Universidad de las Ciencias Informáticas
Facultad 9

TRABAJO DE DIPLOMA

PARA OPTAR POR EL TÍTULO DE INGENIERO EN INFORMÁTICA

TÍTULO: Diseño de una arquitectura para el Portal del Grupo Calidad de la Facultad 9

AUTOR: Tahymi Ramírez García

TUTOR: Ing. Julio Alberto Leyva Durán

CO-TUTOR: Ing. Liester Cruz Castro

Habana, Junio 2010

"Año 52 de la Revolución"

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 _____ del año _____.

Tahymi Ramírez García	Julio Alberto Leyva
_____	_____
(Autor)	(Tutor)

Tabla de contenido

Resumen

El desarrollo de las ciencias informáticas en las últimas décadas ha sido considerable y conjuntamente el crecimiento ilimitado de la información, por tanto la gestión de la misma conquista un papel protagónico donde uno de los principales requisitos es la calidad y la ganancia en cuanto a tiempo, recursos e ingresos.

En el Grupo Calidad de la Facultad 9 de la Universidad de las Ciencias Informáticas (UCI), se han detectado problemas con el tratamiento de la documentación que poseen, por lo que el presente trabajo de diploma pretende diseñar una arquitectura que, acorde a las necesidades del proyecto, permita optimizar el proceso de gestión de la información y lograr una mayor eficiencia en el entorno productivo.

Introducción

La evolución en el campo de la informática constituye hoy un proceso que se ha globalizado mundialmente. No se puede afirmar que exista algún país al margen del desarrollo que ha tenido la industria del software, pero sí que muchos dependen, casi en su totalidad, de esta rama tecnológica. Esto ha contribuido a que se considere como una ciencia de las más importantes de finales del pasado siglo y actualidad del presente.

Cuba, luego de la caída del campo socialista y la acentuación del bloqueo impuesto en la isla desde el 7 de febrero de 1962 por parte de los Estados Unidos, queda interrumpida económico, financiero y comercialmente. Con la intención de contrarrestar la realidad existente, el país se ve obligado a priorizar e invertir en distintos sectores de la economía, surgiendo así, la necesidad de informatizar la sociedad. Nace entonces la Universidad de las Ciencias Informáticas (UCI), institución educacional que tiene como objetivo vincular el proceso de enseñanza con la producción de software. Esta, entre otras características hacen que sea catalogada como una cuidad digital; en donde el conocimiento de informaciones a cualquier nivel es trascendental, razón por la cual una buena gestión de las mismas se hace imprescindible.

¿Qué se entiende por gestión de la información?

Las especialistas en sistemas de gestión de la información y de la documentación en las organizaciones: Carlota Bustelo Ruesta y Raquel Amarilla exponen que “La gestión de la información se puede definir como el conjunto de actividades realizadas con el fin de controlar, almacenar y, posteriormente, recuperar adecuadamente la información producida, recibida o retenida por cualquier organización en el desarrollo de su actividades.” (Bustelo, y otros, 2008)

Una inadecuada gestión de la información implica que empresas que no puedan tener la necesaria, en el tiempo indicado se encuentren inmersos en distintas anomalías. La toma tardía o apresurada de decisiones, productos sin la calidad requerida y demoras en la entrega de proyectos, son algunas de las deficiencias que se pueden encontrar y que deriban en servicios ineficientes, pocos ingresos y baja competitividad en el mercado. Con la revolución de las Tecnologías de la Información y las

Comunicaciones (TIC), se han podido crear software que ayudan a compensar las pérdidas de tiempo, recursos e ingresos que devienen de la ineficiente gestión de la información y que estas empresas han asumido de acuerdo a sus necesidades.

Partiendo de todo lo explicado anteriormente se ha constatado que el Grupo Calidad de la Facultad 9 no cuenta con un sistema que gestione de manera ventajosa la documentación con la que trabaja. En dicho grupo se manejan usualmente gran número de planillas, las cuales no se encuentran disponibles para los profesores y estudiantes en todo momento, lo que trae consigo que en ocasiones se le pueda ver duplicada. La comunicación entre los integrantes del proyecto habitualmente se realiza vía correo electrónico y solo entre integrantes de un área, dejando al margen a los que participan en las demás, para los cuales, generalmente, la información es inaccesible.

En consecuencia a lo ya expuesto, se determinó el siguiente **Problema Científico**: *La ineficiente gestión de la información con respecto a la documentación existente en el Grupo Calidad de la Facultad 9, deviene en un manejo lento de la misma y tributa a retrasos en el flujo normal del trabajo del proyecto.*

El **objeto de estudio** de la investigación se centra en el *Proceso de Gestión de la Información del Grupo Calidad de la Facultad 9*, siendo el **campo de acción** la *definición de la arquitectura de los Portales.*

El **objetivo general** del trabajo consiste en *diseñar una arquitectura para el Portal del Grupo Calidad de la Facultad 9 y s*e definen como **objetivos específicos*:***

- Identificar los patrones arquitectónicos adecuados para lograr una organización del portal de forma tal que permita agregar nuevas funcionalidades sin necesidad de realizar cambios significativos en la arquitectura.

- Identificar los estilos arquitectónicos que puedan ser utilizados en la descripción de la arquitectura del portal a desarrollar.

- Comparar la arquitectura propuesta con otras soluciones similares.

La **idea a defender** plantea que: *El diseño de una arquitectura, permitirá que se agilice el manejo de la información y el flujo de trabajo en el proyecto.*

Con el objetivo de guiar, controlar y evaluar la investigación se definieron las siguientes **tareas:**

1. Identificar los procesos de gestión de la información del Grupo Calidad de la Facultad 9.
2. Identificar propuestas de los estilos arquitectónicos que se puedan utilizar.
3. Comparar diferentes propuestas de modelos de arquitecturas consultadas, teniendo en cuenta aspectos positivos y negativos.
4. Seleccionar las herramientas y tecnologías a utilizar para el desarrollo del Portal.
5. Definir una propuesta arquitectónica que cumpla con los requerimientos de la aplicación.
    5.1 Identificar, fundamentar y reutilizar patrones arquitectónicos que puedan ser utilizados en la arquitectura del Portal y que se adecuen a las necesidades del mismo.
6. Evaluar la arquitectura propuesta con estrategias de pruebas.

Posibles resultados que presupone la investigación:

- El diseño flexible de una arquitectura que les permita a los analistas y programadores contar con elementos necesarios para el desarrollo del Portal de Calidad.
- Modelo de implementación, Documento de arquitectura, Modelo de Despliegue

Los **métodos teóricos** que se proponen para dar cumplimiento a las tareas y alcanzar los resultados son:

Análisis Histórico-Lógico: para dar seguimiento a la evolución de la arquitectura a través de los años y determinar las corrientes principales y tendencias actuales de esta.

Analítico-Sintético: este método se ha empleado para poder resumir la gran bibliografía existente con respecto a los estilos y patrones arquitectónicos a los que se hace referencia en la investigación.

Modelación: en la investigación se realizan diferentes abstracciones como son los modelos de los estilos arquitectónicos, de casos de uso, entre otros.

Organización del Trabajo de Diploma

En el Capítulo 1:"Fundamentación Teórica" se realizará un estudio de los principales temas relacionados con el dominio de la investigación, conceptos de arquitectura de software, estilos y patrones arquitectónicos. En el segundo capítulo "Descripción de las tecnologías a utilizar" se describirán las tecnologías a utilizar con la finalidad de definir los que resulten más adecuados para el desarrollo del portal. Luego se realizará el "Diseño de la arquitectura propuesta" en el capítulo 3 donde se definirá la arquitectura, principalmente mediante las vistas arquitectónicas de Casos de Uso, Lógica, Implementación y Despliegue. Finalmente, en el capítulo 4: "Evaluación de la Arquitectura" se abordarán conceptos de evaluación de la arquitectura y se procederá a evaluar la arquitectura propuesta en el trabajo, con el objetivo de verificar si es la propuesta arquitectónica realizada adecuada o no para el desarrollo del Portal del Grupo Calidad de la Facultad 9 .

Capítulo 1: Estado del Arte y Fundamentación Teórica

1-Introducción

La arquitectura de software (en lo adelante AS) tiene una relación inmediata y un compromiso directo con el éxito del software a implementar; por lo que en el presente capítulo se realiza un estudio del estado del arte del tema en cuestión y sus tendencias actuales; se abordan además los diferentes conceptos relacionados con el dominio de la investigación: AS, estilos y patrones arquitectónicos, las herramientas y tecnologías a utilizar para el desarrollo del portal del Grupo Calidad de la Facultad 9. El objetivo central de este apartado es el logro de un buen entendimiento de los temas esenciales que se relacionan con la AS.

1.1-Descripción del entorno

El Grupo Calidad de la Facultad 9 de la Universidad de Ciencias Informáticas se organiza en cuatro diferentes áreas. Estas áreas se corresponden con los procesos que se llevan a cabo dentro del grupo. Procesos que han sido creados por la importancia que manifiesta el cumplimiento del principal objetivo de la existencia de este grupo. Este objetivo consiste en verificar que los software que se creen como parte de la producción de la facultad, tengan la calidad demandada y cumplan con los lineamientos requeridos para su implantación y comercialización.

Estos procesos son:

- Proceso de Inicio de Proyecto

  El área correspondiente a este proceso tiene como objetivo principal garantizar que los proyectos productivos cumplan con las normativas y condiciones especificadas para alcanzar un nivel de calidad básico como requerimiento para ser certificado. La certificación no es más que el hecho de que se le permita a dichos proyectos comenzar a trabajar con el producto pronosticado.

- Proceso de Auditorías y Revisiones

Es el proceso que se pone en marcha al realizar auditorías o revisiones a los proyectos productivos, que no es más que la verificación de que el trabajo que se está haciendo este acorde a lo estipulado. Para esto se le pide la documentación existente al proyecto en cuestión y se comprueba que tenga paridad con el trabajo realizado y que cumpla con los lineamientos de calidad requeridos. Además de que dicha documentación se encuentre debidamente actualizada.

- Proceso de Recursos Humanos

  Los trabajadores de esta área se dedican a la colección y registro de los datos de los estudiantes y profesores que laboran en el proyecto, así como al inventario de las computadoras que están a disposición del grupo.

- Proceso de Estrategia de Prueba

  En esta área se realiza una compilación de los diferentes tipos de pruebas que se hacen en los proyectos productivos de la facultad. Es la encargada además de capacitar a los probadores de cada uno de estos proyectos y de evaluar que el producto obtenido está listo para su comercialización en el mercado.

La mayoría de estos procesos se realizan de forma manual, generan un número excesivo de documentación y tornan el trabajo lento y tedioso. Para la mejora de estos y para agilizar el trabajo del Grupo Calidad de la Facultad 9, se ha planteado realizar un Portal Web. El presente trabajo de diploma se propone el diseño de una arquitectura para este portal.

