales.

Con la elaboración de este trabajo quedarán establecidas las herramientas a usar para realizar este sistema, así como los estilos y patrones arquitectónicos de acuerdo a las características de la aplicación.

Como **problema a resolver** se presenta la necesidad de una arquitectura sólida que permita el desarrollo, mantenimiento, y reusabilidad del Sistema de Gestión de Procesos de la Dirección de Televisión Universitaria (SGPDTU).

A partir de esta situación problémica se determina como **objetivo general** definir una arquitectura de software para el proyecto SGPDTU que permita un sistema funcional, escalable y seguro, usando herramientas libres para el desarrollo de este sistema. Teniendo como **objetivos específicos** identificar una arquitectura robusta que sea aplicable al sistema, y las herramientas de software libre que permitan el desarrollo del mismo.

Para este trabajo se tiene como **objeto de estudio** las arquitecturas de software usadas para el desarrollo de un sistema gestor de procesos, centrando el **campo de acción** en la arquitectura de software para el Sistema de Gestión de Procesos para la Dirección de Televisión Universitaria. Teniendo como **idea a defender** el diseño de una arquitectura robusta que cumpla con los requerimientos establecidos y atributos de calidad para el Sistema de Gestión de Procesos para la Dirección de Televisión Universitaria permitirá el desarrollo del sistema.

Para dar cumplimiento al objetivo general que se explica en este trabajo se trazaron las siguientes **tareas:**

1. Definir los estilos y patrones de la arquitectura de software a utilizar.
2. Identificar las herramientas y tecnologías más convenientes para el desarrollo del software.
3. Especificar la metodología de prueba aplicable a la arquitectura definida.
4. Proponer la arquitectura de software a utilizar en el proyecto Sistema de Gestión de Procesos para la Dirección de Televisión Universitaria.
5. Comparar diferentes sistemas similares al que se va a desarrollar, teniendo en cuenta su arquitectura.

Con el cumplimiento de dichas tareas se deben obtener los siguientes **resultados**:

- Definición de las herramientas a utilizar para el desarrollo del sistema.
- Artefactos desarrollados por el rol Arquitecto durante las fases de desarrollo.
- Definición del tipo de arquitectura.
- Definición de la configuración de los puestos de trabajo.

Los procesos realizados en la DTU son semejantes a los de cualquier otra institución de comunicación audiovisual por lo que este sistema de gestión de procesos pudiera servir a otros centros como el Instituto Cubano de Radio y Televisión además de permitir poder adicionarle más funcionalidades en caso que se necesiten. Esto es posible ya que estará basado en una arquitectura flexible a cambios.

CAPÍTULO 1: Fundamentación Teórica.

1.1 Introducción.

En este capítulo se abordaran temas relacionados con el análisis de la arquitectura de software, definiéndose los estilos y patrones arquitectónicos con sus características principales, para así ver cuales satisfacen las necesidades para el Sistema de Gestión de Procesos para la Dirección de Televisión Universitaria.

1.2 Arquitectura de Software.

La Arquitectura de Software (AS) es un campo de investigación relativamente nuevo. Aunque según la historia el termino arquitectura de software se comenzó a usar en 1990, y se dio gracias al surgimiento de sistemas muy grandes.

La definición de la AS es algo compleja pues no todos los arquitectos tienen el mismo concepto respecto a la misma a pesar de tener aspectos en común. Para argumentar mejor el conocimiento es necesario conocer la opinión de los principales autores y conocedores de este tema.

"La AS es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones". (1)

La arquitectura de software de un programa o sistema informático es la estructura o las estructuras del sistema, que incluyen elementos de software, las propiedades externamente visibles de esos elementos y las relaciones entre ellos. (2)

"La AS 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." (3)

De modo general se pudiera llegar a la conclusión que la AS es en una vista estructural de alto nivel que define estilos o combinación de estilos para dar una solución que se concentra principalmente en los requerimientos no funcionales del sistema teniendo la responsabilidad de:

- Definir los módulos principales
- Definir las responsabilidades que tendrá cada uno de estos módulos
- Definir la interacción que existirá entre dichos módulos
- Control y flujo de datos
- Protocolos de interacción y comunicación
- Determinar las herramientas adecuadas

Es importante resaltar que la arquitectura afecta a todos los relacionados con el proyecto, afecta a los clientes, al gerente, al equipo de desarrollo, al equipo de pruebas, etc. Cada stakeholder[1] se preocupa por partes específicas del sistema, y esto se ve reflejado en la arquitectura del sistema. Provee un lenguaje mediante el cual los stakeholders comprenden el sistema y se comunican para tomar decisiones importantes.

Las razones por las cuales la arquitectura de software es importante para los grandes sistemas de software son las siguientes:

- Comunicación entre stakeholders: Al ser la arquitectura de software la representación de una abstracción de alto nivel de un sistema, permite que la mayoría de los estos tomen decisiones y halla un entendimiento mutuo.

- Decisiones de diseño: La arquitectura de software permite hacer diseños que se acomoden a las necesidades del sistema, a los requerimientos funcionales y los no funcionales.

- Abstracción transferible de un sistema: La arquitectura de software muestra la estructura de todo un sistema, esto permite que se haga un re-uso de los diferentes diseños en sistemas con requerimientos similares.

[1] **Persona que afecta o se ve afectada por las actividades de una organización**

1.3 Patrones.

Varios autores han comentado sobre el concepto de patrones. Dentro de los más conocidos se destaca Christopher Alexander el autor del libro “The Timeless Way of Building” el cual lo definió de la forma siguiente:

“Como un elemento en el mundo, cada patrón es una relación entre cierto contexto, cierto sistema de fuerzas que ocurre repetidas veces en ese contexto y cierta configuración espacial que permite que esas fuerzas se resuelvan. Como un elemento de lenguaje, un patrón es una instrucción que muestra la forma en que esta configuración espacial puede usarse, una y otra vez, para resolver ese sistema de fuerzas, donde quiera que el contexto la torne relevante. El patrón es, en suma, al mismo tiempo una cosa que pasa en el mundo y la regla que nos dice cómo crear esa cosa y cuándo se debe crear. Es tanto un proceso como una cosa; tanto una descripción de una cosa que está viva como una descripción del proceso que generará esa cosa”. (4)

“Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de la solución a ese problema, de tal manera que puedes usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces”

1.3.1 Clasificación de los Patrones para la Ingeniería de Software.

Los patrones se definen en:

- **Patrones arquitectónicos**: basados sobre los aspectos fundamentales de la estructura de un sistema. Especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes. Es decir son aquellos que organizan la estructura del software.

- **Patrones de diseño**: se enmarcan sobre aspectos relacionados con el diseño de los subsistemas. Por tanto se centran en aspectos más específicos.

- **Idiomas:** patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto. (1)

1.4 Estilos y Patrones Arquitectónicos.

Los estilos y patrones ayudan al arquitecto a definir la composición y el comportamiento del sistema de software, y una combinación adecuada de ellos permite alcanzar los requerimientos de calidad permitiendo de esa manera que el sistema de software a construir perdure en el tiempo.

1.4.1 Estilos Arquitectónicos.

Garlan y Shaw definen un estilo arquitectónico como el vocabulario de los componentes y conectores que pueden ser utilizados en instancias de los mismos, junto con una serie de limitaciones sobre la forma en que pueden ser combinados. Estos pueden incluir limitaciones topológicas y descripciones de la arquitectura. (2)

También es definido como una familia de sistemas de software en términos de un patrón de organización estructural, que define un vocabulario de componentes y tipos de conectores, y un conjunto de restricciones de cómo pueden ser combinadas. Para muchos estilos puede existir uno o más modelos semánticos que especifiquen cómo determinar las propiedades generales del sistema partiendo de las propiedades de sus partes. (3)

Un estilo describe una categoría de sistema que abarca un conjunto de componentes que realizan una función requerida por el sistema, un conjunto de conectores que posibilitan la comunicación, la coordinación y cooperación entre los componentes, las restricciones que definen como se integran los componentes para conformar el sistema, y los modelos semánticos que facilitan al diseñador el entendimiento de todas las partes del sistema. El estilo arquitectónico es también un patrón de construcción**.** (4)

Los estilos conjugan las cuatro “C” de la AS (componentes, conectores, configuraciones y restricciones), por sus siglas en inglés. (5).

De modo general se puede llegar a la conclusión de que los estilos arquitectónicos:

- Sirve para sintetizar estructuras de soluciones que luego serán refinadas a través del diseño.

- Un estilo contempla los cuatro fundamentos elementales de la AS(componentes, conectores, configuraciones y restricciones) viéndose aquí el gran peso que tienen en el proceso de determinación de una AS
- Definen los patrones posibles de las aplicaciones, evitando errores arquitectónicos.

A continuación se muestran los distintos ejemplos de estilos arquitectónicos:

- **Estilos Centrados en Datos**
  - Arquitecturas de Pizarra o Repositorio.
- **Estilos de Flujo de Datos**
  - Tubería y filtros.
- **Estilos Peer-to-Peer**
  - Arquitecturas Basadas en Eventos.
  - Arquitecturas Orientadas a Servicios (SOA).
- **Estilos de Código Móvil**
  - Arquitectura de Máquinas Virtuales.
  - Arquitecturas Basadas en Recursos.
- **Estilos Heterogéneos**
  - Sistema de Control de Procesos
  - Arquitecturas Basadas en Atributos
- **Estilos Derivados**
  - C2
  - GenVoca
  - Rest
- **Estilos de Llamada y Retorno**
  - Modelo-Vista-Controlador (MVC).
  - Arquitecturas en Capas.
  - Arquitecturas Orientadas a Objetos.
  - Arquitecturas Basadas en Componentes.

La familia de estilos “Llamada y Retorno”, enfatiza la modificación y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. En este grupo pudiese estar la línea base por la que se fuese a seguir la AS de este sistema después de una intensa investigación sobre el mismo.

1.4.2 Patrones Arquitectónicos.

Los patrones de arquitectura se pueden ver como la descripción de un problema en particular y recurrente de diseño, que aparece en contextos de diseño arquitectónicos específicos, y representa un esquema genérico demostrado con éxito para su solución. El esquema de solución se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades y desarrollos, así como también la forma en que estos colaboran entre sí. (1)

Viendo lo antes expresado se puede llegar a la conclusión que los patrones arquitectónicos se definen para la estructura fundamental del sistema siendo a través de los mismos que se determina la estructura de los subsistemas con sus funcionalidades y relaciones entre ellos

Patrón en Capas.

El patrón en capas es definido como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. (2)
En este patrón los componentes se estructuran en niveles o capas donde cada nivel invoca sólo al nivel inferior y las interfaces entre capas están claramente definidas. Contribuye a la disminución del acoplamiento, favorece la portabilidad y la sustitución de componentes y proporciona un alto nivel de abstracción.
Capas que conforman este patrón:

- **Capa de Presentación:** contiene todos los elementos de la interfaz de la aplicación, permitiendo la interacción entre los usuarios y el software mediante sus ventanas, menú, gráficos, reportes, etc.
- **Capa de Lógica de Negocio:** en ella están implícitos los procesos del sistema. Los datos que se envían desde la capa de presentación aquí serán validados, y se determinará el proceso que conlleva el mismo para el funcionamiento del sistema
- **Capa de Acceso a Datos:** esta sirve de enlace entre la capa lógica de negocio y la base de datos. En esta capa existen componentes que hacen transparente el acceso a la base de datos siendo

esta la causa por la que sea el mejor lugar para implementar los objetos de acceso para permitiendo así las funcionalidades de insertar, eliminar buscar, actualizar y información de la base de datos.

- **Capa de Seguridad:** Se relaciona con la administración del sistema. Esta capa es de gran importancia ya que en ella se definen los permisos que tendrá cada usuario para acceder a la aplicación, teniendo como objetivo fundamental alcanzar la máxima seguridad del sistema.

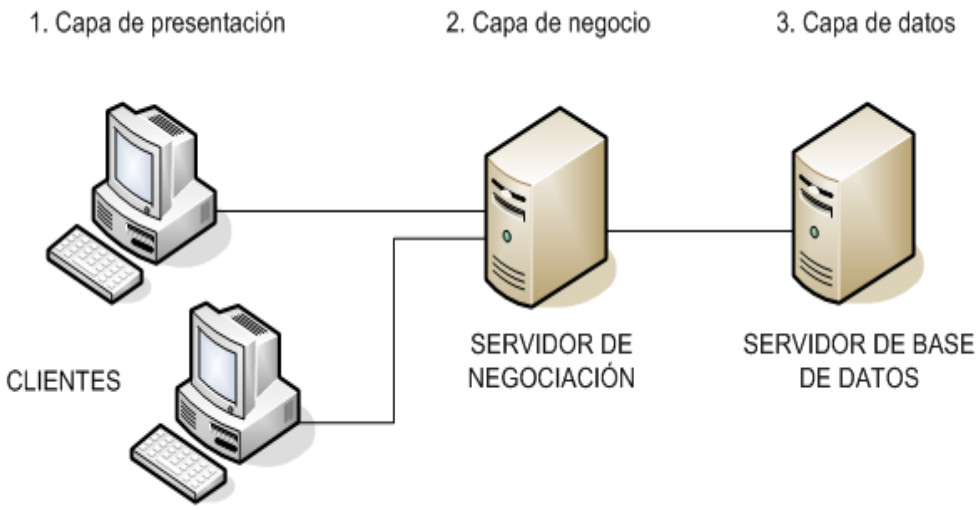

Figura 1: Patrones de Capas.