1.2-Arquitectura de Software

La historia de la AS comienza con el emblemático científico de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, Edsger Dijkstra, quien hacia 1968 propuso que se estableciera una estructuración correcta de los sistemas de software. Parten, además, de sus ensayos la noción de los sistemas operativos organizados en capas y los llamados “niveles de abstracción” tan común en las prácticas actuales. Aunque el término arquitectura no fue mencionado por el científico, sentó las bases como ideas predecesoras para lo que luego se le conoció como programación en grande y que sirvió de

argumento para los programadores de la época. Para 1969 en la conferencia de la Organización del Tratado del Atlántico Norte (OTAN) realizada en Roma, se realizaron varias afirmaciones a lo planteado un año antes por Dijkstra, pero la arquitectura como concepto no tenía aún precisión semántica, aunque varios autores con el paso del tiempo escriben ensayos y proponen definiciones, no es hasta años después que se le conoce con el sentido que se le da en la actualidad.

Una novedad importante es el hecho de que en la década de 1970 apareciera el diseño estructurado de los primeros modelos explícitos de desarrollo de software. Estos modelos dejaban atrás el conocido desarrollo en cascada y propicia que se independice paulatinamente el diseño de la implementación. Pero estos métodos de desarrollo estructurado no durarían mucho pues hacia 1980 un nuevo paradigma ocuparía su lugar, el de la programación orientada a objetos, que según parecía, permitía modelar el dominio del problema y el de la solución en un solo lenguaje de implementación.

Finalmente es en 1992 que se realiza el primer estudio titulado Fundamentos para el Estudio de la Arquitectura de Software por Dewayne Perry profesor del Departamento de Ingeniería Eléctrica y Computación de la Universidad de Austin en Texas y Alexander Wolf profesor del Departamento del Imperial College de Londres. En este estudio los autores proponen una definición para la AS análoga a la arquitectura de edificios, declarando lo que se considera un vaticinio: “La década de 1990, creemos, será la década de la arquitectura de software. Usamos el término “arquitectura” en contraste con “diseño”, para evocar nociones de codificación, de abstracción, de estándares, de entrenamiento formal (de los arquitectos de software) y de estilo. Es tiempo de re-examinar el papel de la arquitectura de software en el contexto más amplio del proceso de software y de su administración, así como señalar las nuevas técnicas que han sido adoptadas.” (Perry, y otros, 1992). Es así que se caracteriza entonces la década de 1990 como época de florecimiento, enraización y utilización de la arquitectura de software, vasta en eventos como el surgimiento de la programación basada en componentes y de los patrones arquitectónicos.

Llega entonces el siglo XXI donde, según considera el Licenciado Carlos Reynoso, Profesor Titular Regular de la Universidad de Buenos Aires, aparece la AS “dominada por estrategias orientadas a líneas de productos y por establecer modalidades de análisis, diseño, verificación, refinamiento, recuperación,

diseño basado en escenarios, estudios de casos y hasta justificación económica, redefiniendo todas las metodologías ligadas al ciclo de vida en términos arquitectónicos" (Reynoso, 2004)

Son muchas las definiciones para AS, pero si bien discrepan, en algunos aspectos, la mayoría de los autores la consideran como un punto de vista que concierne a un alto nivel de abstracción. Para Paul Clemens, profesor del Instituto de Ingeniería de Software de la Universidad Carnegie Mellon en Pittsburgh, la AS es "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" (Clemens, 1996). Según la metodología Proceso Unificado de Racional (RUP) la arquitectura "es la organización o la estructura de los componentes importantes del sistema que interactúan mediante interfaces, con componentes compuestos de interfaces y componentes cada vez más pequeños (IBM, 2003). Por su parte el Instituto de Ingenieros Electricistas y Electrónicos (IEEE) plantea la que se considera como definición oficial y que fue adoptada por importantes empresas como Microsoft: "La arquitectura de software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán y los principios que sustentan su diseño y evolución" (Organización Internacional de Normalización, 2007).

Por la variedad de conceptos, de forma resumida y esclarecedora, extrayendo definiciones y puntos de vista, los profesores de la Universidad Carnegie Mellon en Pittsburgh, Mary Shaw y David Garlan en su libro Arquitectura de Software: perspectivas sobre una disciplina emergente. (Shaw, y otros, 1996) clasifican los modelos de la forma que sigue:

Modelos estructurales: Sostienen que la AS está formada por componentes, conexiones entre ellos y (usualmente) otros aspectos tales como configuración, estilo, restricciones, semántica, análisis, propiedades, racionalizaciones, requerimientos, necesidades de los participantes. El trabajo en esta área está caracterizado por el desarrollo de Lenguajes de Descripción.

Modelos de Framework: Son similares a la vista estructural, pero su énfasis primario radica en la (usualmente una sola) estructura coherente del sistema completo, en vez de concentrarse en su composición. Los modelos de Framework a menudo se refieren a dominios o clases de problemas específicos. El trabajo que ejemplifica esta variante incluye arquitecturas de software.

Modelos dinámicos: Este modelo es llamado dinámico porque puede referirse a los cambios en la configuración del sistema, o a la dinámica involucrada en el progreso de la computación, tales como valores cambiantes de datos.

Modelos funcionales: Una pequeña minoría de estudiosos y académicos considera la arquitectura como un conjunto de componentes funcionales, organizados en capas que proporcionan servicios hacia arriba. Es tal vez útil pensar en esta visión como un Framework particular.

1.3- Corrientes Principales de la Arquitectura de Software.

Arquitectura estructural

Constituye la corriente fundacional y clásica de la disciplina. Concebida, representada y tratada por varios académicos como: Mary Shaw, Paul Clements, David Garlan, Robert Allen, Gregory Abowd, John Ockerbloom entre otros. Aunque estos profesores han hecho el esfuerzo más importante por el reconocimiento de la AS como disciplina, sus categorías y herramientas son todavía mal conocidas en la práctica industrial. Es una corriente donde el diseño arquitectónico es el de más alto nivel de abstracción el cual puede no armonizar con la configuración de las aplicaciones y en donde rara vez se encontrarán referencias a lenguajes de programación o piezas de código, y en general nadie habla de clases o de objetos

Arquitectura como etapa de ingeniería y diseño orientada a objetos.

Modelo que se entiende como una de las corrientes principales en las tendencias actuales de la AS. Propuesto por James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman y otros, se relaciona fuertemente con el Lenguaje de Modelado Unificado (UML) el cual es utilizado para las descripciones arquitectónicas. En este la AS se desenvuelve entre la fase inicial y preliminar de un proceso pero no estrechamente relacionada con los requerimientos o el diseño. Es una tendencia que reconoce el valor

primordial de la abstracción y del ocultamiento de información pero desde un punto de vista más centrado en la programación que en la arquitectura en sí, la cual a su vez es confundida con el modelado y el diseño.

Arquitectura basada en patrones.

Reconoce la importancia de un modelo emanado del diseño Orientado a Objetos (OO), esta corriente surgida hacia 1996 no se encuentra tan rígidamente vinculada a UML. El texto sobre patrones que esta variante reconoce como referencia es la serie de Patrones Orientados a la Arquitectura de Software (POSA) del ingeniero de software alemán Frank Buschmann y en segundo lugar el texto de la Banda de los Cuatro(o GoF como se le conoce por sus siglas en ingles, formada por los estudiosos Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides) referido a patrones en el texto “Design Patterns, Elements of reusable object-oriented software (Los patrones de diseño, elementos reutilizables del software orientado a objeto)”. En esta manifestación de la AS prevalece cierta tolerancia hacia modelos de procesos tácticos, no tan macroscópicos. El diseño consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura.

Arquitectura procesual y metodologías.

Se crea a principios del siglo XXI, con el Instituto de Ingeniería del Software (SEI por sus siglas en ingles) como médula y con participación de arquitectos tales como Rick Kazman, Len Bass, Paul Clements de la Universidad Carnegie Mellon. Es una corriente que intenta establecer modelos de ciclo de vida y técnicas de delimitación de requerimientos, brainstorming (tormenta de Ideas), diseño, análisis, selección de alternativas, validación, comparación, estimación de calidad y justificación económica específicas para la arquitectura de software. Otras variantes dentro de la corriente procesual caracterizan de otras maneras de etapas del proceso: extracción de arquitectura, generalización y reutilización.

Arquitectura basada en escenarios.

Es la corriente más nueva y menos reconocida por distintos autores por considerarla un híbrido de las anteriores, en esta los teóricos y practicantes se mueven por los cánones de la arquitectura procesual y utiliza diagramas de casos de uso UML como herramienta informal u ocasional al igual que la arquitectura como etapa de ingeniería y diseño orientada a objetos. Aunque esta corriente posee varios detractores

tiene su significación por recuperar la unión de la AS con los requerimientos y la funcionalidad del sistema, característica que había llegado a ser difusa en la arquitectura clásica. Los autores vinculados con esta modalidad han sido entre otros los arquitectos holandeses de la Universidad Técnica de Eindhoven, de la Universidad Brije, de la Universidad de Groningen y de Philips Research: Mugurel Ionita, Dieter Hammer, Henk Obbink, Hans de Bruin, Hans van Vliet, Eelke Folmer, Jilles van Gurp, Jan Bosch, por lo que se puede decir que ha sido un movimiento predominantemente europeo, con centro en Holanda.

1.4 Estilos Arquitectónicos.

Cuando se define la arquitectura de un software se debe escoger cual o cuales estilos arquitectónicos serán utilizados, ponderando el hecho de que el éxito de la misma dependerá de la elección correcta de estos. Esta arquitectura obedecerá en gran medida a los requisitos de la aplicación a desarrollar y a cómo se dará respuesta a los mismos.

La primera referencia que se tiene de los estilos arquitectónicos data del año 1992 propuesta por los ya mencionados profesores Dewayne Perry y Alexander Wolf quienes definen un estilo arquitectónico como una abstracción de tipos de elementos y aspectos formales a partir de diversas arquitecturas específicas. “Un estilo arquitectónico encapsula decisiones esenciales sobre los elementos arquitectónicos y enfatiza restricciones importantes de los elementos y sus relaciones posibles” (Perry, y otros, 1992).

En un ensayo de 1996 en el que aportan fundamentos para una caracterización formal de las conexiones arquitectónicas, los profesores de la Universidad de Carnegie Mellon Robert Allen y David Garlan asimilan los estilos arquitectónicos a descripciones informales de arquitectura basadas en una colección de componentes computacionales, junto a una colección de conectores que describen las interacciones entre los componentes.

Por otro lado, Mark Klein y Rick Kazman de dicha Universidad en 1999 proponen una definición donde afirman que son artefactos de ingeniería importantes porque definen clases de diseño junto con las propiedades conocidas asociadas a ellos. Ofrecen evidencia basada en la experiencia sobre la forma en que se ha utilizado históricamente cada clase, junto con razonamiento cualitativo para explicar por qué cada clase tiene esas propiedades específicas (Klein, 1999).

Todas las definiciones antes descritas, aunque hasta cierto punto heterogéneas, cuentan con elementos comunes de los cuales se pueden concluir que los estilos arquitectónicos expresan la arquitectura en el sentido más formal y teórico, enlazan los componentes, conectores, configuraciones y restricciones de un sistema o subsistema. Generalmente se les utiliza combinados; cada capa o componente puede ser internamente de un estilo diferente al de la totalidad; muchos estilos se encuentran ligados a dominios específicos, o a líneas de producto particulares.