Este patrón es muy usado en el mundo debido a las siguientes ventajas:

- Para la implementación permite la división de un problema complejo en una secuencia de pasos crecientes que lo hace más entendible y organizado.
- Permite optimizaciones y refinamientos sin afectar el sistema en conjunto.
- Permite la reutilización ya que se puede utilizar implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes.

Arquitectura Orientada a Servicios (SOA).

Las Arquitecturas Orientadas a Servicios (SOA) están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma y del lenguaje de programación.

Un servicio web es un sistema de software diseñado para soportar interacción máquina a máquina sobre una red. Posee una interfaz descripta en un formato procesable por máquina (específicamente WSDL). Otros sistemas interactúan con el servicio de una manera prescripta por su descripción utilizando mensajes SOAP[2], típicamente transportados usando HTTP. (6)

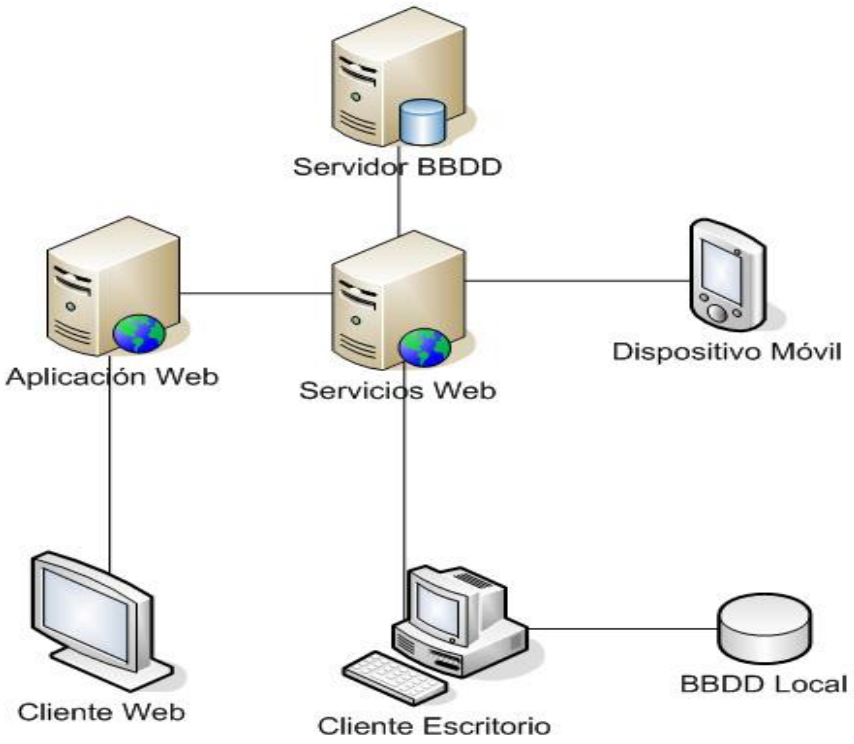

Figura 2 : Arquitectura Basada en Servicios (SOA).

[2] Formato de mensajes comunicado sobre el transporte que por defecto es HTTP

Ventajas:

- Cualquier cambio en la implementación no afectaría siempre que se mantenga la interfaz lo que la hace ser flexible
- La reutilización de los servicios de negocios.
- Los clientes y servicios se comunican independientemente de la plataforma en que radican.
- Debido al bajo acoplamiento de estos servicios, las aplicaciones que los usan son escalables ya que existe poca dependencia entre las aplicaciones clientes y los servicios que usan.
- Mejoran productividad de empleados.
- Proporcionan una mayor capacidad de respuesta a los clientes.
- Le da más seguridad al sistema**.**

Desventajas:

- Los tiempos de llamado no son despreciables, gracias a la comunicación de la red, tamaño de los mensajes, entre otros.
- La respuesta del servicio es afectada directamente por aspectos externos como problemas en la red, configuración, etc.
- Debe manejar comunicaciones no confiables, mensajes impredecibles, reintentos, mensajes fuera de secuencia, entre otros.

SOA no es algo que pueda ser de mucho provecho para empresas pequeñas. Solamente tiene sentido en caso de que el tamaño del departamento de tecnologías de la información sea considerable y esté formado, al menos, por más de un sistema primario. Es decir, su mayor beneficio se ubica dentro de sistemas complejos y heterogéneos. Dentro de las grandes corporaciones, con sistemas tecnológicos complejos y heterogéneos, SOA es de gran utilidad.

Arquitectura Orientada a Objetos (AOO).

En AOO los componentes del estilo se basan en principios orientados a objetos donde se puede realizar polimorfismo, encapsulamiento, y herencia. Las unidades de modelado, diseño e implementación, los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación. En tantos componentes, los objetos interactúan a través de invocaciones de funciones y procedimientos.

Los objetos representan una clase de componentes que ellos llaman managers, debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos. (2)

Entre las cualidades fundamentales se encuentran las siguientes:

- Puede modificar la implementación de un objeto sin afectar a sus clientes.
- Posibilita descomponer problemas en colecciones de agentes en interacción.
- Un objeto es ante todo una entidad reutilizable en el entorno de desarrollo.
- Las interfaces están separadas de la implementación. La distribución de objetos es transparente, apenas importa si los objetos son locales o remotos.
- Como los objetos interactúan a través de invocaciones de funciones y procedimientos.

Entre las limitaciones, el principal problema del estilo se manifiesta en el hecho de que para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad.

Patrón Modelo-Vista-Controlador (MVC).

Es un patrón de AS que separa los datos, interfaz de usuario, y la lógica de control de una aplicación en tres componentes distintos.

Modelo: es la representación específica de los datos e información con que el sistema interactúa.
Vista: es la encargara de mostrar la información e interactuar con el usuario.
Controlador: responde a eventos, usualmente acciones de usuario e invoca cambios en el modelo y probablemente en la vista.

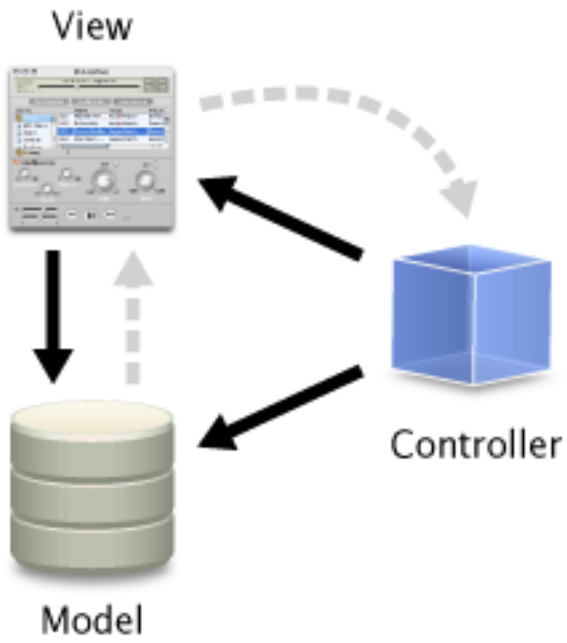

Figura 3: Arquitectura Orientada en Objetos.

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo independientemente de la representación visual.

Desde el punto de vista de la complejidad este patrón introduce nuevos niveles de dirección por lo que aumenta ligeramente la complejidad de la solución. Como consecuencia de las frecuentes actualizaciones, el hecho de desacoplar el modelo de la vista no significa que los desarrolladores del modelo puedan ignorar la naturaleza de las vistas. Si el modelo experimenta cambios frecuentes, por ejemplo, podría desbordar las vistas con una amplia actualización de requerimientos.

1.5 Conclusiones del Capítulo.

Con la conclusión de este capítulo se abordaron temas relacionados con la AS, entre ellos los estilos y patrones arquitectónicos, su clasificación y caracterización de los más utilizados para tomar posición de los idóneos para el desarrollo del sistema.

CAPÍTULO 2: Herramientas para la Solución.

2.1 Introducción.

En el presente capítulo se le dará tratamiento a las distintas metodologías que se pudieran usar en función del desarrollo del sistema. También se hará una profunda investigación sobre las tecnologías y herramientas más convenientes para este proyecto. Como resultado se tendrá definida la metodología y tecnología a usar como son las herramientas CASE, el lenguaje de programación, gestor de bases de datos y otras herramientas necesarias para lograr la evolución del proyecto.

2.2 Metodología de Desarrollo de Software.

Durante el proceso de desarrollo de un sistema con este nivel de complejidad se hace necesaria una metodología que soporte todo el proceso. Las metodologías de desarrollo de software posibilitan lograr los productos con una mayor calidad y eficiencia, asegurando su desarrollo y mantenimiento pues es una guía basada en buenas prácticas que organizan el trabajo.

La metodología de desarrollo se define como un conjunto de procedimientos, técnicas, herramientas, y un soporte documental que ayuda a los desarrolladores a realizar nuevos software. (7)

Se conforman en dos grupos fundamentales: agiles, y tradicionales o robustas. Las tradicionales o robustas confieren gran peso a la planificación, conceptualización y descripción del sistema que se pretende realizar. De esta forma garantizan que al comenzar la implementación esté bien definido cada elemento a desarrollar. Este tipo de metodologías se ajusta para proyectos a largo plazo de gran envergadura, con equipos de desarrollo numerosos donde la organización sea fundamental. También es recomendable cuando se requiera una documentación amplia que detalle cada elemento para lograr un entendimiento posterior del software u otras razones.

Las metodologías ágiles o livianas eliminan el burocratismo de las robustas, que en ocasiones resulta contraproducente emplearlas. Estas se centran en la capacidad de las personas involucradas en el proceso, evitando ir al detalle en cada paso, pero obteniendo el mayor fruto del trabajo de cada integrante. Perfecta para equipos de desarrollo con gran experiencia, siendo este aspecto vital en este tipo de metodologías para obtener buenos resultados. Estos equipos son preferiblemente pequeños y en proyectos que lo fundamental sea el producto final y la rapidez con que se concluya, sin que la documentación sea primordial.

El sistema a desarrollar es a largo plazo y complejo por lo que requiere una amplia documentación para un mejor entendimiento en su implementación y después de concluido el proceso. Por las descripciones antes mencionadas es que se llega a la conclusión de que para el desarrollo del mismo se requiere de una metodología tradicional o robusta.

Entre las metodologías de desarrollo de software más importantes por su calidad y aplicación en el mundo se encuentran la Rational Unified Process (RUP) o Proceso Unificado que se enmarca en el grupo de las tradicionales o robustas, aunque existen variaciones de la misma para adaptarse a procesos ágiles. También se tiene la Metodología Extreme Programing (XP), exponente de las metodologías ágiles más usadas en estos momentos.

2.2.1 Extreme Programing (XP).

XP es una metodología de desarrollo de proyectos a corto plazo centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo.

Es la más exitosa en la actualidad utilizada para proyectos en los que se hace fundamental su desarrollo en un corto tiempo y con un reducido equipo. La misma consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

Las historias de usuarios son un elemento de gran importancia en XP pues es la técnica que utiliza para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

XP consta de seis fases: Exploración, Planificación de la Entrega, Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

Entre las prácticas que propone fundamentalmente se encuentran:

El juego de la planificación: El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.

Metáfora: El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema.

Entregas pequeñas: Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad pretendida para la aplicación.

Diseño simple: Propone el diseño de la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.

Pruebas: La producción de código está dirigida por las pruebas unitarias. Las pruebas unitarias son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del sistema. Los clientes escriben las pruebas funcionales para cada historia de usuario que deba validarse.

Refactorización: Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios.

Programación en parejas: Toda la producción de código debe realizarse con trabajo en parejas de programadores.

Propiedad colectiva del código: Cualquier programador puede cambiar cualquier parte del código en cualquier momento.

Integración continua: Cada pieza de código es integrada en el sistema una vez que esté lista.

2.2.2 Metodología Rational Unified Process (RUP).

RUP es un proceso que define claramente quién, cómo, cuándo y qué debe hacerse. Su enfoque está basado en modelos y para este fin utiliza un lenguaje bien definido que es UML. Éste aporta herramientas como los casos de uso, que definen los requerimientos y permite la ejecución iterativa del proyecto y del control de riesgos.

Esta metodología de desarrollo de software define tres elementos fundamentales que conforman la guía para el desarrollo de la arquitectura como parte de la solución.

- Guiado por los Casos de Uso
- Centrado en la Arquitectura
- Iterativo e Incremental

Que el desarrollo de la solución sea guiado por los casos de uso permitirá la configuración del sistema en cuanto a módulos y subsistema principalmente teniendo en cuenta la prioridad y la complejidad de los distintos casos de usos, y la seguridad de que el producto final satisfaga los requerimientos reales del cliente. Esto se logra ya que los casos de uso representan los requisitos funcionales de la aplicación, guían el diseño, la implementación y prueba, es decir, guían el proceso de desarrollo**.** La arquitectura debe posibilitar el desarrollo de todos los casos de usos y estos a su vez tienen que ajustarse a la arquitectura, por lo que la arquitectura y los casos de usos deben desarrollarse de forma paralela**.** Es iterativo e incremental pues propone dividir el desarrollo del productos en varias iteraciones, en cada una de ellas se obtiene una series de artefactos que servirá como base de la próxima. Así, en cada iteración se incrementa o modifican los artefactos, posibilitando entre otras cosas la identificación de riesgos, la corrección de errores, mejoras de las funcionalidades y refinar en sentido general el producto, hasta obtener el resultado esperado. Cada una de las fases definidas por la metodología de desarrollo termina con la consecución de un conjunto de hitos, la terminación de cada uno de ellos no se pretende que se realicen de una vez sino que se vayan alcanzando a medida que se vayan refinando, iterando una y otra vez de acuerdo con el plan de iteraciones definidos en el plan de desarrollo del sistema.

El ciclo de vida del RUP cuenta de cuatro fases: inicio, elaboración, construcción y transición, las que se subdividen en iteraciones y nueve flujos de trabajos, seis flujos de ingeniería y tres de apoyo.

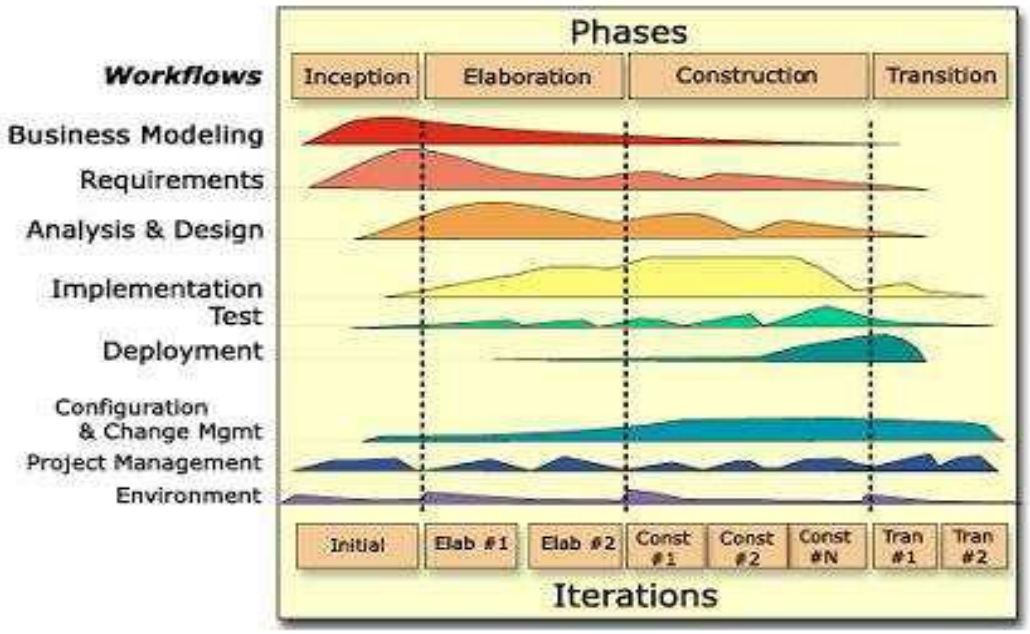

Figura 4: Flujos de Trabajo, Fases y Esfuerzo por Iteración.

Descripción de las fases de RUP:

Fase	Objetivos	Puntos de Control
Inicio	• Definir el alcance del proyecto • Entender que se va a construir	Objetivo del proyecto.
Elaboración	• Construir una versión ejecutable de la arquitectura de la aplicación. • Entender cómo se va a construir.	Arquitectura de la Aplicación.
Construcción	• Completar el esqueleto de la aplicación con su funcionalidad. • Construir una versión beta.	Versión operativa inicial de la aplicación
Transición	• Hacer disponible la aplicación para los usuarios finales. • Construir la versión final.	Liberación de la versión de la aplicación.

Tabla 1: Descripción de las fases de RUP.

Flujos de Trabajo:

- Modelamiento del Negocio
- Requerimientos
- Análisis y Diseño
- Implementación
- Prueba (Testeo)
- Despliegue

Flujos de apoyo:

- Administración del proyecto
- Administración de configuración y cambios
- Ambiente

Después de este análisis la metodología seleccionada es RUP pues sus características son factibles para la realización del sistema. Además porque es una propuesta de proceso para el desarrollo de software orientado a objetos que utiliza Unified Model Language (UML) como único lenguaje para describir el proceso. Está basado en componentes, lo cual quiere decir que el sistema de software en construcción está formado por componentes interconectados a través de interfaces bien definidas. Otra razón fundamental es porque el sistema se va desarrollando y documentando al mismo tiempo, que si algún miembro del equipo no puede seguir con el trabajo, el que ocupe su lugar tiene por donde guiarse y conoce que fue lo que se hizo.

2.3 Lenguaje Unificado de Modelado (UML).

Lenguaje Unificado de Modelado (UML), por sus siglas en inglés, es un lenguaje para visualizar, especificar, construir, y documentar los artefactos que se crean durante el proceso de desarrollo. Permite la modelación de sistemas con tecnología orientada a objetos. No es una guía para realizar el análisis y diseño orientado a objetos, es decir, no es un proceso. UML no es método, ni una metodología. Ofrece un estándar para describir un plano del sistema (modelo), incluyendo aspectos conceptuales tales como

procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Una fundamental ventaja es que es un lenguaje gráfico con sintaxis y semántica bien definidas. La sintaxis de la notación gráfica se especifica mediante su correspondencia con los elementos del modelo semántico subyacente, cuya semántica se define por medio de textos descriptivos y restricciones.

UML se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como RUP), aunque no especifica en sí mismo qué metodología o proceso usar.

Según diversos autores UML no es, en sí, un lenguaje de descripción arquitectónica pues su forma de expresar ciertas características, sobre todo dinámicas de las estructuras no es suficiente para los arquitectos.

2.4 Herramienta CASE.

Las herramientas de Ingeniería de Software Asistida por Ordenador(CASE) por sus siglas en ingles, son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, calculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

2.4.1 Rational Rose Enterprise.

Rational Rose es la herramienta CASE que cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases y entregables. El navegador UML de Rational Rose permite establecer una trazabilidad real entre el modelo (análisis y diseño) y el código ejecutable. Facilita el desarrollo de un proceso cooperativo en el que todos los agentes tienen sus propias vistas de información (vista de casos de uso, lógica, de componentes y de despliegue), pero utilizan un lenguaje común para comprender y comunicar la estructura y la funcionalidad del sistema en construcción.

Rational Rose no se integra con varios IDE como Visual Paradigm, solo lo hace con Borland JBuilder de la versión 7.0 en lo adelante y Microsoft Visual Studio de la versión 2003 en adelante. Además no es multiplataforma y es aconsejable utilizar Windows 2000, Windows NT y Windows XP.

2.4.2 Visual Paradigm.

Visual Paradigm es un herramienta CASE muy usada en la actualidad para desarrollar proyectos importantes y de alta complejidad.

Entre sus principales características se destacan:

- Es robusto y portable.
- Genera código y realiza ingeniería inversa para diez lenguajes de programación.
- Se integra con el Visio para importar imágenes del mismo para realizar los diagramas de despliegue.
- Además exporta e importa los diagramas con estándar XML y como imágenes.
- Es multiplataforma y gratis en su edición Community.

Esta herramienta es colaborativa, es decir, soporta múltiples usuarios trabajando sobre el mismo proyecto, genera la documentación del proyecto automáticamente en varios formatos como Web o PDF (Portable Document Format), además de permitir el control de versiones.

Para el desarrollo de este sistema se seleccionó la herramienta Visual Paradigm ya que es una poderosa herramienta CASE que al igual que el Rational Rose utiliza UML para el modelado. Es la herramienta por excelencia para ser utilizada en un ambiente de software libre. Permite crear tipos diferentes de diagramas en un ambiente totalmente visual. Es muy sencillo de usar, fácil de instalar y actualizar. Genera código para varios lenguajes. Tiene integrado el MS Visio y es compatible con otras ediciones. Posibilita la representación gráfica de los diagramas permitiendo ver el sistema desde diferentes perspectivas, como el de componentes, despliegue, secuencia, casos de uso, clase, actividad, estado, entre otros.

2.5 Lenguajes de Programación del Lado del Servidor.

Los lenguajes de programación del lado del servidor son aquellos que se ejecutan en el servidor web para la construcción de las páginas web antes de ser enviadas al navegador. Los códigos puestos en el servidor construyen las páginas que serán enviadas de respuesta al cliente, pueden acceder a bases de datos, recursos en la red y ficheros en el servidor. Estos lenguajes son imperceptibles por el cliente, cuando se realiza la petición al servidor este convierte todo el código a que el navegador lo interprete. Entre los principales lenguajes y más usados del lado del servidor se encuentra PHP.

2.5.1 PHP.

Hypertext Pre-Processor (PHP), por sus siglas en ingles, fue diseñado originalmente para la creación de páginas dinámicas. Es un lenguaje script del lado del servidor.

Entre sus principales características se destaca que es libre, por lo que es de fácil acceso para todos. Es multiplataforma ya que es soportado por Windows y Linux, además que cuenta con muchas funcionalidades. Tiene comportamiento modular, brinda la posibilidad de adicionar nuevas funcionalidades a través de nuevos módulos. Permite conectarse a varios sistemas de bases de datos brindando múltiples funcionalidades, destacando en este aspecto la conectividad con PostgreSQL y MySQL que son gestores muy utilizados en el desarrollo de aplicaciones web, destacando que los desarrolladores están preparados en este lenguaje. Cuenta con amplia documentación en su página oficial la cual incluye descripciones y ejemplos de cada una de sus funciones.

Permite el empleo de Programación Orientada a Objeto, y de herencia en cierto modo, no requiere que se le especifique el tipo de datos de las variables y maneja excepciones a partir de la versión 5.0. Por todas estas razones antes expuestas entonces se decide que PHP será el lenguaje a usar para la programación por el lado del servidor del sistema a desarrollar.

2.6 Frameworks.

Un framework es un conjunto de clases que cooperan y forman un diseño reutilizable para un tipo específico de software. Ofrece una guía arquitectónica partiendo el diseño en clases abstractas y definiendo sus responsabilidades y sus colaboraciones. Un desarrollador personaliza el framework para una aplicación particular mediante herencia y composición de instancias de las clases.

Para el desarrollo de aplicaciones web existe una serie de frameworks con una estructura de clases bien definida incluyendo las de interacción con los sistemas gestores de bases de datos y las clases

especificas para el desarrollo de la lógica de la aplicación. Para ayudar a los desarrolladores incluyen soportes a distintos lenguajes script, librerías, paquetes de clases que automatizan la interacción con base de datos y simplifica la creación de funciones complejas. Esto permite dedicar más tiempo a desarrollar la lógica de la aplicación sin necesidad de comenzar por los pasos básicos nativos a cada programa para asegurar las funcionalidades.

Entre los inconvenientes para el uso de los frameworks se encuentra el tiempo que requiere la preparación para lograr entender el funcionamiento y la estructura del mismo, restando tiempo para el diseño y desarrollo del producto. Añadido a esto, en ocasiones incluye código innecesario, principalmente para aplicaciones pequeñas. Por otro lado es factible el empleo de framework como base para el desarrollo de sistemas complejos en cuanto a funcionalidades e interacción con bases de datos.

Después de una amplia investigación se realizo un análisis de varios frameworks como son: CakePHp, Cumbia y Symfony, teniendo en cuenta varios criterios como por ejemplo: utilización de patrón MVC, compatibilidad con diferentes bases de datos, uso de PHP 5, uso de plantillas, curva de aprendizaje suave, buena documentación, una comunidad activa y facilidad de instalación; además de la experiencia y tendencia que existe en la UCI hacia este framework es entonces que se selecciona Symfony.

2.6.1 Symfony.

Symfony es un completo framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. 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.

Symfony ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Symfony es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft, además está desarrollado completamente con PHP 5. Se puede ejecutar tanto en plataformas Linux como en plataformas Windows.

Características de Symfony

- Fácil de instalar y configurar en la mayoría de plataformas (y con la garantía de que funciona correctamente en los sistemas Windows y Linux).
- Independiente del sistema gestor de bases de datos.
- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos.
- Basado en la premisa de "convenir en vez de configurar", en la que el desarrollador solo debe configurar aquello que no es convencional.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.
- Preparado para aplicaciones empresariales y adaptables a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Es posible realizar cambios "en caliente" de la configuración sin necesidad de reiniciar el servidor.
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros.