Los estilos arquitectónicos más utilizados en la actualidad se encuentran divididos en función de algunos criterios en clases o grupos, entre estos se encuentran:

Estilos de Flujo de Datos	Tubería y Filtros
Estilos Centrados en Datos	Arquitecturas de Pizarra o Repositorio
Estilos de Llamada y Retorno	Modelo-Vista-Controlador (MVC) Arquitecturas en Capas Arquitecturas Orientadas a Objetos Arquitecturas Basadas en Componentes
Estilos Derivados	C2 GenVoca REST
Estilos de Código Móvil	Arquitectura de Máquinas Virtuales
Estilos heterogéneos	Sistemas de control de procesos Arquitecturas Basadas en Atributos

Estilos Peer-to-Peer	Arquitecturas Basadas en Eventos Arquitecturas Orientadas a Servicios (SOA) Arquitecturas Basadas en Recursos

Tabla 1- Estilos Arquitectónicos. (Elaboración propia)

A continuación se caracterizan algunos de éstos estilos por la importancia que tienen en la solución del problema, exponiendo además las ventajas y desventajas de cada una de las arquitecturas que encierran dichos estilos.

1.4.1-Estilos Centrados en Datos:

Entre las propiedades más notorias de este estilo arquitectónico se encuentra la de crear persistencia e integridad de los datos almacenados. Este además se estima apropiado para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Sub-estilos característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra.

- Arquitecturas de Pizarra o Repositorio

  Existen dos tipos de componentes pertenecientes a esta arquitectura, un componente que realiza el almacenaje y persistencia de los datos y otros componentes que interactúan con dichos datos. En base a esta distinción se han definidos dos sub-categorías principales del estilo:

  - Si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, el repositorio puede ser una base de datos tradicional (implícitamente no cliente-servidor).

  - Si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control.

1.4.2-Estilos de Llamada y Retorno

Este estilo es interesante en los sistemas que se centran en la interacción de un usuario con el propio sistema. Podemos dividir la arquitectura en dos partes, la primera representa la interfaz del usuario con el que este realiza la llamada al sistema, la segunda contiene la lógica de negocio que se realiza tras la correspondiente llamada del usuario.

Según lo expuesto por el profesor Carlos Reynoso en un estudio llamado Estilos y Patrones en la Estrategia de Arquitectura de Microsoft, esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas de gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.

- Modelo-Vista-Controlador (MVC)

  Separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. MVC se ve frecuentemente en aplicaciones web, donde la vista generalmente es la página HTML y el código que provee de datos dinámicos a la página; el modelo es el Sistema de Gestión de Base de Datos (SGBD) y la Lógica de negocio; y el controlador es el responsable de comunicar el modelo y la vista, respondiendo a peticiones en ambos sentidos.

  Modelo: Es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos.

  Vista: Presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario. Maneja la presentación visual de los datos representados por el Modelo.

  Controlador: Responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

  Entre las ventajas que se exponen de la utilización de este estilo se encuentran: Separación entre interfaz, lógica de negocio y de presentación. Evita poner el código indebido en la

capa impropia. La soportabilidad de múltiples vistas, dada por la independencia que tiene el modelo con respecto a la vista permitiendo entonces que la interfaz de usuario pueda mostrar múltiples vistas de los mismos datos concurrentemente. Amplia adaptación al cambio, ideal para usuarios que puedan preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo.

Si bien este tipo de arquitecturas ofrecen ventajas bien tentativas no se pueden dejar de mencionar los aspectos negativos que trae consigo la utilización de estas como el aumento de la complejidad de la solución como consecuencia de nuevos niveles de no direccionamiento que introduce el estilo y los costos de actualizaciones frecuentes: desacoplar el modelo de la vista no significa que los desarrolladores del modelo puedan ignorar la naturaleza de las vistas.

- Arquitecturas en Capas

Este es un estilo estructurado jerárquicamente por capas que suelen ser entidades complejas, compuestas de varios paquetes o subsistemas, donde cada una proporciona servicios a la inmediatamente superior y actúan sobre los operadores de la inmediatamente inferior.

Una arquitectura de aplicaciones en capas está basada en componentes y su meta es unificar las aplicaciones para los ordenadores, las aplicaciones cliente/servidor y las aplicaciones basadas en la Web, lo cual es posible para aplicaciones de cualquier tamaño.

Este tipo de arquitectura proporciona una amplia reutilización de capas, admite muy naturalmente optimizaciones y refinamientos, facilita la estandarización, permite la construcción de sistemas débilmente acoplados, las dependencias se circunscriben entre las mismas capas por lo que resulta más fácil sustituir la implementación de una capa sin afectar al resto del sistema.

Las características anteriores constituyen los aspectos positivos de esta arquitectura, a continuación de exponen algunas desventajas: puede ser difícil definir que componentes ubicar en cada una de las capas. En ocasiones no se logra la contención del cambio en las mismas capas y se requiere una cascada de cambios, lo que acarrea una pérdida de eficiencia. El trabajo innecesario por parte de capas más internas o redundante entre varias de estas.

- Arquitecturas Orientadas a Objetos

Estilo conocido también por Arquitectura Basada en Objetos, Abstracción de Datos y Organización Orientada a Objetos.

Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. Un objeto está compuesto por estado, comportamiento e identidad. Estos componentes se basan en los principios de la Programación Orientada a Objetos entiéndase por: encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación; los objetos y sus interacciones el centro de las incumbencias en el diseño de la arquitectura.

Este estilo se caracteriza porque se puede modificar la implementación de un objeto sin afectar a sus clientes, se hace posible descomponer problemas en colecciones de agentes en interacción y donde un objeto es ante todo una entidad reutilizable en el entorno de desarrollo.

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 y que cuando se modifica un objeto se deben modificar también todos los objetos que lo invocan.

1.5 Patrones Arquitectónicos

En el continuo progreso de proyectos de software se encuentran problemas que se hacen frecuentes en comunidades de desarrollo, se hizo necesario entonces el hallazgo de una solución conjunta que permitiera apocoparlos, así surgieron los patrones arquitectónicos, que han resultado disímiles desde su creación. Por tanto, un patrón no es más que la descripción detallada de una solución adecuada a un problema concreto.

Existen varias definiciones de patrón arquitectónico entre ellas la del profesor Buschmann que define un patrón de software como aquel que "describe un problema recurrente que ocurre en un contexto específico en el diseño de sistemas y presenta un esquema de solución genérica y probada. La especificación de la solución incluye la descripción de los componentes, sus responsabilidades y relaciones y la forma en que colaboran entre sí" (Buschmann y otros, 1996).

Es interesante referirse también a la definición de patrones que se da en el libro Patterns of Enterprise Application Architecture (Patrones de Arquitectura de Aplicaciones Empresariales), del cual se cita: "Cada patrón describe un problema que ocurre una y otra vez en nuestro medio ambiente y, a continuación, se describe el núcleo de la solución a ese problema, de tal manera que puede utilizar esta solución un millón de veces más, sin hacerlo de la misma manera dos veces (Fowler, 2002)."

En líneas generales, un patrón sigue el siguiente esquema:

- Nombre
  - Define un vocabulario de diseño
  - Facilita abstracción
- Problema
  - Describe cuando aplicar el patrón
  - Conjunto de fuerzas: objetivos y restricciones
  - Prerrequisitos

- Solución
    - Elementos que constituyen el diseño (*template*)
    - Forma canónica para resolver fuerzas
- Consecuencias
    - Resultados, extensiones y compensaciones

Existen dos corrientes básicas en términos de patrones de arquitectura:

Patrones Orientados a Arquitectura de Software (POSA): Estos patrones agrupan una vista general de distintos niveles de abstracción en concepción de patrones, entre ellos un grupo orientado a un alto nivel de arquitectura. Los patrones aquí definidos son también tratados indistintamente como estilos arquitectónicos.

Ejemplos de patrones POSA: Diseño, Tuberías y Filtros, Pizarra, Broker, Modelo-Vista-Controlador, Presentación-Abstracción-Control, Microkernel.

Patrones de Arquitectura Empresarial (PEEA): Se centran en unificar las partes de una aplicación empresarial mediante formas comunes. Se basan en experiencias recopiladas sobre formas de modelar partes de un sistema, fundamentalmente mediante capas y la forma en que estas internamente se organizan.

Ejemplos de patrones PEAA: Transacciones Script, Modelo de Dominio, Tabla Módulo, Controlador de Páginas, Lista de Plantillas, Vista en Dos Pasos.

Son disímiles las clases de patrones existentes: de análisis, de arquitectura (divididos en progresivamente estructurales, sistemas distribuidos, sistemas interactivos, sistemas adaptables), de diseño (conductuales, creacionales, estructurales), de organización o proceso, de programación y los llamados idiomas, entre otros. Cada autor que escribe sobre el asunto agrega una clase diferente, y los estándares en vigencia no hacen ningún esfuerzo para poner un límite a la proliferación de variedades y ejemplares, a continuación se relacionan:

Tipo de Patrón	Comentario	Problemas	Soluciones	Fase de Desarrollo
Patrones de Arquitectura	Relacionados a la interacción de objetos dentro o entre niveles arquitectónicos	Problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, performance, modularidad, acoplamiento	Patrones de llamadas entre objetos (similar a los patrones de diseño), decisiones y criterios arquitectónicos, empaquetado de funcionalidad	Diseño inicial
Patrones de Diseño	Conceptos de ciencia de computación en general, independiente de aplicación	Claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, etc.	Comportamiento de factoría, Clase-Responsabilidad-Contrato (CRC)	Diseño detallado
Patrones de Análisis	Usualmente específicos de aplicación o industria	Modelado del dominio, completitud, integración y equilibrio de objetivos múltiples, planeamiento para capacidades adicionales comunes.	Modelos de dominio, conocimiento sobre lo que habrá de incluirse (p. ej. logging & reinicio)	Análisis
Patrones de Procesos o Organización	Desarrollo o procesos de administración de proyectos, o técnicas, o estructuras de organización	Productividad, comunicación efectiva y eficiente	Armado de equipo, ciclo de vida del software, asignación de roles, prescripciones de comunicación	Planeamiento
Idiomas	Estándares de codificación y proyecto	Operaciones comunes bien conocidas en un nuevo ambiente, o a través de un grupo. Legibilidad, predictibilidad.	Sumamente específicos de un lenguaje, plataforma o ambiente	Implementación, Mantenimiento, Despliegue

Tabla 2- Relación de Patrones. (Reynoso, 2004)

El libro “Patrones de diseño, elementos reutilizables del software orientado a objeto” plantea que “un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular.” (Gamma, et al., 2005):

Creacionales	Estructurales	Comportamiento
Fábrica abstracta	Adaptador	Cadena de responsabilidad
Constructor virtual	Puente	Orden
Método de fabricación	Objeto compuesto	Intérprete
Prototipo	Envoltorio	Iterador
Instancia única	Fachada	Mediador
	Peso ligero	Recuerdo
	Proxy	Observador
		Estado
		Estrategia
		Método plantilla
		Visitante

Tabla 3: Patrones de Diseño (Elaboración propia).

1.6- Lenguajes de Modelado

Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software según plantea Rumbaugh. El UML constituye un estándar para construir modelos orientados a objetos. Es un lenguaje de modelado

independiente del lenguaje de programación que se utilice así como de la plataforma en la que se desarrolla la aplicación. UML documenta toda la información estática y dinámica de un sistema, su distribución, estructura y comportamiento, de ahí que logre que los implicados tengan un entendimiento común independiente del código de la aplicación que se desee implementar. Este lenguaje está compuesto por tres elementos fundamentales, los elementos, los diagramas y las relaciones. UML ha sido seleccionado como lenguaje de modelado del portal por ser un lenguaje sencillo, visual, fácil de utilizar y comprender, que omite detalles específicos y permite la comprensión necesitada para el desarrollo de un sistema en concreto. En otras palabras simplifica la complejidad real, permite la reutilización de soluciones o de modelos, permite descubrir fallas y ahorra tiempo de desarrollo.

1.7- Herramientas de Ingeniería de Software Asistidas por Computadora (CASE)

Las herramientas CASE 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. El uso de estas proporciona un incremento en la velocidad de desarrollo y calidad del sistema pues ayudan en la documentación, la fiabilidad, utilidad y el rendimiento. Estas herramientas proveen muchos beneficios en todas las etapas del proceso de desarrollo de software, verificando el uso de todos los elementos en el sistema diseñado. Además automatizan el dibujo de diagramas, permiten a los analistas tener más tiempo para el análisis y diseño, ayudan en la creación de relaciones en la BD y hacen que el trabajo de diseño del software sea más fácil y agradable.

Rational Rose Enterprise Edition

Es comercializada por los desarrolladores de UML y líder en el mundo de modelación visual de sistemas que permite especificar, analizar y diseñar el sistema antes de codificarlo, sin embargo posee limitantes que la hacen débil en comparación a otras herramientas como Visual Paradigm para UML con las mismas facilidades de modelado, estas debilidades radican en la dependencia de la plataforma Windows y la integración solo con herramientas que estén en el mismo grupo de software propietario.

Tiene características como:

- Herramienta propietaria, perteneciente a la familia Rational Rose.

- Madura, bien establecida y con gran aceptación en el mercado.
- Incluye una Modelación Añadida de la Web que proporciona visualización, modelación y herramientas para el desarrollo de aplicaciones Web.
- Ofrece habilidades de análisis de calidad de código y generación del código, con capacidades de sincronización, así como manejo más granular y uso de modelos.

Visual Paradigm para UML

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. La herramienta también proporciona abundantes tutoriales, demostraciones interactivas y proyectos que utilicen UML y paradigma orientado a objetos. Soporta capacidades de ingeniería inversa y directa y brinda facilidad al usuario a partir de que posee una interfaz amigable y fácil de utilizar. Entre sus principales características se encuentran:

- Licencia gratuita y comercial.
- Producto de calidad.
- Soporta aplicaciones web.
- Fácil de instalar y actualizar.
- Compatibilidad entre ediciones.
- Es multiplataforma y muy útil para la generación de código fuente en PHP.
- Soporta a miles de usuarios trabajando sobre el mismo proyecto y permite que cada uno vea los cambios realizados en tiempo real.
- Genera la documentación del proyecto automáticamente tanto en formato PDF como Web.
- Permite el control de versiones.
- Ofrece entorno de creación de diagramas para UML 2.0, 2.1.
- Disponibilidad de integrarse en los principales IDEs de desarrollo.

1.8 - Lenguajes de Programación

Un lenguaje de programación es una construcción mental del ser humano que permite crear programas y software. Está constituido por un grupo de reglas gramaticales, símbolos utilizables, términos con sentido único y una regla principal que resume las demás. Los lenguajes pueden ser de alto o bajo nivel. En los de bajo nivel las instrucciones son simples y cercanas al funcionamiento de la máquina, por ejemplo el código máquina y el ensamblador. En los lenguajes de alto nivel hay un alto grado de abstracción y el lenguaje es más próximo al de los humanos.

PHP

PHP es un acrónimo recursivo que significa Hypertext Pre-processor. (Inicialmente PHP Tools o Personal Home Page Tools) Diseñado originalmente para la creación de páginas web dinámicas, es multiparadigma, multiplataforma, de propósito general ampliamente difundido que permite la creación de aplicaciones con interfaz gráfica, conexión a servidores de base de datos como Oracle, MySQL, Postgres y puede ser ejecutado en sistemas Unix, Windows, Linux y Mac OS X., principalmente en interpretación del lado del servidor (server-side scripting) aunque puede ser incrustado dentro de código HTML. Es uno de los lenguajes de programación más populares por la gran fluidez y rapidez de sus scripts.

El programador que trabaje con PHP puede aplicar en su trabajo cualquier técnica de programación o desarrollo que le permita escribir código ordenado, estructurado y manejable sin necesidad de regirse estrictamente por una metodología. Un ejemplo de esto son los desarrollos que en PHP se han hecho del patrón de diseño Modelo Vista Controlador (MVC), que permiten separar el tratamiento y acceso a los datos, la lógica de control y la interfaz de usuario en tres componentes independientes.

Las ventajas que se encuentran en la utilización de este lenguaje son:

- Completamente orientado al desarrollo de aplicaciones web dinámicas con acceso a información almacenada en una Base de Datos.
- El código fuente escrito en PHP es invisible al navegador y al cliente lo que hace que la programación en PHP sea segura y confiable.
- Interfaces para una gran cantidad de sistemas de Base de Datos diferentes.

- Posee una amplia documentación en su página oficial, entre la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite aplicar técnicas de programación orientada a objetos.
- Biblioteca nativa de funciones sumamente amplia e incluida.
- No requiere definición de tipos de variables aunque sus variables se pueden evaluar también por el tipo que estén manejando en tiempo de ejecución.
- Tiene manejo de excepciones desde PHP5.
- Mantiene un bajo consumo de recursos de máquina.
- Posee gran seguridad, muy poca probabilidad de corromper los datos.
- Permite embeber sus pequeños fragmentos de código dentro de la página HTML.
- Permite leer y manipular datos desde diversas fuentes, incluyendo datos que pueden ingresar los usuarios desde formularios HTML.

Entre tantos beneficios la única limitación que se puede encontrar en las bibliografías es que la ofuscación de código es la única forma de ocultar las fuentes.

Java

Es uno de los lenguajes más usados para el desarrollo de software. Este ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de lenguajes, como los punteros del lenguaje de programación C++. Dentro de las características más importantes de este se puede mencionar:

- Orientado a Objetos: Como la mayoría de los lenguajes, este cumple con el paradigma de la orientación a objetos. Posee una arquitectura neutral: La compilación de los software hechos en java es completamente independiente de la arquitectura de la maquina donde se ejecute. Para java lo único que es verdaderamente dependiente del sistema es la Máquina Virtual de Java (JVM) y las

librerías fundamentales que sean usadas. La neutralidad de la arquitectura mide en gran medida el grado de portabilidad de este lenguaje.

- Interpretado y compilado a la vez: Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina (bytecodes), semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se haya instalado la maquina virtual de java.

- Indiferente a la arquitectura: Soporta aplicaciones para ser ejecutadas en diferentes tipos de redes, desde Unix a Windows NT, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. El compilador de Java genera bytecodes: un formato intermedio indiferente a la arquitectura, diseñado para transportar el código eficientemente a múltiples plataformas hardware y software. El resto de problemas los soluciona el intérprete de Java.

- Curva de aprendizaje: Es lenguaje de programación un poco más complejo que C#, de ahí que a los desarrolladores experimentados en C++ les sea más sencillo emigrar a java

- Portable: La indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas.

- Multihebra: Java soporta sincronización de múltiples hilos de ejecución (multithreading) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas.

- Dinámico: El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.

- Seguro: Se elimina el uso de los punteros con el objetivo de eliminar el uso indebido de recursos en memoria. Además la maquina virtual se encarga de verificar que el software no posea

fragmentos de código ilegal (código que falsea punteros, viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto) antes de ejecutarlo.

JavaScript

Javascript es un lenguaje que puede incluirse en cualquier documento. Es compatible con HTML en el navegador del cliente, ya sea PHP, Active Server Pages, ASP, JSP y SVG. JavaScript no requiere de compilación, es decir, es interpretado. Ha tenido influencias de múltiples lenguajes (Perl, Phyton) y fue diseñado con una sintaxis similar a la de Java. La principal diferencia radica en que Java es un lenguaje orientado a objetos y JavaScript está basado en prototipos. Es compatible con la mayoría de los navegadores como Netscape, Internet Explorer, Mozilla Firefox, entre otros.

Para que interactúe con una página web evitando incompatibilidades, el World Wide Web Consortium (W3C) diseñó el estándar DOM (Modelo de Objetos del Documento). Finalmente se debe señalar que este lenguaje de scripting es seguro y fiable.

1.9- Sistema Gestor de Bases de Datos (SGBD)

Los SGBD Se han creado para gestionar grandes volúmenes de información, así como simplificar y facilitar el acceso a los datos, haciendo que los tiempos de respuesta a las solicitudes de los usuarios sean muy reducidos y el acceso a la información se pueda realizar de forma concurrente; además los SGBD garantizan una eficiente seguridad en el tratamiento de los datos.

MySQL

Es un sistema de gestión de bases de datos relacional, utiliza el Lenguaje Estructurado de Consultas (SQL por sus siglas en ingles) y entre sus características se destacan:

- Tiene diferentes interfaces de programación de aplicaciones (APIs por sus siglas en ingles) disponibles para lenguajes de programación tales como: C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, y Tcl.
- Proporciona sistemas de almacenamientos transaccionales y no transaccionales.

- Presenta un sistema de privilegios y contraseñas que es muy flexible y seguro, y permite verificación basada en el host. Las contraseñas son seguras porque todo el tráfico de contraseñas está encriptado cuando se conecta con un servidor.
- Los clientes pueden conectar con el servidor MySQL usando sockets de protocolo TCP/IP en cualquier plataforma.
- La interfaz para el conector J MySQL proporciona soporte para clientes Java que usen conexiones JDBC. Estos clientes pueden ejecutarse en Windows o Unix y es un código abierto.
- El tamaño efectivo máximo para las bases de datos en MySQL usualmente lo determinan los límites de tamaño de ficheros del sistema operativo, y no límites internos de MySQL.

Oracle

Es una herramienta cliente/servidor para la gestión de bases de datos considerado de los mejores SGBD. Es un producto vendido a nivel mundial pero es comercial, por lo que necesita de licencias para poder usarlo, así mismo la gran potencia que tiene y su elevado precio hacen que sólo se vea en empresas muy grandes y multinacionales, por norma general. Este SGBD se considera débil en comparación con otros como PostgreSQL o MySQL. Entre sus principales ventajas están:

- Es el motor de Base de Datos relacional más usado a nivel mundial.
- Es multiplataforma.
- Un soporte aceptable.
- Es la base de datos más orientada a Internet.
- Alta seguridad.
- Estabilidad.

Sus principales desventajas son:

- Su elevado precio.
- Las licencias Personal Oracle, son excesivamente caras.

- La necesidad de ajustes, una mala configuración de Oracle puede resultar desesperadamente lento.

PostgreSQL

Es un SGBD relacional orientada a objetos de software libre, publicado bajo la licencia BSD (licencia de software libre permisiva). Este sistema permite la administración y la escalabilidad, ayuda a lograr ahorros considerables en costos de operación conservando todas las características, estabilidad y rendimiento; es extensible, el código fuente está disponible para todos y es posible modificarlo en dependencia de las necesidades de la aplicación. Los requisitos para la instalación son mínimos y posibilita realizar copias de seguridad evitando la pérdida de información.