Este potente frameworks automatiza la mayoría de los elementos comunes de proyectos web, como por ejemplo:

- Los formularios incluyen validación automatizada y relleno automático de datos, lo que asegura la obtención de datos correctos y mejora la experiencia de usuario.
- Los datos incluyen mecanismos de escape que permiten una mejor protección contra los ataques producidos por datos corruptos.
- La capa de internacionalización que incluye Symfony permite la traducción de los datos y de la interfaz, así como la adaptación local de los contenidos.
- La capa de presentación utiliza plantillas y layouts que pueden ser creados por diseñadores HTML sin ningún tipo de conocimiento del framework. Los helpers incluidos permiten minimizar el código utilizado en la presentación, ya que encapsulan grandes bloques de código en llamadas simples a funciones.
- El sistema de enrutamiento y las URL limpias permiten considerar a las direcciones de las páginas como parte de la interfaz, además de estar optimizadas para los buscadores.
- La gestión de la caché reduce el ancho de banda utilizado y la carga del servidor.

- La autenticación y la gestión de credenciales simplifican la creación de secciones restringidas y la gestión de la seguridad de usuario.
- Los listados son más fáciles de utilizar debido a la paginación automatizada, el filtrado y la ordenación de datos.
- Las interacciones con Ajax son muy fáciles de implementar mediante los helpers que permiten encapsular los efectos JavaScript compatibles con todos los navegadores en una única línea de código.

2.7 Sistema Gestor de Base de Datos.

Un Sistema Gestor de Base de Datos (SGBD) es un conjunto de programas que permiten crear y mantener actualizada una base de datos, asegurando así su integridad, confidencialidad y seguridad. Presenta una interfaz, mediante la cual el usuario puede comunicarse con el sistema físico y realizar operaciones como almacenar o recuperar datos de ella. Las principales funciones que debe cumplir un SGBD son la creación y mantenimiento de la base de datos, el control de accesos, la manipulación de datos de acuerdo con las necesidades del usuario, el cumplimiento de las normas de tratamiento de datos, evitar redundancias e inconsistencias.

Características a las que se deben llegar con un SGBD:

- Control de la redundancia: La redundancia de datos tiene varios efectos negativos (duplicar el trabajo al actualizar, desperdicia espacio en disco, puede provocar inconsistencia de datos) aunque a veces es deseable por cuestiones de rendimiento.
- Restricción de los accesos no autorizados: cada usuario ha de tener unos permisos de acceso y autorización.
- Cumplimiento de las restricciones de integridad: el SGBD ha de ofrecer recursos para definir y garantizar el cumplimiento de las restricciones de integridad.

Entre los Sistemas Gestores de Base de Datos más usados están Oracle, MySQL, SQL Server de Microsoft, PostgreSQL entre otros.

2.7.1 Oracle.

Oracle es considera uno de los sistemas de bases de datos más completos. Es un sistema gestor de base de datos robusto que por sus características garantiza la seguridad e integridad de los datos. Permite que las transacciones se ejecuten de forma correcta, sin causar inconsistencias. Ayuda a administrar y almacenar grandes volúmenes de datos y presenta estabilidad, escalabilidad además de ser multiplataforma.

La tecnología Oracle se encuentra prácticamente en todas las industrias alrededor del mundo. Garantiza el funcionamiento de sus bases de datos, que en caso de caídas del servidor compensa económicamente con cifras cercanas a las 7 cifras.

A pesar de ser un sistema tan potente, no cumple con los requisitos necesarios para su utilización en este sistema a desarrollar, pues requiere de una licencia para poderlo utilizar, es decir, es necesario pagar para poder utilizarlo.

2.7.2 MySQL.

MySQL es un sistema de gestión de base de datos muy potente y popular. Usado tanto por empresas pequeñas como de gran envergadura. Es relacional, multihilo, multiusuario y multiplataforma caracterizándose por:

- Aprovechar la potencia de sistemas multiprocesador, gracias a su implementación multihilo.
- Soportar gran cantidad de tipos de datos para las columnas.
- Disponer de API's en gran cantidad de lenguajes (C, Java, ODBC, .NET, PHP, Perl, Python, etc.).
- Disponibles en múltiples plataformas como Linux, Windows, y Mac, ambos de 32 y 64 bit.
- Soporta hasta 32 índices por tabla.

Tiene como una de sus principales ventajas la velocidad en la lectura de datos, pero a costa de eliminar un conjunto de facilidades que presentan otros SGBD: integridad referencial, bloqueo de registros, procedimientos almacenados. Además de ser un software propietario por lo que para su uso se necesita pagar a sus proveedores.

2.7.3 PostgreSQL.

PostgreSQL es un sistema gestor de base de datos de objetos relacionales basado en software libre. Ofrece una alternativa ante otros sistemas gestores de bases de datos comerciales. PostgreSQL no es controlado por ninguna compañía, pero responde al esfuerzo de una comunidad global de desarrolladores. Ofrece una potencia adicional sustancial al incorporar los siguientes cuatro conceptos adicionales básicos en una vía en la que los usuarios pueden extender fácilmente el sistema: clases, herencia, tipos y funciones. Además aportan potencia y flexibilidad adicional como son las restricciones (constraints), disparadores (triggers), reglas (rules) e integridad transaccional.

Características principales de PostgreSQL:

- Base de Datos Objeto relacional: Aproxima los datos a un modelo objeto-relacional, y es capaz de manejar complejas rutinas y reglas.
- Alta concurrencia: PostgreSQL permite acceso de lectura y escritura en una tabla de forma concurrente sin necesidad de bloqueos. Siempre se obtiene la última versión de datos que se ha actualizado usando MVCC (Control de Versiones Concurrente).
- Integridad Referencial para garantizar la validez de los datos de la Base de Datos.
- Write Ahead Logging (WAL): Consiste en registrar los cambios antes de que estos sean escritos en la base de datos. Esto garantiza que en caso de que la BD se caiga, existirá un registro de las transacciones a partir del cual se puede restaurar la BD desde el punto en que se quedó.
- Gran variedad de tipos de datos nativos: fecha, monetarios, elementos gráficos, MAC, IP, cadenas de bits, figuras geométricas, etc.
- Soporte SQL Comprensivo: Soporta la especificación SQL99 e incluye características avanzadas tales como las uniones (joins) SQL92.

De acuerdo a lo antes planteado se decidió usar como SGBD el PostgreSQL caracterizado como un motor de bases de datos avanzado y de código abierto. Además de las características antes mencionadas es importante señalar que este sistema cuanto mayor es el número de registro y más compleja es la consulta sus resultados son más rápido. Debido a estas características es seleccionado PostgreSQL como gestor de bases de datos donde su principal cualidad es que es código abierto, además de permitir soporte con la mayoría de los lenguajes de programación y se considera como una de los SGBD más potentes.

2.8 Conclusiones del Capítulo.

Después de un previo estudio realizado se tiene como resultado en este capítulo las herramientas principales de desarrollo y modelado para darle solución del sistema a realizar. Teniendo en cuenta como principales criterios el de ser herramientas libres y que cumplen con las condiciones necesarias que permita el desarrollo de la aplicación.

Capitulo 3: Propuesta de la Arquitectura del Sistema.

3.1 Introducción.

En este capítulo se tiene como principal objetivo obtener una mejor visión del sistema, posibilitando esto su desarrollo de forma eficiente. Serán descritas las metas y restricciones que influyen de manera significativa en la arquitectura, además se obtendrán artefactos definidos por la metodología RUP como son las vistas arquitectónicas.

3.2 Estructura del Equipo de Desarrollo.

El equipo de trabajo está distribuido de la siguiente forma:

- Líder de proyecto
- Analista-Diseñador
- Arquitecto de software
- Diseñador de base de datos
- Jefe de Grupo de Desarrollo
- Implementador
- Grupo de Calidad

3.2.1 Configuración de los puestos de trabajos por roles.

- **Analista**
    1. PC, con mouse y teclado.
    2. Instalación de Visual Paradigm.
    3. Instalación del paquete Office.

- **Implementador**
    1. PC, con mouse y teclado.
    2. Instalación del IDE NetBeans 6.8 con el frameworks Symfony.
    3. Servidor Apache.

- **Diseñador de BD**
    1. PC, con mouse y teclado.
    2. Instalación del Visual Paradigm.
    3. Instalación de PostgreSQL.

- **Arquitecto de software**
    1. PC, con mouse y teclado.
    2. Instalación del Visual Paradigm.
    3. Instalación del paquete Office.

- **Líder del Proyecto**
    1. PC, con mouse y teclado.
    2. Instalación de Visual Paradigm.
    3. Instalación del paquete Office.

3.3 Subsistemas del Sistema.

- Subsistema de Administración.
- Subsistema de Media.
- Subsistema de Programación.
- Subsistema Producción.
- Subsistema Equipamiento.

3.4 Requerimientos no funcionales del sistema.

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Son las características que hacen al producto atractivo, usable, rápido o confiable. Los requerimientos no funcionales influyen de manera significativa en la arquitectura. El SGPDTU debe cumplir con los siguientes requerimientos:

3.4.1 Requerimientos de Software.

Las PCs clientes deben tener instalado:

- Windows XP Profesional ó GNU/Linux.
- Navegador Internet Explorer (IE), Mozilla

La PC servidor debe tener instalado:

- Un servidor Apache con la versión 5 de PHP
- Symfony como framework
- PostgreeSQL 8.2 como SGBD

3.4.2 Requerimientos de Hardware.

Las PC Clientes deberán tener las siguientes características:

- 64 MB de Memoria tipo RAM como mínimo.
- Debe contar con un microprocesador de 1 GHz de velocidad.
- Requerimientos mínimos sobre los cuales pueda correr un sistema operativo con soporte para navegadores Web.

La PC Servidor deberá tener las siguientes características:

- 1 Gb de Memoria tipo RAM como mínimo.
- Microprocesador a 2.0 GHz de velocidad de procesamiento.
- HDD 80 Gb.

3.4.3 Restricciones en el diseño y la implementación.

- Diseño e implementación de una arquitectura flexible, que permita la fácil integración o desintegración de componentes.
- El patrón arquitectónico que se debe emplear en el desarrollo es el modelo-vista-controlador.
- La arquitectura debe soportar migrar la interfaz de usuario de forma rápida.
- El lenguaje de programación que se debe utilizar es PHP con el uso de frameworks.
- Los protocolos para la comunicación entre el cliente y el servidor web que se deben usar es HTTPS.

3.4.4 Apariencia o Interfaz Externa.

- Las interfaces de la aplicación tendrán los componentes visuales necesarios para las operaciones correspondientes, evitando la sobrecarga de imágenes.
- Interfaz amigable, interactiva, permitiendo al usuario navegar con facilidad e intuición por la aplicación.
- El sistema debe mostrar un ambiente profesional, sin información repetida o en exceso.

3.4.5 Requerimientos de confidencialidad.

- Permitir autenticación segura.
- La seguridad se establecerá por roles que se le asignarán a los usuarios que interactúen con el sistema.

3.4.6 Requerimientos de disponibilidad.

Los usuarios autorizados tendrán acceso a la información en todo momento, se debe lograr balancear la carga de acceso entre múltiples servidores, disminuyendo los tiempos de respuesta.

3.4.7 Requerimientos de Seguridad.

- Se deberá utilizar usuarios de base de datos con roles bien definidos para cada una de las operaciones del sistema.
- Debe quedar constancia de quién, desde donde, y cuando se realizó una operación determinada en el sistema
- La asignación de usuarios y sus opciones sobre el sistema se garantizan desde el subsistema de Administración.
- Los mecanismos de seguridad no deben impedir que los usuarios autorizados accedan a la información, la misma debe estar disponible en todo momento.
- La base de datos deberá estar disponible las 24 horas, pudiendo realizar operaciones sobre la misma en cualquier momento que se necesite. En caso de una falla en el servidor (interrupción del servicio eléctrico, desconexión de la red) se deberá contar con un respaldo para cada subsistema que le permita seguir con sus actividades.

3.4.8 Rendimiento.

- La BD deberá resistir un gran número de operaciones simultáneas sobre la misma.
- El sistema debe responder a las peticiones del usuario en un tiempo relativamente rápido (menos de 5 segundos).

3.4.9 Eficiencia.

Teniendo en cuenta la cantidad de clientes que existen en la DTU, el sistema podrá alojar entre 70 y 80 transacciones. El tiempo de respuesta por transacción estimado es de 0.2 a 0.5 segundos. El sistema no consume gran cantidad de recursos, pues este no necesita de disco y agota muy poca memoria.

3.5 Vistas de la Arquitectura de Software.

A veces la arquitectura del software tiene secuelas de un diseño del sistema que fue muy lejos en particionar prematuramente el software, o de un énfasis excesivo de algunos de los aspectos del desarrollo del software: ingeniería de los datos, eficiencia en tiempo de ejecución, o estrategias de desarrollo y organización de equipos. A menudo la arquitectura tampoco aborda los intereses de todos sus "clientes".

Para remediar este problema fue entonces desarrollado el modelo 4+1 que describe la AS usando cinco vistas concurrentes, donde cada vista se refiere a un conjunto de intereses de los diferentes stakeholders del sistema permitiendo así una visión más detallada de la AS del sistema. Este modelo consta con las siguientes vistas:

- **La vista lógica** describe el modelo de objetos del diseño cuando se usa un método de diseño orientado a objetos. Para diseñar una aplicación muy orientada a los datos, se puede usar un enfoque alternativo para desarrollar algún otro tipo de vista lógica, tal como diagramas de entidad-relación.
- **La vista de procesos** describe los aspectos de concurrencia y sincronización del diseño.
- **La vista de implementación** describe la organización estática del software en su ambiente de desarrollo.
- **La vista despliegue** describe el mapeo del software en el hardware y refleja los aspectos de distribución.

- **Vista de casos de uso escenarios o vista de escenario** que consiste en una selección de casos de uso o de escenarios que los arquitectos pueden elaborar a partir de las cuatro vistas anteriores.

Figura 5: Vistas de la AS.

3.5.1 Vista de Caso de Uso.

A partir de la vista de Casos de Uso, se puede definir los escenarios o los casos de uso que serán de interés para cada iteración del ciclo de desarrollo. Esta describe los escenarios o casos de uso que tienen significación y que encapsulan la funcionalidad central del sistema.

Los casos de uso arquitectónicamente significativos, son aquellos que describen funcionalidades imprescindibles para el sistema, y que a través de estos se valida la arquitectura propuesta para el mismo.

A continuación se muestra la cantidad de casos de uso arquitectónicamente significativos por cada subsistema:

Subsistema	Cantidad de Casos de Uso
Administración	5
Producción	1
Programación	1
Equipamiento	1

Tabla 2: Casos de Uso Arquitectónicamente Significativos

Casos de uso arquitectónicamente significativos:

- Autenticar usuario.
- Gestionar Solicitud de Medias
- Gestionar Usuario.
- Gestionar Rol.
- Gestionar Plan de Producción.
- Gestionar Medias.
- Gestionar Plan de Programación.
- Gestionar Reportes.
- Asignar Equipamiento.

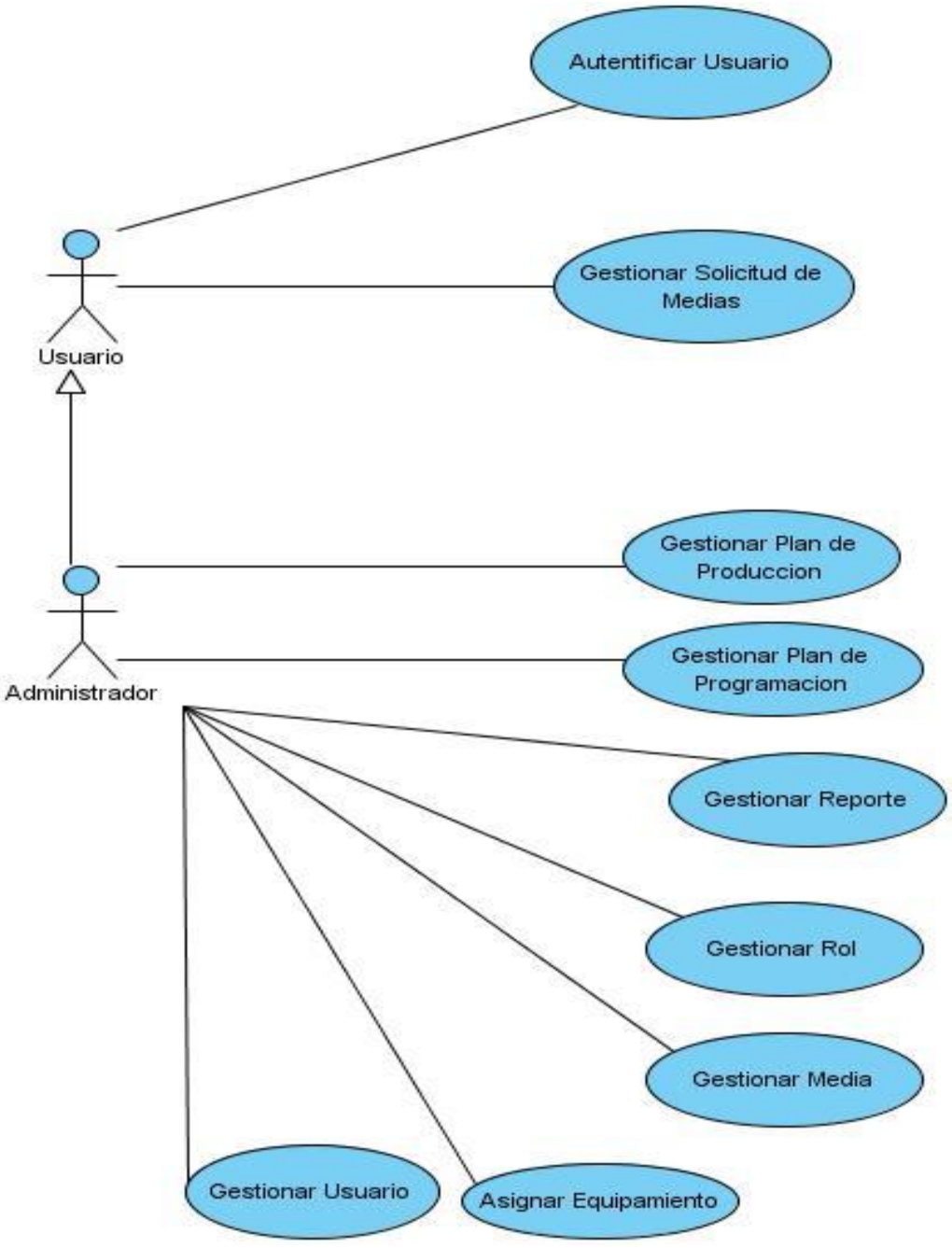

Figura 6: Vista de Casos de Uso del Sistema.

3.5.2 Vista Lógica.

En la descripción de la arquitectura, la vista lógica describe las clases más importantes, su organización en paquetes y subsistemas, y la organización de estos paquetes y subsistemas en capas. Apoya principalmente los requisitos funcionales del sistema, el cual se descompone en una serie de abstracciones clave, tomadas principalmente del dominio del problema en la forma de objetos o clases de objetos, aplicándose aquí los principios de abstracción, encapsulamiento y herencia. Esta descomposición no solo se hace para potenciar el análisis funcional, sino también sirve para identificar elementos y mecanismos del diseño comunes en diversas partes del sistema.

La siguiente imagen muestra la distribución de los paquetes más significativos dentro de cada una de las partes definidas para la aplicación y la dependencia entre ellos.

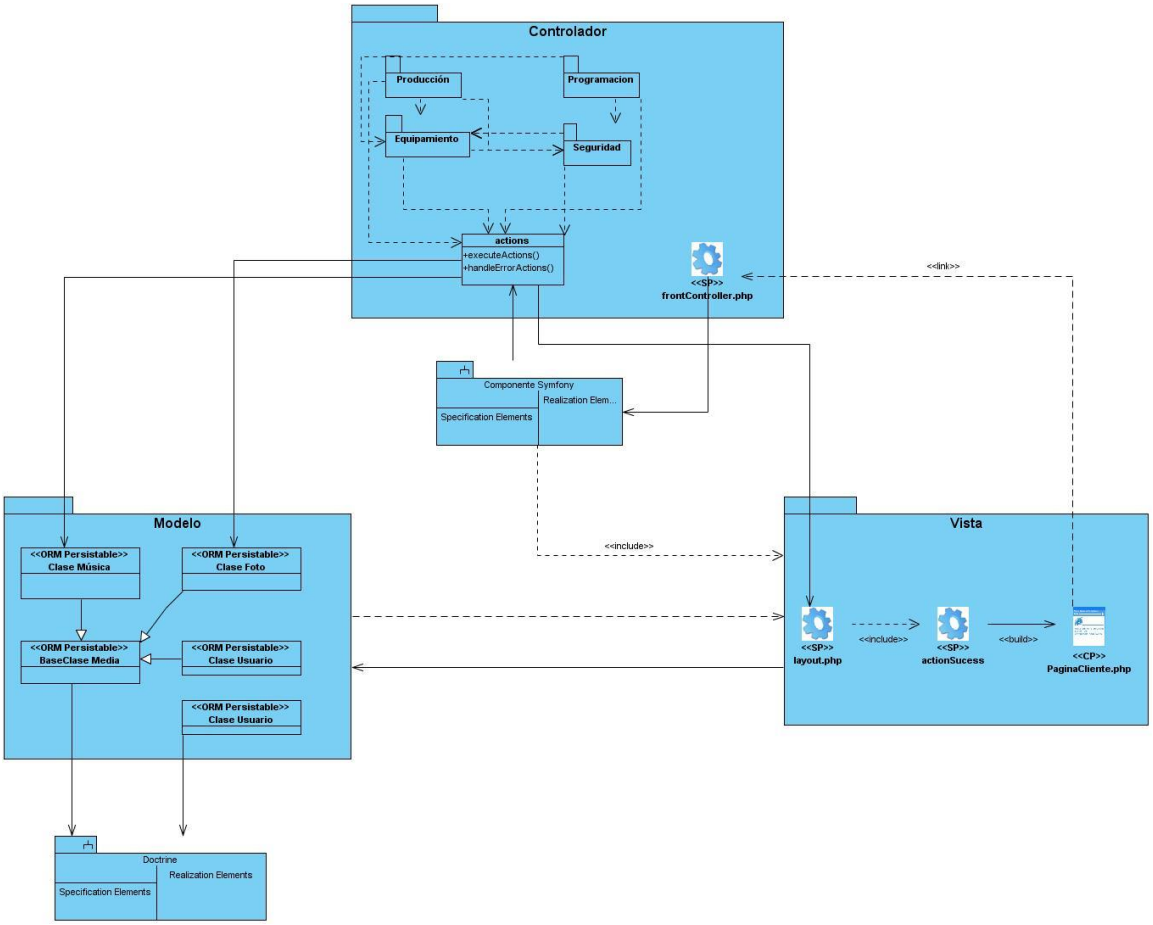

Figura 7: Vista Lógica.

3.5.3 Vista de procesos.

La arquitectura de procesos toma en cuenta algunos requisitos no funcionales tales como la performance y la disponibilidad. Se enfoca en asuntos de concurrencia y distribución, integridad del sistema, de tolerancia a fallas. La vista de procesos también específica en cuál hilo de control se ejecuta efectivamente una operación de una clase identificada en la vista lógica.

La arquitectura de procesos se describe en varios niveles de abstracción, donde cada nivel se refiere a distintos intereses. El nivel más alto la arquitectura de procesos puede verse como un conjunto de redes lógicas de programas comunicantes (llamados procesos) ejecutándose en forma independiente.

Esta vista suministra una base para la comprensión de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos concurrentes o hilos. Nuestro proyecto no cuenta con hilo ni procesos concurrentes.

3.5.4 Vista de Implementación.

La vista de desarrollo se centra en la organización real de los módulos de software en el ambiente de desarrollo del software. El software se empaqueta en pequeñas partes (subsistemas) que pueden ser desarrollados por uno o un grupo pequeño de desarrolladores. Los subsistemas se organizan en una jerarquía de capas, cada una de las cuales brinda una interfaz estrecha y bien definida hacia las capas superiores.