A continuación se enumeran las principales características:

- Implementación del estándar SQL92/SQL99.
- Soporta distintos tipos de datos.
- Permite la creación de tipos propios.
- Incorpora una estructura de datos array (arreglos).
- Incorpora funciones de diversa índole: manejo de fechas, geométricas, orientadas a operaciones con redes, entre otras.
- Permite la declaración de funciones propias, así como la definición de disparadores.
- Soporta el uso de índices, reglas y vistas.
- Incluye herencia entre tablas.
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.

1.10- Frameworks para el desarrollo de software

Un framework o marcos de trabajo, en el desarrollo de software, es una estructura de soporte mediante la cual los proyectos de software pueden ser organizados y desarrollados. Sin embargo, fuera de las

aplicaciones de la informática, puede ser considerado como el conjunto de procesos y tecnologías usados para resolver un problema complejo. Son diseñados con el intento de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requisitos de software que tratando con los tediosos detalles de bajo nivel y así realizar aplicaciones con una mayor rapidez. En el mundo de las aplicaciones web existen varios frameworks que facilitan y automatizan muchas funcionalidades comunes en el desarrollo de software. Las principales funcionalidades que proveen son, entre otras, el acceso a los datos mediante ORM (Mapeo objeto-relacional), la implementación de un patrón arquitectónico, así como algunos asistentes para el diseño de interfaces de usuario de mayor calidad en menos tiempo. Existen varios marcos de trabajo para PHP como CakePHP, Zend Framework, Symfony, CodeIgniter, Prado, P4A, entre otros.

CakePHP

CakePHP es un marco de desarrollo rápido que provee una arquitectura extensible para desarrollar, mantener y desplegar aplicaciones web. Utilizando conocidos patrones de diseño como MVC. Dentro de la convención del paradigma de configuración, CakePHP reduce los costos de desarrollo y ayuda a los desarrolladores a escribir menos código, sus principales características son:

- Compatible con PHP4 y PHP5.
- CRUD (*Create, Read, Update and Delete*) de la base de datos integrado.
- Localizador de Recurso Uniformes (URL) amigables.
- Sistema de plantillas rápido y flexible.
- Ayuda para AJAX, Javascript, HTML, formularios y más.
- Trabaja en cualquier subdirectorio del sitio.
- Validación integrada.
- Scaffolding (andamio) de las aplicaciones.
- Listas de control de acceso.

- Componentes de seguridad y sesión.
- Se distribuye bajo licencia MIT (Instituto Tecnológico de Massachusetts), es desarrollado por este instituto y es multiplataforma.

Zend Framework

Framework para desarrollo de aplicaciones y servicios web, brinda soluciones para construir sitios modernos, robustos y seguros. Además es Open Source (software distribuido y desarrollado libremente) y trabaja con PHP 5, a diferencia de CakePHP que trabaja con PHP 4 y PHP 5. La principal ventaja de Zend Framework es que es desarrollado por Zend (empresa que respalda comercialmente a PHP). Las principales características de dicho framework son:

- Implementa MVC.
- Cuenta con módulos para manejar archivos PDF, canales RSS, Web Services (Amazon, Flickr, Yahoo), entre otros.
- Para el acceso a base de datos, balancea el ORM con eficiencia y simplicidad.
- Completa documentación y pruebas de alta calidad.
- Soporte avanzado para i18n (internacionalización).
- Robustas clases para autenticación y filtrado de entrada.
- Zend Framework es multiplataforma, desarrollado y mantenido por Zend Technologies y se puede utilizar bajo licencia New BSD License (licencia de software libre permisiva).

Symfony

Symfony es un framework bien completo diseñado para optimizar el desarrollo de las aplicaciones web mediante algunas de sus principales características. 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 compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. Es

compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y Microsoft SQL Server.

“Es uno de los frameworks para PHP más populares entre los usuarios y las empresas, permite que los programadores sean mucho más productivos a la vez que crean código de más calidad y más fácil de mantener. Symfony es maduro, estable, profesional y está muy bien documentado. Utiliza las mejores prácticas y los patrones de diseño más importantes, incorpora muchas de las ideas del RAD (Desarrollo Rápido de Aplicaciones).” (Eguiluz, 2007)

Sus principales características son:

- Fácil de instalar y configurar en la mayoría de plataformas (y con la garantía de que funciona correctamente en los sistemas Windows y *nix estándares).
- Independiente del sistema gestor de bases de datos.
- Implementa Front Controller (Controlador Frontal) como patrón derivado del MVC.
- Acceso a datos a través de ORM.
- Incluye soporte para AJAX.
- Provee varias ayudas para la creación de interfaces.
- Soporta la instalación de extensiones o plugins para añadir nuevas funcionalidades.
- URLs amigables y configurables a través de un poderoso sistema de enrutamiento.
- 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 las mejores prácticas y patrones de diseño para la web.

- Preparado para aplicaciones empresariales y se adapta a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Código fácil de leer que incluye comentarios de phpDocumentor y que permite un mantenimiento muy sencillo.
- Implementa MVC
- Fácil de extender, lo que permite su integración con las librerías de otros fabricantes.
- Symfony es multiplataforma, desarrollado por Sensio Labs, Inc y liberado por una licencia de tipo MIT para software libre compatible con la GPL (GNU General Public License).

1.11- Entornos Integrados de Desarrollo (IDE).

Un entorno de desarrollo o IDE (Integrated Development Environment), es una aplicación o conjunto de estas en las que se combinan las tecnologías a utilizar para el desarrollo del software, hace el trabajo más sencillo, permite versatilidad para depurar los programas ya que tienen depuradores avanzados, consienten entre otras tareas: escribir el código, compilarlo o ejecutarlo, detectar errores, gestionar versiones y crear instaladores.

Zend Studio

Zend Studio es un editor de texto para páginas PHP que proporciona un buen número de ayudas desde la creación y gestión de proyectos hasta la depuración del código, esta última es una herramienta muy interesante que permite ejecutar páginas y conocer en todo momento el contenido de las variables de la aplicación y las variables del entorno como las cookies. Existen dos ediciones: Standard y Professional. ZendStudio divide sus funcionalidades en dos partes: la del cliente y la del servidor las cuales se instalan por separado. La parte del cliente contiene la interfaz de edición y además permite hacer depuraciones simples de scripts. Proporciona la visualización, edición y la capacidad de ejecución para bases de datos populares que poseen lenguaje SQL, incluyendo MySQL, Oracle, IBM DB2 y Cloudscape, Microsoft SQL Server, SQLite y PostgreSQL; es desarrollado el despliegue de código rápido y el control Windows. Esta herramienta permite agiliza el desarrollo web y simplificar proyectos complejos, posee integración con programas controladores de versiones como el subversión y con servicios de FTP.

Ventajas del trabajo con el Zend Studio

- Agiliza el trabajo.
- Poco consumo de recursos de la máquina para su ejecución.
- Posee auto completamiento de código.
- Cuenta con un buen Depurador.
- Posee infinitas opciones que permiten el desarrollo profesional de aplicaciones.

Eclipse

Es un completo IDE de programación distribuido y desarrollado libremente (Open Source) independiente de una plataforma de desarrollo. Esta considerado uno de los más poderosos editores para lenguajes como JAVA, C y PHP. Tiene como una de sus principales ventajas que brinda la posibilidad de realizar una aplicación con mayor calidad. Este entorno es uno de los más populares en el mundo y además muy usado por la facilidad de integración con diferentes herramientas.

Ventajas de Eclipse:

- Buen soporte de refactorización
- Fluidez de la interfaz
- Integración con otros lenguajes de programación
- Excelente tiempo a la hora de compilar/ejecutar.
- Completamente funcional para aplicaciones PHP.
- Constituido por complementos (plug-ins) que interactúan y permiten una funcionalidad más fácil.

1.12- Metodologías de desarrollo

Las metodologías de desarrollo de software surgen como consecuencia de la necesidad de establecer procesos de organización que guíen el trabajo de creación de software. En un proyecto de desarrollo de

software la metodología define Quién debe hacer Qué, Cuándo y Cómo debe hacerlo. No existe una metodología de software universal. A continuación se hace referencia a tres metodologías de las más utilizadas en la actualidad.

Proceso Unificado de Rational (RUP).

RUP se incluye dentro de las metodologías llamadas tradicionales o robustas, es una de las más importantes para alcanzar un grado de certificación en el desarrollo del software, por el uso de artefactos en cada ciclo de iteración. Es una metodología orientada a objetos que utiliza a UML como lenguaje de modelado que se caracteriza por ser:

- Dirigida por Casos de Uso
- Centrada en la Arquitectura
- Iterativa e Incremental

RUP se basa en la definición de diferentes roles y flujos de trabajos, donde se desarrollan las actividades que arrojan los diferentes artefactos que deben realizar los trabajadores. Estos trabajadores se encuentran dentro de cada rol en las diferentes etapas que conforman el ciclo de vida del software.

Estos flujos son (Jacobson, et al., 2000):

- Modelación del negocio: Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
- Requerimientos: Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
- Análisis y diseño: Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.
- Implementación: Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.
- Prueba (Testeo): Busca los defectos a los largo del ciclo de vida.

- Instalación: Produce release del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.

- Administración del proyecto: Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.

- Administración de configuración y cambios: Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, etc.

- Ambiente: Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.

Así mismo se encuentra dividido en cuatro fases

- Conceptualización o Inicio: Se describe el negocio y se delimita el proyecto describiendo sus alcances con la identificación de los casos de uso del sistema.

- Elaboración: Se define la arquitectura del sistema y se obtiene una aplicación ejecutable que responde a los casos de uso que la comprometen. A pesar de que se desarrolla a profundidad una parte del sistema, las decisiones sobre la arquitectura se hacen sobre la base de la comprensión del sistema completo y los requerimientos (funcionales y no funcionales) identificados de acuerdo al alcance definido.

- Construcción: Se obtiene un producto listo para su utilización que está documentado y tiene un manual de usuario. Se obtiene 1 o varios release del producto que han pasado las pruebas. Se ponen estos release a consideración de un subconjunto de usuarios.

- Transición: El release ya está listo para su instalación en las condiciones reales. Puede implicar reparación de errores.

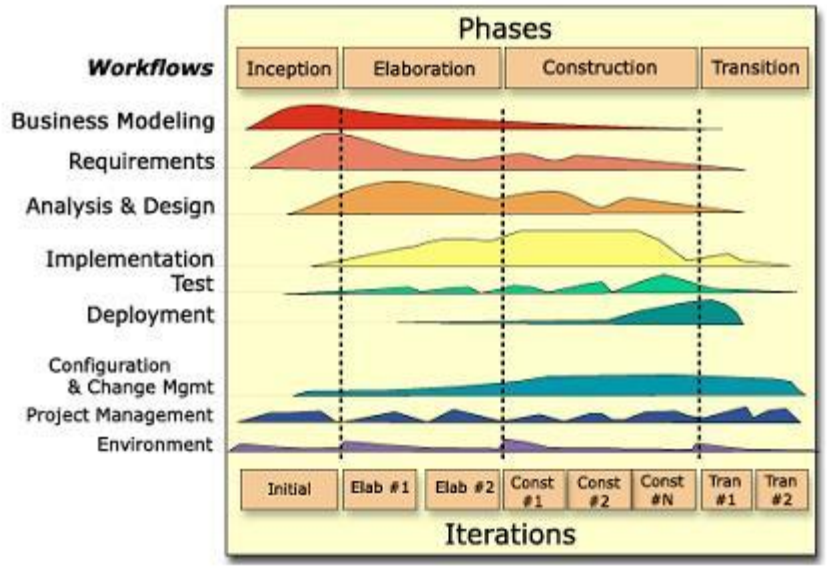

Figura 1: Fases de desarrollo de Software

Xp

Extreme Programming: Se caracteriza por poseer bajo riesgo, por ser flexible, formada por valores, principios y prácticas. Centrada en potenciar las relaciones entre cliente y equipo de desarrollo como clave para el éxito en desarrollo de software, es adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Esta metodología está basada en:

- Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de manera tal que se pueda prevenir posibles errores que en el futuro pudieran ocurrir.
- Refabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- Programación en pares: Una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

Para un desarrollo exitoso con esta metodología es necesario que exista (Sánchez, 2004):

- La comunicación, entre los clientes y los desarrolladores.
- La simplicidad, al desarrollar y codificar los módulos del sistema.
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.