La vista de implementación proporciona una descripción de las principales capas y subsistemas de componentes de la aplicación. Los paquetes principales de la aplicación por cada uno de los módulos son:

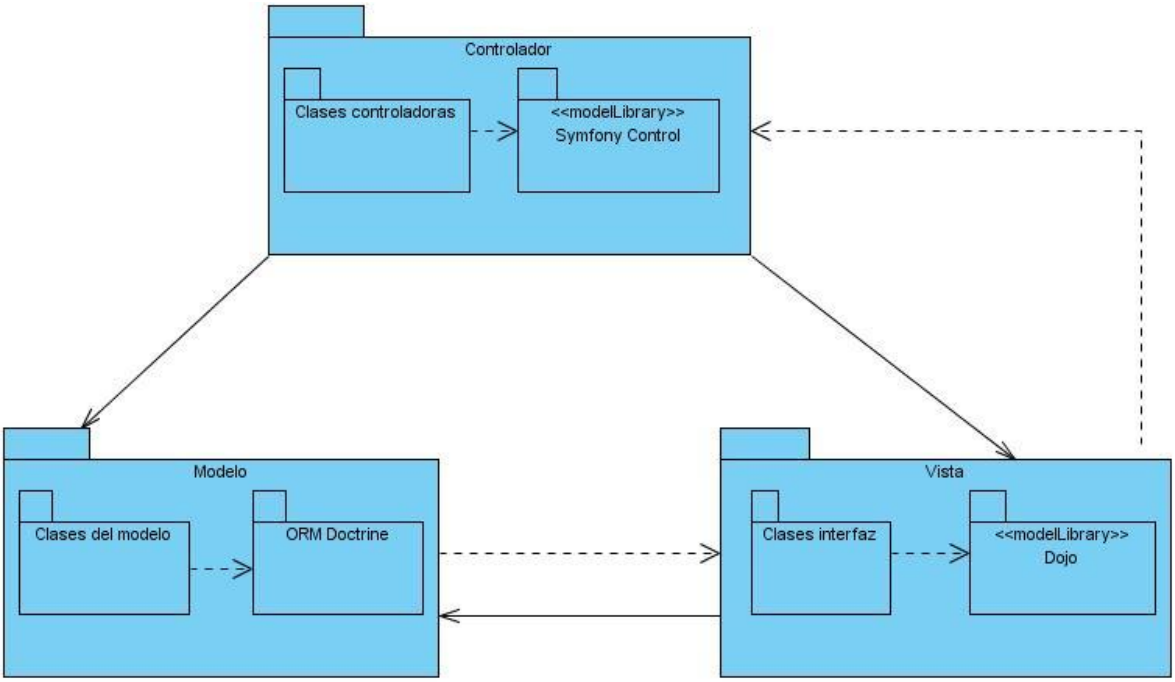

Figura 8: Vista de Implementación.

3.5.5 Vista de Despliegue.

La vista de despliegue o física toma en cuenta primeramente los requisitos no funcionales del sistema tales como la disponibilidad, confiabilidad (tolerancia a fallas), y escalabilidad. Muestra las relaciones físicas entre los componentes hardware y software en el sistema. Es un conjunto de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes software, objetos y procesos. La distribución física de los principales componentes del sistema puede ser adaptada a múltiples configuraciones dependiendo del presupuesto disponible para el mismo.

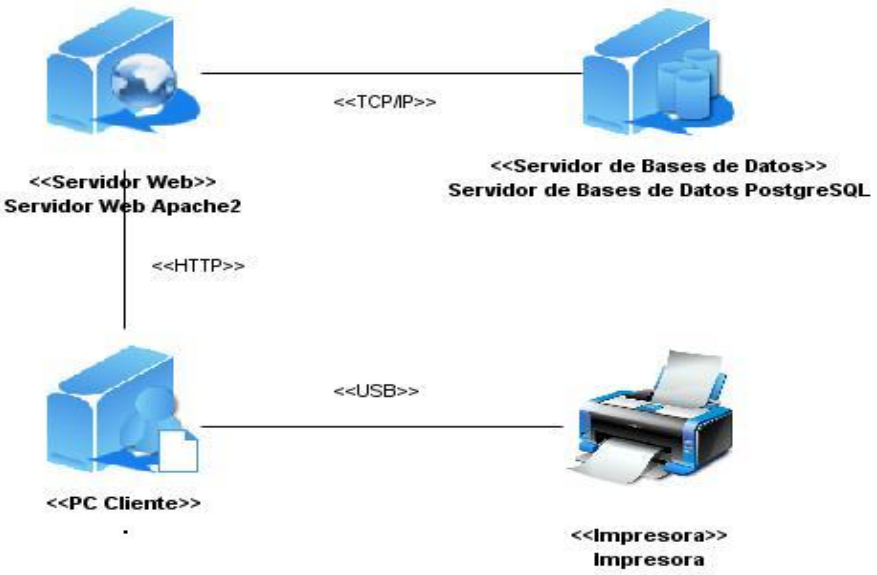

Figura 9: Diagrama de Despliegue.

3.6 Conclusiones del Capítulo.

En el presente capítulo se realizó la propuesta de la arquitectura del proyecto Sistema de Gestión de Procesos para la Dirección de Televisión Universitaria. Con esto quedaron plasmados los requerimientos no funcionales, que influyen de manera significativa en la Arquitectura de Software. Se definieron las vistas arquitectónicas (Vista de casos de uso, Vista Lógica, Vista de procesos, Vista de despliegue y Vista de Implementación) vitales para el desarrollo de la sistema y la estructura del equipo de desarrollo.

CAPÍTULO 4: Evaluación de la Arquitectura del Sistema.

4.1 Introducción.

Uno de los factores que determina el éxito o el fracaso del un sistema es su arquitectura, es decir, la estructura del sistema. Si la arquitectura ha sido bien diseñada, entonces garantiza que el sistema cumpla con uno o varios atributos de calidad.

La evaluación de la arquitectura de software es fundamental para medir el grado de calidad de la misma. Durante esta evaluación se puede conocer los riesgos asociados con el desarrollo del sistema, permitiendo así conocer los problemas y debilidades para posibilitar una corrección a tiempo de estos. Así cuanto más temprano se encuentren los problemas, pues menor serán los daños que puedan ocasionar.

4.2 ¿Por qué evaluar una Arquitectura?

El propósito de realizar evaluaciones a la arquitectura, es para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante, verificar que los requerimientos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad. Cabe señalar que los requerimientos no funcionales también son llamados atributos de calidad (8).

Es importante evaluar una arquitectura por las siguientes razones:

- Cuanto más temprano se encuentre un problema en un proyecto de software, mejor.
- Realizar una evaluación de la arquitectura es la manera más económica de evitar desastres.
- Fija la estructura organizacional, tanto del desarrollo, construcción y ejecución del sistema.
- Logra los atributos de calidad.
- Permite estimaciones más certeras. (9)

4.3 ¿Cuándo evaluar una Arquitectura?

Un aspecto fundamental para realizar la evaluación de la arquitectura de software es el momento que se realiza. Existen dos momentos para realizar la misma, la evaluación temprana y la evaluación tardía.

Evaluación Temprana

Para realizar este tipo de evaluación no es necesario que la arquitectura se encuentre completamente realizada. Permite efectuar decisiones sobre la arquitectura en cualquier nivel, puesto que se pueden atribuir cambios arquitectónicos producto de una evaluación en función de los atributos de calidad esperados.

Evaluación Tardía

Para realizar este tipo de evaluación es necesario que la arquitectura del sistema se encuentre establecida y su implementación esté concluida, es decir, que el sistema ya esté terminado. Se considera que la evaluación en este punto es muy importante y útil, puesto que puede observarse el cumplimiento de los atributos de calidad asociados al sistema y su comportamiento general. La realización de la evolución de la AS debe realizarse cuando estén listos los elementos necesarios para justificarla. Puede ser determinada en el momento que el equipo de desarrollo necesite tomar decisiones que dependen de la arquitectura propuesta y que el costo de no tenerlas en cuenta daría al trasto con un costo superior al de llevar a cabo una evaluación. Generalmente se responsabiliza de la evaluación de la AS a los miembros del equipo de desarrollo (Arquitecto, Analista, Diseñador, etc.) pero no son los únicos que pueden realizar esta tarea. Se pueden contratar especialistas en el tema para que realicen las pruebas o los clientes involucrados en el negocio (stakeholder) los que en dependencia de los resultados pueden decidir si es factible para ellos continuar el proyecto o no.

4.4 Cualidades a evaluar en la Arquitectura de Software.

Las cualidades o atributos de calidad son requerimientos del sistema que hacen referencia a características que éste debe satisfacer. Los atributos por los cuales puede ser evaluada una arquitectura pueden ser observables, o no observables vía ejecución.

Atributo de Calidad	Descripción
Disponibilidad (Availability)	Es la medida de disponibilidad del sistema para el uso
Confidencialidad (Confidentiality)	Es la ausencia de acceso no autorizado a la información
Funcionalidad (Funtionality)	Habilidad del sistema para realizar el trabajo para el cual fue concebido

Desempeño (Perfomance)	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de restricciones dadas, como velocidad, exactitud o uso de memoria
Confiabilidad (Reliability)	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo.
Seguridad Externa (Safety)	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información.
Seguridad Interna (security)	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.

Tabla 3: Descripción de atributos de calidad observables vía ejecución.

Atributo de Calidad	Descripción
Configurabilidad (Configrability)	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema
Integrabilidad (Integrability)	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
Integridad (Integrity)	Es la ausencia de alteraciones inapropiadas de la información
Interoperabilidad (Interoperability)	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de integrabilidad.
Modificabilidad (Modifiability)	Es la habilidad de realizar cambios futuros al sistema
Mantenibilidad (Maintainability)	Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.

Portabilidad (Portability)	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.
Reusabilidad (Reusability)	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.
Escalabilidad (Scalability)	Es el grado con que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
Capacidad de Prueba (Testability)	Es la medida de la facilita con la que el software, al ser sometido a una serie de pruebas, pueden demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba.
Flexibilidad (Flexibility)	Grado en que el sistema se puede adaptar a los cambios y la puesta en práctica de innovaciones.

Tabla 4: Descripción de atributos de calidad no observables vía ejecución.

4.5 Técnicas de Evaluación.

La evaluación de la arquitectura de software se puede realizar usando varias técnicas, estas se clasifican en cualitativas y cuantitativas. En las técnicas de evaluación cualitativas se utilizan escenarios, cuestionarios o listas de verificación. Por su parte en las técnicas de evaluación cuantitativas se emplean métricas, simulaciones, prototipos, experimentos o modelos matemáticos.

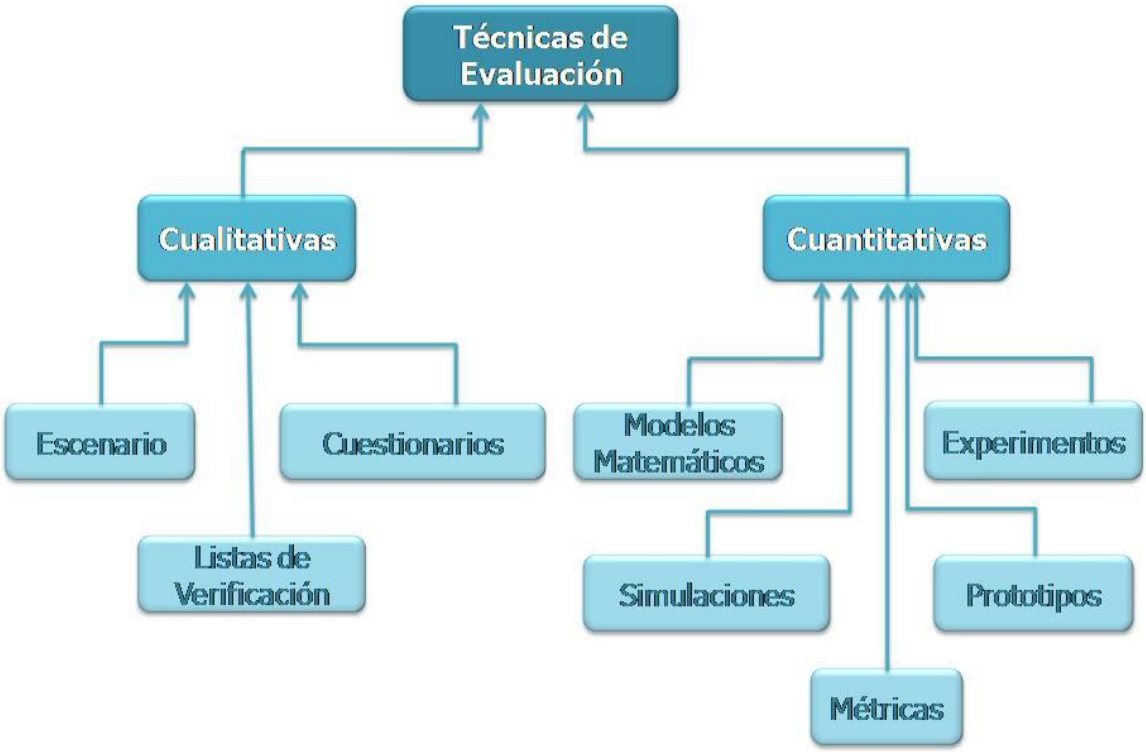

Figura 10: Clasificación de las Técnicas de Evaluación.

Las técnicas de evaluación cualitativas son usadas cuando la arquitectura se encuentra en construcción y arrojan como resultados respuestas de sí o no. Posibilitan evaluar una arquitectura o hacer comparaciones entre arquitecturas candidatas y determinar cual satisface más un atributo de calidad específico. Mientras que las técnicas de evaluación cuantitativas, se usan cuando la arquitectura ya ha sido implantada.

4.6 Métodos de prueba de Arquitectura de Software.

Existen muchos métodos para realizar pruebas a la arquitectura de software, cada uno con sus características especificas. Es necesario tener en cuenta las características fundamentales del software para escoger el método ideal, teniendo en cuenta las fortalezas y debilidades del mismo. Entre los

principales métodos se encuentran: SAAM (Software Architecture Analysis Method), ARD (Active Design Review), ATAM (Architecture Tradeoff Analysis Method) y ARID (Active Review Intermediate Designs).