Scrum.

El desarrollo de software se realiza mediante iteraciones, denominadas sprints o carreras cortas, con una duración de 30 días. Antes de que comience una carrera se define la funcionalidad requerida para esa carrera y entonces se deja al equipo para que la entregue. El punto es estabilizar los requisitos durante la carrera. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. No es propiamente un método o metodología de desarrollo, e implantarlo como tal resulta insuficiente, Scrum define métodos de gestión y control para complementar la aplicación de otros métodos ágiles como XP o alguna otra metodología de desarrollo.

1.13- Servidor Web

Un servidor web es un programa que se encarga de transferir páginas web y de almacenar documentos HTML. Está diseñado también para soportar y trabajar con imágenes, archivos de textos y demás elementos web que contengan datos, para enviar esta información a los usuarios que la soliciten por la red. Los servidores web implementan el protocolo HTTP lo que hace posible todo este intercambio. El término de servidor web también se emplea para referirse al ordenador que ejecuta el programa.

Microsoft Internet Information Services

Es un servidor web propietario cuyos servicios se limitan a los ordenadores que trabajan con Windows. Estos servicios facilitan las funciones y herramientas necesarias para administrar de forma sencilla un servidor web seguro. Además se basa en varios módulos que le dan la posibilidad de poder procesar distintos tipos de páginas

Apache

Este servidor HTTP es uno de los más utilizados y populares pues es compatible con múltiples sistemas operativos y es una tecnología libre de código abierto. Las capacidades de este servidor pueden ser ampliadas incorporándole nuevos módulos, pues su diseño modular es altamente configurable. Esto le ha permitido incorporar nuevas extensiones entre las que se destaca PHP. Se debe destacar que algunos de los más grandes sitios web del mundo se ejecutan sobre Apache y otros utilizan versiones modificadas del mismo, por su robustez, flexibilidad, estabilidad y eficiencia.

1.14- Conclusiones Parciales.

A lo largo del Capítulo 1 se han definido los principales conceptos relacionados con la AS, que servirán de guía al lector para adentrarse y entender el tema en cuestión. Se realizó un estudio y conceptualización de las arquitecturas o corrientes arquitectónicas más usadas actualmente en el ámbito internacional. Se identificaron y definieron los procesos que se desarrollan dentro del Grupo Calidad de la Facultad 9, exponiendo la necesidad de la automatización de estos. Este capítulo constituye la base fundamental para el inicio de la construcción de la AS a proponer.

Capítulo 2: Descripción de las tecnologías a utilizar.

2-Introducción

Luego de haber realizado en el capitulo anterior un estudio de las principales tecnologías se está en condiciones de seleccionar las metodologías, herramientas y lenguajes a utilizar en el desarrollo del Portal del Grupo Calidad de la Facultad 9. Definiendo además los Estilos y Patrones Arquitectónicos asociados, que se ajustan a la solución adecuada.

2.1- Selección de Metodologías, Herramientas y Lenguajes.

2.1.1- Metodología de desarrollo.

Como parte de las metodologías sugeridas en el capítulo 1 del trabajo científico se definió la metodología RUP. En esta se agrupan las mejores prácticas de metodologías anteriores y las modernas, que se adaptan al contexto y necesidad de cada aplicación. En el libro El Proceso Unificado de Desarrollo de Software los autores plantean que "el Proceso Unificado es más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto." (Jacobson, et al., 2000). A esto se le suma la comparación de los elementos necesarios para guiar el proceso (roles, artefactos, actividades y flujo de trabajo) y que es conocida de antemano por los demás implicados en el desarrollo del Portal de Calidad. El uso de esta metodología asegura que se produzca desde las primeras fases de desarrollo del software, un producto de calidad que cumpla con las características de funcionalidad, usabilidad y fiabilidad. Entre las ventajas que se pueden encontrar de RUP están:

- Es un marco del proyecto que describe una clase de los procesos que son iterativos e incrementales.
- Define las actividades y los artefactos que se necesitan para construir un proceso individual propio.

- Es el proceso de desarrollo más general de los existentes actualmente (Ver epígrafe 1.13).
- Los procesos de RUP estiman tareas y horario del plan midiendo la velocidad de iteraciones concerniente a sus estimaciones originales. Las iteraciones tempranas de proyectos conducidos RUP se enfocan fuertemente sobre arquitectura del software.
- Esta metodología enfatiza en los requisitos y el diseño de los proyectos.
- La ventaja principal de RUP es que se basa todo en las mejores prácticas que se han intentado y se han probado en el campo.

2.1.2- Lenguaje de Modelado

Con la intención de complementar la decisión de trabajar con la metodología RUP, se escoge como lenguaje de modelado el UML. Esta elección es consecuencia de la consideración de varios expertos quienes plantean que la combinación de RUP y UML constituye la metodología más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Este lenguaje divide cada proyecto en un número de diagramas que representan las distintas vistas del proyecto y juntos representan la arquitectura del mismo, además, permite describir un sistema en diferentes niveles de abstracción. Las ventajas más señaladas de UML son:

- Diseño y documentación.
- Código reutilizable.
- Descubrimiento de fallas.
- Ahorro de tiempo en el desarrollo del software.
- Mucho más fáciles las modificaciones.
- Más fácil comunicación entre programadores.

2.1.3- IDE

El entorno integrado de desarrollo con que se trabajará es el Zend Studio. Una de las ventajas principales con la que cuenta este IDE es la de permitir agilizar el desarrollo web y simplificar proyectos complejos.

Fue creado para trabajar con PHP que es el lenguaje de programación escogido para el desarrollo del Portal. Entre las características que posee Zend Studio y que resultan definitorias para su elección están:

- Es multiplataforma.
- Consume pocos recursos de la máquina.
- Cuenta con un buen depurador.
- Posee auto completamiento de código.
- Tiene optimizadores de código
- Se beneficia de herramientas de base de datos
- Contiene una ayuda contextual con todas las librerías de funciones del lenguaje que asiste.

2.1.4- Herramienta CASE

Por las características y ventajas planteadas en el epígrafe 1.8, además de su perfecto acople con la metodología y el lenguaje de modelado que se emplearan, se determinó como herramientas CASE a utilizar en el desarrollo de la investigación, el Visual Paradigm para UML. Es ideal para la creación de un ambiente de software con calidad y perfección como el que se desea para el desarrollo del Portal del Grupo Calidad de la Facultad 9. Es entre las herramientas CASE multiplataforma la que realiza con mayor calidad el modelado de los artefactos generados en cada uno de los flujos de trabajo definidos por la metodología de desarrollo RUP.

Visual Paradigm para UML soporta 13 tipos de diagramas UML, muchos de los cuales son imprescindibles para el desarrollo de la investigación como son diagramas de clases, de casos de uso, de secuencia, de comunicación, de máquina de estados, de actividad, de componentes, de implementación, de paquetes, de objetos, de estructura compuesta, de tiempos, de interacciones.

2.1.5- Lenguaje de Programación

El lenguaje de programación que se propone específicamente es PHP5 para el lado del servidor. Este lenguaje es bien conocido por los implementadores del Portal. Además es un software libre relativamente

rápido, que proporciona costos menores, servidores más baratos y reducción de tiempo en la resolución de un problema detectado.

PHP5 es muy cambiante y potente, capaz de ajustarse a las necesidades de un sistema, que bien puede ser grande y complejo como pequeño. Su diseño está encaminado a facilitar la creación de página web, pero es posible desarrollar aplicaciones con una interfaz gráfica para el usuario. Puede además interactuar con los servidores web más populares y conectarse fácilmente a servidores de bases de datos.

Estas características sumadas a las expuestas en el epígrafe 1.9 sustentan la elección realizada.

Para el lado del cliente el lenguaje que se propone es JavaScript. Este lenguaje será utilizado en la realización de la aplicación para crear páginas dinámicas con efectos sobre los componentes, para crear animaciones y otras funcionalidades. En conjunto con dicho lenguaje se estará utilizando el lenguaje de marcación de hipertexto (HTML) que es el lenguaje autorizado para crear documentos en la World Wide Web y define la estructura de un documento web usando etiquetas y atributos.

2.1.6- SGBD

De los SGBD estudiados se escoge como el más adecuado PostgreSQL. Las peculiaridades siguientes conjuntamente con las mencionadas en el epígrafe 1.10 avalan esta elección

PostgreSQL está soportado por una importante comunidad de profesionales y entusiastas que proporciona una mejora diaria del software en cuestión. El código fuente está disponible para todos sin costo alguno, constituyendo uno de los SGBD libres más avanzados. Tiene amplia capacidad para almacenar grandes volúmenes de datos y permitir el acceso de diferentes usuarios al mismo tiempo, acreditando permiso a cada uno de ellos, por lo que se dice que tiene buena escalabilidad. Al ser un software libre se puede modificar de acuerdo a las necesidades del Portal y logra un ahorro considerable en costos de operación. Los requisitos para la instalación son mínimos y posibilita realizar copias de seguridad evitando la pérdida de información.

2.1.7- Framework

Luego de escoger un lenguaje de programación como el PHP, se hace necesario un framework que consienta una facilidad de trabajo adecuada, ajustable a cualquier plataforma, compatible con el SGBD PostgreSQL y que permita el desarrollo de un Portal con la mayor rapidez posible. Se selecciona entonces Symfony como un marco de trabajo capaz de dar respuesta a los requerimientos mencionados.

Con la utilización de este marco de trabajo se consigue una gran calidad en el código y en la documentación de los proyectos .Tiene entre sus características la de separar la lógica de negocio, la lógica de servidor y la presentación de la aplicación web, empleando el tradicional patrón arquitectónico MVC. Utiliza además otros patrones de diseño. Symfony es apoyado por una gran comunidad de usuarios. Por su versatilidad y sus altas posibilidades de configuración además de su enorme conjunto de herramientas y utilidades, es un framework adecuado para cualquier proyecto de aplicaciones Web desarrollado en PHP.

2.1.8- Selección del Servidor Web

Por su relación con las herramientas ya escogidas, por ser usado para tareas donde el contenido necesita ser puesto a disposición en una forma segura y confiable, entre otras características, se selecciona como servidor web el Apache. Entre las ventajas más comunes de este servidor se pueden mencionar las siguientes:

- Modular
- Código abierto
- Multiplataforma
- Extensible
- Popular
- Es fácil conseguir ayuda y soporte

2.2-Selección de Estilos y Patrones Arquitectónicos.

2.2.1- Estilo de llamadas y Retornos: Arquitectura Orientada a Objetos

Desde el punto de vista de que la arquitectura de la aplicación estará basada en los principios de la Programación Orientada a Objetos (POO), Abstracción, Herencia y Polimorfismo, los principales elementos de la arquitectura estarán centrados hacia los objetos y serán estos las unidades de modelado a partir de las clases que los definen interactuando a través de funciones y métodos.

2.2.2- Estilo de llamadas y Retornos: Modelo Vista Controlador

Este estilo permite tener una organización lógica de los componentes del sistema al dividirlos en tres niveles diferentes: el modelo, la vista y la lógica del negocio. Constituyendo esta su particularidad más ventajosa, pues elimina el complejo trabajo de tener que cambiar la lógica del negocio cada vez que cambie la interfaz de usuario.

Entre sus principales características se encuentran su versatilidad y adaptación al cambio, propiciando que se ajuste a las necesidades del sistema a desarrollar. Siendo su uso vital cuando se desea que el producto cuente con la menor cantidad de código posible y simplicidad del mismo

2.2.3- Patrones arquitectónicos

Dentro de los patrones arquitectónicos a los que se hizo referencia en el epígrafe 1.5 se encontraban los de la corriente arquitectónica de Patrones Orientados a Arquitectura de Software (POSA). En el epígrafe mencionado se expresa que los patrones que se encuentran dentro de esta familia son considerados también como estilos arquitectónicos. En la mayor parte de la bibliografía consultada el término de estilo o patrón arquitectónico se utiliza indistintamente. Su principal diferencia radica fundamentalmente en el nivel de abstracción, el cual se puede observar en la siguiente imagen:

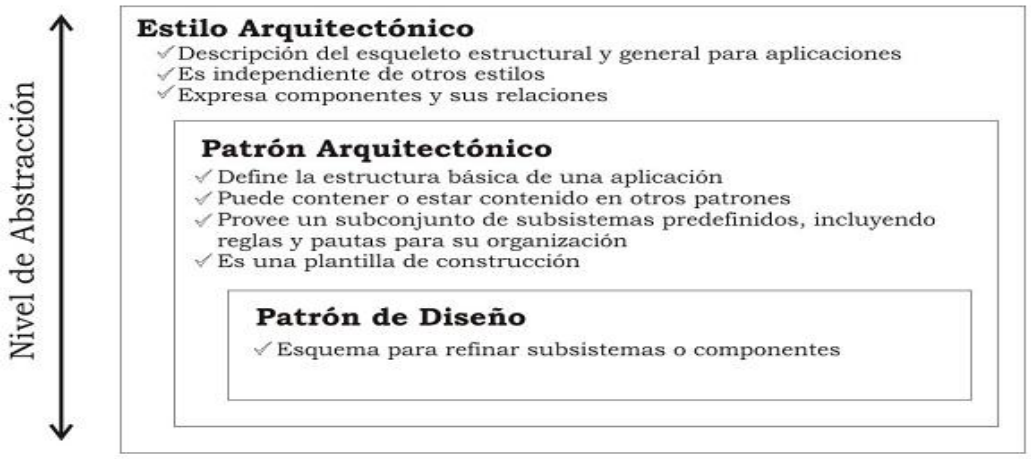

Figura 2: Relación entre Estilo Arquitectónico, Patrón Arquitectónico y Patrón de Diseño. (Cid, 2009).

Por todo lo anterior planteado se selecciona patrón MVC en analogía con el estilo arquitectónico. Este es uno de los patrones empleados por el framework seleccionado para el desarrollo del Portal, lo que hace altamente acertada su elección.

2.2.4- Patrones GRASP

Los Patrones Generales de Software para Asignación de Responsabilidades (GRASP por sus siglas en ingles) son considerados buenas prácticas a seguir en la creación de un software. Estos describen los principios básicos, simples y fundamentales de diseño de objetos para la asignación de responsabilidades. El framewok Symfony implementa cinco de los nueve patrones existentes. Estos son (Rabaix, 2008):

- Creador

  Symfony tiene implementado como parte de su librería la clase Actions, que es la encargada de crear los objetos de las clases que representan las entidades.

- Experto

  Symfony utiliza la librería externa Propel y esta librería su vez Creole, que es el componente encargado de realizar su capa de abstracción en el modelo, encapsula toda la lógica de los datos y son generadas las clases con todas las funcionalidades comunes de las entidades.

- Alta Cohesión

  Symfony permite asignar responsabilidades con una alta cohesión, por ejemplo la clase Actions tiene la responsabilidad de definir las acciones para las plantillas y colabora con otras para realizar diferentes operaciones, instanciar objetos y acceder a las properties, es decir, está formada por diferentes funcionalidades que se encuentran estrechamente relacionadas proporcionando que el software sea flexible frente a grandes cambios.

- Controlador

  Todas las peticiones Web son manejadas por un solo controlador frontal (sfActions), que es el punto de entrada único de toda la aplicación en un entorno determinado. Cuando el controlador frontal recibe una petición, utiliza el sistema de enrutamiento para asociar el nombre de una acción y el nombre de un módulo con la URL entrada por el usuario.

- Bajo Acoplamiento

  La clase Action hereda solamente de sfActions para lograr un bajo acoplamiento de clases.

2.2.5- Patrones GOF

Gang of Four (GoF, La Banda de los Cuatro) es el nombre con el que se conoce comúnmente a los autores del libro Design Patterns, Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, referencia en el campo del diseño orientado a objetos. En este libro se enumeran distintos tipos de patrones de diseño, divididos en Creacionales, Estructurales y de Comportamiento. Symfony implementa algunos de estos patrones.

Creacionales:

- Singleton (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. En el controlador frontal hay una llamada sfContext::getInstance(). En una acción, el método getContext (), es un objeto muy útil que guarda una referencia a todos los objetos del núcleo de Symfony.

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

esté usando. Cuando el framework necesita por ejemplo crear un nuevo objeto para una petición, busca en la definición de la factoría el nombre de la clase que se debe utilizar para esta tarea.

Estructurales:

- Decorator (Envoltorio): Añade funcionalidad a una clase dinámicamente. Esto permite no tener que crear clases sucesivas que hereden de la primera incorporando la nueva funcionalidad, sino otras que la implementan y se asocian a la primera. El framework Symfony tiene un archivo layont.php o plantilla global como también se le conoce que se encarga de almacenar el código HTML que es común a todas las páginas de la aplicación.

- Composite (Objeto compuesto): Permite tratar objetos compuestos como si de uno simple se tratase. Sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol .Esto simplifica el tratamiento de los objetos creados, ya que al poseer todos ellos una interfaz común, se tratan todos de la misma manera.

2.3- Conclusiones Parciales

Este capítulo es resultado de la búsqueda y análisis de la información del estado del arte de todos los aspectos mencionados en el capítulo 1. Se realizó la selección justificada de las tendencias y tecnologías que darán soporte al desarrollo de la propuesta arquitectónica para el Portal del Grupo Calidad de la Facultad 9. Para esta selección se tuvieron en cuenta aspectos significativos como la orientación al desarrollo de software libre y herramientas que se ajusten al desarrollo de la aplicación.

Capítulo 3: Diseño de la arquitectura propuesta

3- Introducción

Este capítulo tiene como objetivo principal diseñar la propuesta arquitectónica obtenida en la investigación. Se enuncian las metas y limitaciones de la arquitectura con el fin de decidir el alcance de la misma. Se describe finalmente la arquitectura del Portal del Grupo Calidad de la Facultad 9 mediante la representación de las vistas arquitectónicas definidas por RUP y se analizan otros aspectos importantes que de manera general proveen un mejor entendimiento de la organización y distribución del software.

3.1- Representación Arquitectónica

El Portal del Grupo Calidad de la Facultad 9 se desarrollará haciendo uso del patrón MVC, además se escogió Symfony como frameworks que utiliza el también seleccionado lenguaje de programación PHP, por esta razón parte de la arquitectura mostrada se ajusta a las características de estos frameworks. Los diagramas serán modelados con la herramienta CASE Visual Paradigm (versión 6.4), usando el Lenguaje Unificado de Modelado (UML 2.1). Para confeccionar la representación de la arquitectura se tomó como guía el proceso de desarrollo de RUP, basada en diferentes vistas que juntan los elementos más significativos del desarrollo de un sistema. Estas vistas son: La Vista de Casos de Uso, la Vista Lógica, la de Despliegue, de Procesos y de Implementación

3.2- Metas y Limitaciones de la Arquitectura

Cuando se va a desarrollar un software se extraen una serie de aspectos necesarios para obtener un producto con un resultado satisfactorio para el cliente. Estos aspectos pueden ser: condiciones o capacidades que el sistema debe cumplir, llamados requerimientos funcionales (RF) o propiedades o cualidades que el producto debe tener, conocidos como requerimientos no funcionales (RNF).

3.2.1- Requerimientos Funcionales

RF1. Autenticar usuario

RF2. Gestionar estudiante

RF2.1 Adicionar nuevo estudiante.

RF2.2 Modificar datos de un estudiante

RF2.3 Eliminar estudiante

RF2.4 Mostrar estudiante

RF3. Gestionar profesor

RF3.1 Adicionar nuevo profesor

RF3.2 Modificar datos de un profesor

RF3.3 Eliminar profesor

RF3.4 Mostrar profesor

RF4. Gestionar computadora

RF4.1 Adicionar nueva computadora

RF4.2 Modificar datos de una computadora

RF4.3 Eliminar computadora

RF4.4 Mostrar computadora

RF5. Mostrar reporte: El sistema debe mostrar un reporte de todos los estudiantes, profesores y computadoras del Grupo Calidad.

RF6 Crear planilla de evaluación de procedimiento de inicio

RF6.1 Adicionar proyecto certificado (si el proyecto cumple con todo lo verificado y obtiene evaluación de certificado, el sistema dará la opción de adicionar el proyecto a la lista de los proyectos certificados).

RF7 Buscador

RF8 Foro

RF9 Gestionar caso de prueba.

RF11 Gestionar paquete de revisiones.

RF11.1 Modificar planilla Acciones correctivas.

RF11.2 Modificar acta de reunión de apertura.

RF11.3 Modificar acta de reunión de cierre.

RF11.4 Gestionar Evaluación de desempeño de revisores (tiene la opción de adicionar más de una planilla de este tipo porque es una para cada revisor que participó en la revisión).

RF11.5 Modificar Informe final.

RF11.6 Mostrar lista de chequeo 1.0.

RF11.7 Mostrar lista de chequeo 2.0.

RF11.8 Gestionar paquete de seguimiento.

RF11.8.1 Modificar registro de evaluaciones del proyecto.

RF11.8.2 Modificar registro de no conformidades.

RF11.8.3 Modificar seguimiento a las no conformidades.

RF11.8.4 Gestionar plan de evaluaciones del proyecto (modificar y eliminar).

RF11.8.5 Modificar registro de evaluaciones

3.2.2- Requerimientos No Funcionales

RNF de Software

- Se debe utilizar como sistema operativo Windows 2000 o superior.
- Como navegador el Internet Explorer v 6.0 o Mozilla Firefox v 1.5 o superior instalado.
- La Base de Datos será implementada en el Gestor de Bases de datos PostgreSQL.