4.6.1 SAAM.

El Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) es un método basado en escenarios que permite la evaluación de atributos de calidad, especialmente en la modificabilidad. También evalúa la relación del diseño arquitectónico con los requerimientos del sistema. Es posible evaluar una arquitectura o múltiples arquitecturas utilizando SAAM aunque el resultado no emite un valor absoluto referente a la calidad del sistema.

Para la realización de las pruebas SAAM define tres roles:

- Interesados externos: Organización, cliente, usuarios finales, administradores del sistema, etc.
- Interesados internos: Arquitectos de Software, analistas, especialista en requerimientos.
- Equipo SAAM: Líder, expertos en el dominio de la aplicación, expertos externos en arquitectura y secretario.

El método SAAM define una serie de pasos para realizar la evaluación:

- Descripción de la arquitectura.
- Desarrollo de escenarios.
- Clasificación de escenarios.
- Evaluación de la interacción entre escenarios.
- Clasificación y asignación de prioridad de los escenarios.
- Evaluación individual de los escenarios indirectos.

Entre las potencialidades de esta metodología de encuentra que los interesados comprenden con facilidad las arquitecturas evaluadas y la documentación es mejorada. El esfuerzo y el costo de los cambios pueden ser estimados con anticipación Como deficiencia de este método se encuentra la generación de escenarios, pues está basada en la visión de los interesados. No provee una métrica clara sobre la calidad de la arquitectura evaluada. El equipo de evaluación depende de la experiencia de los arquitectos para proponer arquitecturas candidatas.

4.6.2 ADR.

El método de Revisión de Diseño Activo o ARD (Active Design Review) se emplea en la evaluación de diseños bien detallados de unidades de software como los componentes. Se centra en la calidad y el nivel de completamiento de la documentación, y en el nivel de conveniencia y suficiencia de los servicios que contiene el diseño propuesto.

4.6.3 ATAM.

El método ATAM (Architecture Tradeoff Analysis Method) pretende ser un método de identificación de riesgo, un medio de detectar áreas de riesgo potencial dentro de la arquitectura de un sistema intensivo de software complejo. Revela la forma en que una arquitectura satisface ciertos atributos de calidad, provee una visión de la forma que interactúan estos atributos con otros. ATAM se inspira en las áreas de estilos arquitectónicos, análisis de atributos de calidad y el método de evaluación SAAM.

ATAM no presenta ninguna restricción con respecto a la característica de calidad a evaluar. Suele ser relativamente barata y rápida (porque se trata de evaluar el diseño de la arquitectura de los artefactos). El análisis de ATAM producirá resultados en consonancia con el nivel de detalle de la especificación de la arquitectura. Además, no es necesario producir análisis detallados de cualquier atributo de calidad del sistema (como el tiempo medio de retardo o el tiempo de fallo) para tener éxito. ATAM consta de nueve pasos, divididos en cuatro fases.

4.6.3 ARID (Active Review Intermediate Designs).

El método ARID es un híbrido del método ARD y ATAM. ARID constituye un método conveniente para la evaluación de diseños parciales en las etapas tempranas del desarrollo usando técnicas de evaluación basada en escenario. Utiliza para la evaluación del diseño unidades del software como los componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto. Define los roles de Arquitecto, Equipo de verificación y stakeholders y comprende nueve pasos agrupados en dos fases.

El método ARID evalúa el grado en que los atributos de calidad satisfacen en cada uno de los escenarios definidos. Como resultado de la aplicación de dicho procedimiento se obtiene un diseño de alta fidelidad acompañado de una alta familiarización con el diseño de los stakeholders.

4.7 Propuesta de Estrategia de Evaluación de la Arquitectura.

Para la evaluación de la Arquitectura de Software del Sistema de Gestión de Procesos de la Dirección de Televisión Universitaria se proponen dos estrategias de evaluación. Se realizará una primera evaluación en etapas tempranas del desarrollo con el fin de medir los principales atributos de calidad propuestos y tomar medidas de acuerdo a los resultados arrojados por la misma. La segunda evaluación se realizará cuando la arquitectura del sistema se encuentre establecida y su implementación esté concluida lo cual podrá arrojar resultados benéficos para el conocimiento del grado de cumplimiento de los atributos de calidad propuestos, cuando el sistema este en funcionamiento.

4.7.1 Estrategia de evaluación temprana (ARID).

En la estrategia de evaluación temprana de la AS se analizarán los atributos de calidad de Modificabilidad, Interoperabilidad, Seguridad, Funcionalidad y Disponibilidad. Se empleará el método de evaluación ARID que posibilita la evaluación en etapas tempranas del desarrollo de software evaluando el grado en que los atributos de calidad satisfacen en cada uno de los escenarios definidos. Se debe seleccionar al menos tres miembros de equipo de desarrollo (preferiblemente los de mayor conocimientos sobre las funcionalidades de la aplicación) para conformar el equipo de evaluación.
El método ARID comprende nueve pasos agrupados en dos fases:
Fase 1: Actividades Previas.

- Identificación de los encargados de la revisión.
- Preparar el informe de diseño.
- Preparar los escenarios base.
- Preparar los materiales

Fase 2: Evaluación.

- Presentación del ARID.
- Presentación del diseño.
- Lluvia de ideas y establecimiento de prioridad de escenarios.
- Aplicación de los escenarios.
- Conclusiones

Se propone para la selección de los escenarios que los miembros del equipo de desarrollo implicado en el proceso de evaluación establezcan un escenario por cada caso de uso del sistema. En cada uno de estos escenarios se realizarán las pruebas en correspondencia con los atributos de calidad definidos.
A continuación se definirán puntos claves a tener en cuenta durante la realización de la estrategia de evaluación según los atributos de calidad:

Evaluación del atributo Modificabilidad:

Debido a que la Modificabilidad es la habilidad de realizar cambios futuros al sistema, se propone valorar las consecuencias y el costo de:

- Cambiar la interfaz de la aplicación.
- Cambiar el Sistema Gestor de Base de Datos.
- Cambiar el Servidor Web.
- Modificar cualquier proceso efectuado por el sistema.
- Agregar o eliminar un proceso del sistema.

Para este atributo es importante analizar los requerimientos que puedan variar para futuras versiones del sistema y valorar la factibilidad de la arquitectura para adaptarse a los mismos.

Evaluación del atributo Interoperabilidad:

Teniendo en cuenta que la Interoperabilidad es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema, se propone que el equipo de evaluación escogerá las funcionalidades que puedan ser de interés para otras aplicaciones. Valorará el costo de realizar las funcionalidades necesarias para posibilitar la interrelación con otras aplicaciones (preferiblemente usando servicios web).
Se deben tener en cuenta las siguientes funcionalidades:

- Generar, mostrar e imprimir reporte.
- Gestión de medias.
- Asignación de equipamientos.

El equipo de desarrollo deberá incluir otras funcionalidades.

Evaluación del atributo Seguridad:

Teniendo en cuenta que la Seguridad es la medida de ausencia de errores que generan pérdidas de información y la habilidad del sistema para resistir a intentos de usos no autorizados y negación del servicio, mientras se sirve a usuarios legítimos, se propone:

- El equipo de evaluación seleccionará diferentes escenarios donde el acceso a los recursos esté limitado a un rol de usuario, al menos dos escenarios por cada rol del sistema. Se probará en cada escenario definido el acceso de usuario con diferente roles.
- Realizar intentos de acceso por usuarios con privilegios y usuarios sin privilegios.
- Introducir cadenas que contengan códigos que puedan dañar el funcionamiento de la aplicación.

Evaluación del atributo Funcionalidad:

Teniendo en cuenta que la Funcionalidad es la habilidad del sistema para realizar el trabajo para el cual fue concebido, se propone que el equipo de evaluación medirá el grado de cumplimiento de cada uno de los requerimientos funcionales definidos en el sistema.

Evaluación del atributo Disponibilidad:

Teniendo en cuenta que la Disponibilidad es la medida de disponibilidad del sistema para el uso, se propone:

Valorar las consecuencias sobre los servicios cuando se efectúan las siguientes acciones:

- Realizar modificaciones en el Sistema Gestor de Bases de Datos sin afectar la conexión con el resto de la aplicación.
- Detener los servicios del Sistema Gestor de Base de Datos.
- Realizar modificaciones en el Servidor Web sin afectar la conexión con el resto de la aplicación.
- Detener los Servicios Del Servidor Web.
- Solicitar el mismo recurso por múltiples usuarios.
- Solicitar recursos independientes por múltiples usuarios.

Incorporar a estas pruebas otras acciones de mantenimiento que el equipo de evaluación estime conveniente. Los resultados obtenidos de la evaluación se expondrán ante todos los involucrados en el desarrollo del sistema. Si se detectan dificultades se procederá a valorar los cambios que se imponen en la arquitectura, aplicar los mismos de forma inmediata y comenzar un nuevo proceso evaluativo empleando nuevamente el método ARID, enfatizando en los errores detectados en la anterior evaluación.

4.7.2 Estrategia de evaluación Tardía (ATAM).

En la estrategia de evaluación tardía de la AS se realizará una vez que el sistema este implementado completamente y esté puesto en funcionamiento. Se analizarán los atributos de calidad definidos en la evaluación temprana Modificabilidad, Interoperabilidad, Seguridad, Funcionalidad y Disponibilidad conjuntamente con los atributos de Mantenibilidad, Reusabilidad y Flexibilidad. Se empleara el método de evaluación ATAM que revela la forma en que la arquitectura satisface los atributos de calidad definidos y provee una visión de la forma que estos atributos interactúan entre sí. ATAM define tres tipos de escenarios:

- Escenarios de casos de Usos son los que describen las interacciones de los usuarios con el sistema.
- Escenarios de Crecimiento son los que representa la anticipación de los cambios.
- Escenarios exploratorios son los que tienen como objetivo fundamental exponer al sistema a condiciones extremas, llevar el diseño a casos extremos de condiciones.

ATAM consta de nueve pasos, divididos en cuatro fases.

Presentación	
Presentación del ATAM	El líder de evaluación describe el método al resto del equipo y se establece el alcance.
Presentación de las metas de negocio.	Se realiza la descripción de las metas del negocio que motivan el esfuerzo y aclara que se persiguen objetivos de tipo arquitectónico.
Presentación de la Arquitectura	El arquitecto describe la arquitectura, enfocándose en cómo esta cumple con los objetivos del negocio
Investigación y Análisis	
Identificación de los enfoques arquitectónicos.	Se detectan sin entrar en detalles.
Generación del Utility Tree.	Se priorizan los atributos de calidad que engloban las funcionalidades del sistema, especificados en forma de escenarios. Se tienen en cuenta los estímulos y respuestas,

	así como se establece las prioridades entre ellos
Análisis de los enfoques arquitectónicos.	Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos identificados. Se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance.
Pruebas	
Lluvia de ideas y establecimiento de prioridad de escenarios.	Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.
Análisis de los enfoques arquitectónicos.	Este paso repite las actividades del análisis de los enfoques arquitectónicos, haciendo uso de los resultados de la lluvia del paso anterior. Los escenarios son considerados como caso de prueba para confirmar el análisis realizado hasta el momento.
Reporte	
Presentación de los resultados	Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

Tabla 5: Fases del ATAM.

De forma similar a la anterior estrategia los resultados obtenidos de la evaluación se expondrán ante todos los involucrados en la realización del sistema. Si se detectan dificultades se procederá a valorar la factibilidad de realizar cambios en la arquitectura u otras variantes que contribuyan con la mitigación de estos problemas. De ser necesario se realizarán los cambios y al concluir el mismo se proponen realizar nuevamente la evaluación de la arquitectura empleando el método ATAM.

4.8 Conclusiones del Capítulo.

En el presente capítulo se han analizado los elementos fundamentales relacionados con la evaluación de la Arquitectura de Software , destacando las ventajas de realizar la evaluación, pues permite identificar el impacto que tiene la arquitectura sobre los atributos de calidad definidos, posibilita detectar los problemas, debilidades y puntos de interés para corregir a tiempo los mismos. Se plantearon las etapas para la evaluación de la AS y la propuesta dada para la misma.

CONCLUSIONES GENERALES.

En el presente trabajo se analizaron los principales aspectos para el desarrollo de la arquitectura. La selección del estilo arquitectónico, las herramientas de modelado, y de desarrollo además de los requisitos no funcionales realizándose a partir de las características o funcionalidades que debía brindar el Sistema de Gestión de Procesos para la Dirección de Televisión Universitaria.

Se realizó un estudio de los principales patrones arquitectónicos seleccionándose para la solución el patrón Modelo Vista Controlador, definiéndose así sus principales componentes, configuraciones y restricciones.

Después de un profundo estudio sobre las distintas metodologías de desarrollo de software se determinó usar la metodología RUP, la cual brinda la documentación necesaria para la el sistema a realizar, garantizando su desarrollo, soporte, mantenimiento y producción de nuevas versiones.

A través de la descripción de la arquitectura mediante las vistas definidas por RUP se garantizó que se una visión de todos los modelos del sistema y de los elementos arquitectónicamente significativos durante el proceso de desarrollo.

Una vez propuesta la AS pues se analizaron elementos fundamentales relacionados con la evaluación de la Arquitectura de Software. Además se mostraron las ventajas que conlleva esta evaluación, la cual permite identificar el impacto que tiene la misma sobre los atributos de calidad definidos, posibilitando detectar los problemas y debilidades.

De forma general con el desarrollo del trabajo se generaron los artefactos necesarios que permiten la implementación del sistema a desarrollar.

RECOMENDACIONES

- El refinamiento permanente de la arquitectura propuesta, durante el ciclo de desarrollo.
- Valorar el uso de otro framework para el desarrollo de la aplicación.
- Realizar la evaluación de la arquitectura mediante las estrategias de evaluación definidas.

Trabajos Citados

1. Buschmann, Frank. ***Pattern Oriented Software Architecture 1 edition.*** **August 8, 1996. 0471958697.**
2. Garlan, David and Shaw, Mary. ***An Introduction to Software Architecture.*** **USA : s.n., Enero, 1994.**
3. Buschmann, Frank. **Pattern-Oriented Software Architecture. *A System of Patterns.* Inglaterra : s.n., 1995. Vol. Volume 1.**
4. Pressman. **2002.**
5. ***dcc.uchile.cl.*** **[Online] [Cited: 12 13, 2009.] http://www.dcc.uchile.cl.**
6. Champion, Michael. **"Towards a reference architecture for Web services". *Conference and Exposition.* Filadelfia : s.n., 2003.**
7. Ballestero, Carlos rafael. ***Arquitectura de Software del Sistema de GEstion de Informacion.*** **Ciudad de la Habana : s.n., 2007.**
8. Gómez, Omar Salvador. ***Evaluando Arquitecturas de Software.*** **Mexico : Brainworx, 2007. 0888..**
9. Gustavo Andrés Brey, Gastóm Escobar, Nicolas Passerini y Juan Arias. ***Arquitectura de Proyectos de IT. Evaluación de Arquitecturas.*** **Buenos Aires. Univeridad Tecnológica : s.n.**
10. Paul Clements Clements, Paul. ***A Survey of Architecture Description Languages. .*** **Alemania : Proceedings of the International Workshop , 1996.**
11. Bass, Len, Clements, Paul and Kazman, Rick. ***Software Architecture in Practice.*** **s.l. : Second Edition., April 11, 2003. 0-321-15495-9.**
12. Buschmann, Frank, Meunier, Regine, Rohnert , Hans, Sommerlad , Peter and Stal, Michael. ***Pattern-Oriented Software Architecture .*** **England : s.n., 1996. 0 417 95869 7.**
13. Christopher Alexander. ***www.microsoft.com .*** **[Online] 1977. [Cited: 12 10, 2009.] http://www.microsoft.com.**
14. Kiccillof, Carlos Reynoso , Nicolás. ***Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.*** **Buenos Aires : s.n., 2004.**
15. Fowler. **2002.**
16. Canal, Carlos. **Arquitectura, marcos de trabajo y patrones. Universidad de Málaga : s.n., marzo 2005.**
17. MINISTERIO DE INDUSTRIA, TURISMO Y COMERCIO. **TDT(Televisión Digital Terrestre). [Online] octubre 1, 2009. [Cited: diciembre 1, 2009.] http://www.televisiondigital.es.**
18. Martínez, Yarisel Rodríguez. ***Diseño de un Sistema Gestor de Guía Electrónica de Programación.*** **Habana : Universidad de las Ciencias Informáticas, 2009.**
19. Huidobro, José Manuel. ***IPTV, la televisión a través de internet.*** **2008.**
20. Huawei Technologies Co., Ltd. **Huawei. [Online] Huawei Technologies Co., Ltd, 2009. http://www.huawei.com/es/catalog.do?id=-4.**
21. Macias, Yolanda Benitez. ***Sistema para la gestión y control de la cartelera del canal cultural de la Universidad de las Ciencias Informáticas.*** **Ciudad de la Habana : Universidad de la Ciencias Informáticas, 2008.**

22. José H. Canós, Patricio Letelier y Mª Carmen Penadés. ***Métodologías Ágiles en el Desarrollo de Software.*** **Valencia : DSIC -Universidad Politécnica de Valencia.**

23. Vázquez, Carlos Luis Serrano Rosales , Solangel Rodríguez. ***Desarrollo de Biblioteca de Métodos Numéricos (BMN), referente a Sistemas de Ecuaciones Lineales, Sistemas de Gran Dimensión y Poco Densos e Integración Numérica.*** **Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2008.**

24. Proyecto GNU. **La Definición de Software Libre. [Online] 2008. http://www.gnu.org/philosophy/free-sw.es.html.**

25. Booch. **1999.**

26. Beck, Kent. ***Extreme Programming Explained. Embrace Change.*** **s.l. : Addison-Wesley Professiona, 2000. 0201616416.**

27. Rosenblum, Nenad Medvidovic y David S. ***Domains of Concern in Software Architectures and Architecture Description Languages. .*** **California : s.n., 1997.**

28. Gómez, Omar Salvador. ***Evaluando Arquitecturas de Software. Parte 1 Panorama General.*** **Mexico : s.n., 2007. 0888.**

BIBLIOGRAFÍA CONSULTADA

An Introduction to Software Architecture. David Garlan, Mary Shaw. New Jersey : V.Ambriola and G.Tortora, 1994, Vol. 1. 15213-3890.

Buschmann, Frank, Meunier, Regine, Rohnert , Hans, Sommerlad , Peter and Stal, Michael Pattern-Oriented Software Architecture 1996 England 0 417 95869 7.

González, Aleksander, et al. Método de Evaluación de Arquitecturas de Software Basadas en Componentes (MECABIC). s.l. : Mexico : s.n., 2005, Vol. vol 1.

Grupo Soluciones Innova S.A. G.S.I Grupo Soluciones Innova . [Online] 2007. [Cited: enero 21, 2009.] http://www.rational.com.ar/herramientas/roseenterprise.html

Clements, Paul, Kazman, Rick and Klein, Mark. Evaluating software architectures: methods and case studies. s.l. : Addison-Wesley, 2002.

Coplien, James O. Idioms and Patterns as Architectural Literature. s.l. : IEEE Software, Enero/febrero 1997. Vol. 14(1).

Larman, Craig. UML y Patrones. Introducción al análisis y diseño orientado a objetos. s.l. : Prentice Hall. 84-205-3438-2.

ORACLE CORPORATION. Oracle. [Online] [Cited: enero 25, 2009.] http://www.oracle.com/global/es/index.html.

Escobar, Miguel A. Paco. La calidad de software y su importancia en el mercado . [PDF] s.l. : Kernel Microsystems, 2003.

Company Headquarters. Visual Paradigm. [Online] 2002. [Cited: enero 20, 2008.] http://www.visual-paradigm.com/.

Reynoso, Carlos and Kicillof, Nicolás. De Lenguajes de descripción arquitectónica de Sofware (ADL), version 1. Argentina : Universidad de Buenos Aires, Marzo, 2004.

GLOSARIO DE TÉRMINO

SGPDTU: Sistema de Gestión de Procesos para la Dirección de Televisión Universitaria.

IEEE: Institute of Electrical and Electronics Engineers (Instituto de Ingenieros Eléctricos y Electrónicos).

TIC: Tecnologías de Información y las Comunicaciones.

HTTP: HyperText Transfer Protocol (Protocolo de Transporte de Hipertexto).

Framework: Marco de trabajo, solución reutilizable y extensible.

Herramientas CASE (Computer Aided Software Engineering): Se incluyen una serie de herramientas, lenguajes y técnicas de programación que permiten la generación de aplicaciones de manera semiautomática.

RUP: Metodología de desarrollo de software basada en UML. Organiza el desarrollo de software en 4 fases.

XP: Es una metodología del software que persigue simplificar los procesos de desarrollo del software. Fue diseñada para ser usado con equipos de desarrollo pequeños que necesiten desarrollos ágiles y requerimientos cambiantes.

ANEXO 1

Un problema que habría que tratar sistemáticamente es la discrepancia en las definiciones de Arquitectura de Software que prevalecen en la academia y las que son comunes en la industria, incluyendo en ella tanto a los proveedores de software de base como a los equipos de desarrollos de las empresas. Jan Bosch ha confeccionado una tabla de contrastes que seguramente podría enriquecerse con nuevas antinomias.

Academia	Industria
La arquitectura se define explícitamente.	Prevalece una comprensión conceptual de la arquitectura. Las definiciones explícitas son mínimas, eventualmente mediante notaciones.
La arquitectura consiste en componentes y conectores de primera clase.	No hay conectores explícitos de primera clase (a veces hay soluciones ad hoc de binding en tiempo de ejecución).
Los lenguajes de descripción de arquitectura (ADLs) describen la arquitectura explícitamente y a veces la generan.	Se utilizan lenguajes de programación.
Los componentes reutilizables son entidades de caja negra.	Los componentes son grandes piezas de software de estructuras interna compleja, no necesariamente encapsulados.
Los componentes tienen interfaces con un solo punto de acceso.	Las interfaces se proporcionan mediante entidades (clases en los componentes). Las entidades de interfaz no tienen diferencias explícitas de entidades que no son de interfaz.
Se otorga prioridad a la funcionalidad y la verificación formal.	La funcionalidad y los atributos de calidad (rendimiento, robustez, tamaño, reusabilidad, mantenibilidad) tienen igual importancia

Tabla 6: Discrepancia en las definiciones de AS.

ANEXO 2

Diagramas de clases.

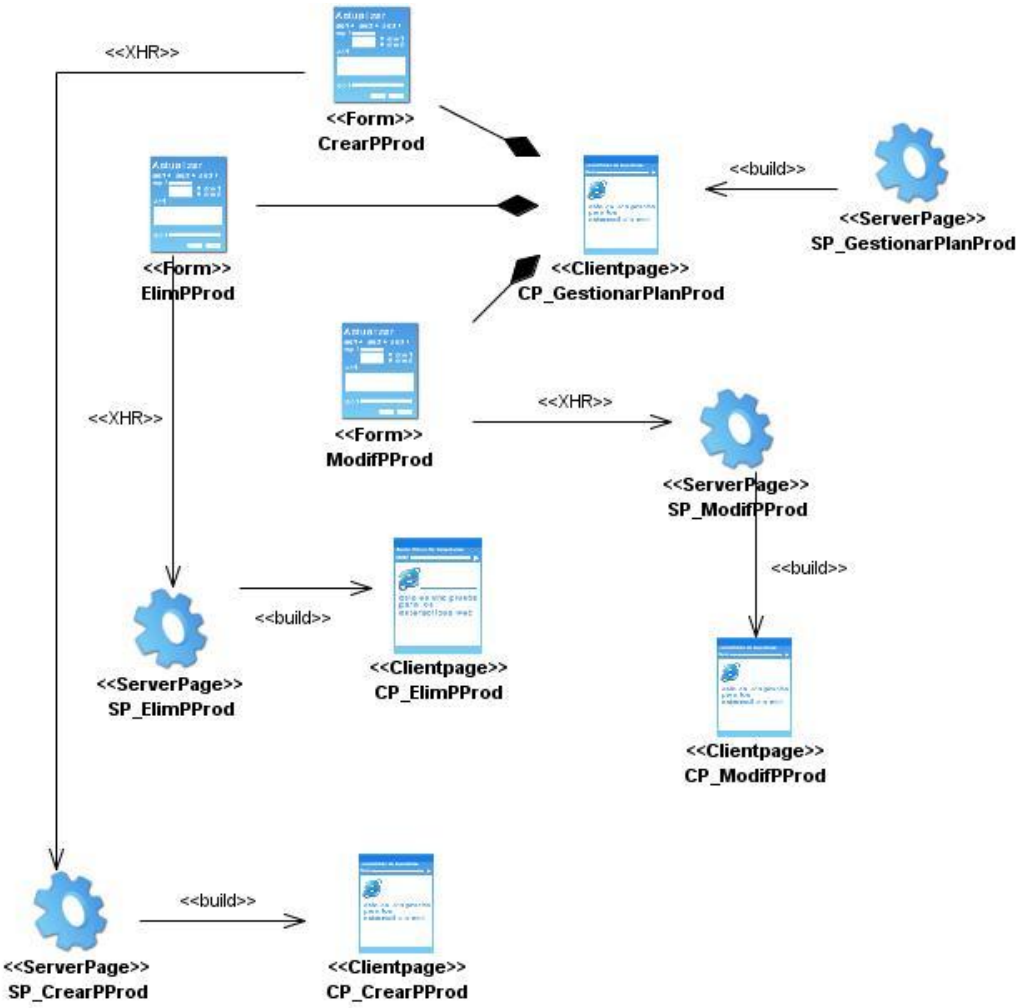

Figura 11: Diagrama de Clases para Gestionar Plan de Producción.

ANEXO 3

Figura 12: Fases de la Metodología XP.