RNF de Hardware

- RAM 256MB

RNF Restricciones en el diseño y la implementación

- PHP5 (lenguaje de programación)
- PostgreSQL (gestor de BD)
- ZendStudio (para programar en PHP5)

RNF Requerimientos de apariencia o interfaz externa

Diseño perfectamente encuadrado para resoluciones de 800x600, pero preparado para verse en otras resoluciones.

RNF Requerimientos de Seguridad

Chequear que el usuario este autenticado antes de que pueda realizar alguna acción sobre el sistema.

En dependencia del rol que tenga el usuario podrá consultar, insertar, modificar o eliminar información.

Disponibilidad: El sistema deberá tener un 100% de disponibilidad por lo que podrá ser usado las 24 horas del día.

RNF Requerimientos de Usabilidad

El sistema podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de la computadora y de un ambiente Web en sentido general.

3.3- Vistas de la Arquitectura

La arquitectura, al ser el centro del desarrollo de software, debe ser descrita de forma que tanto desarrolladores como involucrados sean capaces de comprender la parte del sistema que les concierne, debe constituir un mapa donde se estructure desde diferentes perspectivas el software a construir. RUP propone que sea descrita a través de vistas arquitectónicas (4+1 vistas) que permiten especificar diferentes partes del sistema mediante diagramas y especificaciones.

Vista de Casos de Uso

Los Casos de Uso (CU) son una técnica para especificar el comportamiento y responder a las necesidades del sistema. Esta vista representa un subconjunto del artefacto Modelo de CU y lista los CU

más significativos arquitectónicamente de dicho modelo. Los CU significativos para la arquitectura que son aquellos que dan respuesta directa a requisitos funcionales y no funcionales que condicionan la arquitectura del sistema a desarrollar en dependencia de su impacto dentro del mismo.

Casos de Uso más significativos	Actores
Autenticar Usuario	Usuario
Gestionar Estudiante	Jefe de Recursos Humanos
Gestionar Profesor	Jefe de Recursos Humanos
Gestionar Computadoras	Jefe de Recursos Humanos
Mostrar Reporte	Usuario
Crear planillas de evaluación de procedimientos de inicio	Iniciador
Gestionar Caso de Prueba	Probador
Gestionar paquete de revisiones	Auditor
Modificar Planilla de acciones correctivas	Auditor
Modificar acta de reunión de apertura	Auditor
Modificar acta de reunión de cierre	Auditor
Gestionar evaluación de desempeño de revisores	Auditor
Modificar informe final	Auditor

Mostrar lista de chequeo 1.0	Auditor
Mostrar lista de chequeo 2.0	Auditor
Gestionar paquete de seguimiento	Auditor
Modificar registro de evaluaciones del proyecto	Auditor
Modificar registro de no conformidades	Auditor
Modificar seguimiento a las no conformidades	Auditor

Tabla 3: Casos de Uso más significativos para la arquitectura (Elaboración propia)

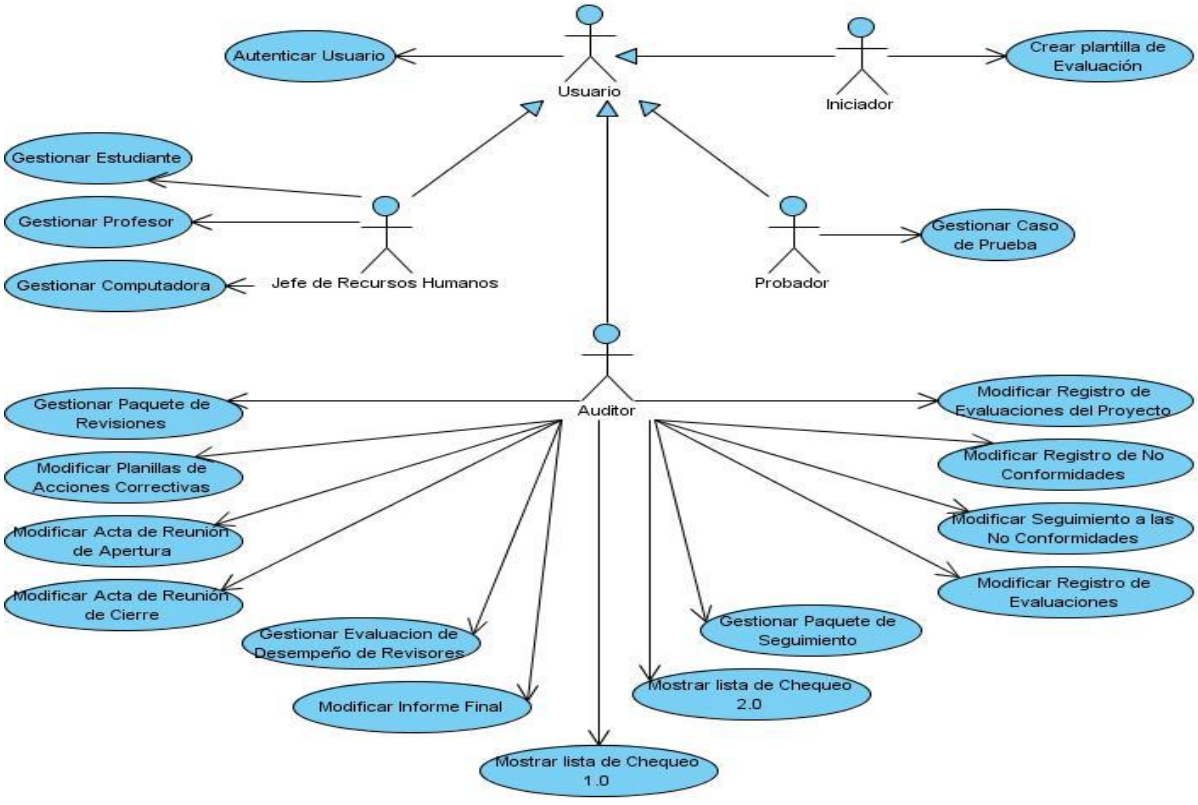

Figura 3: Vista de Casos de Uso (elaboración propia)

Vista Lógica

La vista lógica se encarga de describir un subconjunto del modelo de diseño. Contiene las clases significativas con su explicación detallada. Permite observar en el interior del sistema como está diseñada su funcionalidad, tanto su comportamiento estático como su comportamiento dinámico.

De acuerdo al estilo MVC que implementa el framework Symfony el diseño queda estructurado de manera general de la siguiente forma:

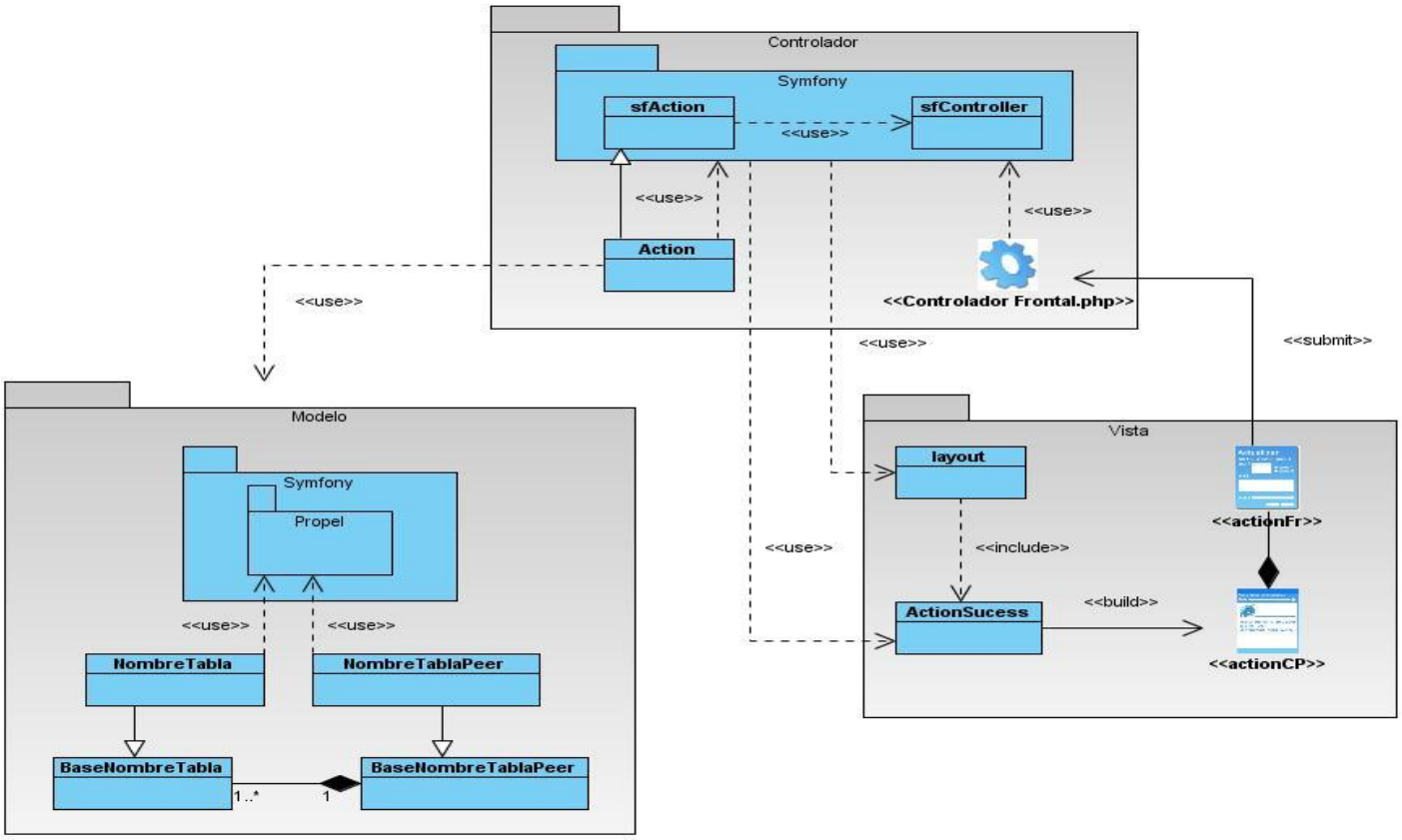

Figura 4: Vista Lógica (Elaboración propia)

El modelo: Se divide en dos partes fundamentales: la capa de acceso a datos y la abstracción de estos. Se establece de esta forma que las funciones que acceden a los datos no utilizan sentencias ni consultas que dependan de una base de datos, de manera que si se decide cambiar de sistema gestor de base de datos solo sería necesario actualizar la capa de abstracción. Este comportamiento permite que se logre un alto nivel de abstracción y una fácil portabilidad.

Symfony utiliza el proyecto Propel que es el encargado de la generación automática de las clases del modelo, además, traduce por cada tabla real de la base de datos cuatro clases las cuales se encargan en conjunto de realizar todo el acceso a los datos y la lógica de la aplicación. Estas clases son:

- NombreTabla y BaseNombreTabla: Son clases objetos que permiten acceder a las columnas de un registro de la base de datos y a los registros relacionados. Se encargan en conjunto de toda la lógica del negocio.

- NombreTablaPeer y BaseNombreTablaPeer: Contienen un conjunto de métodos estáticos para trabajar con las tablas de la base de datos que complementan el acceso y la lógica de los datos.

El controlador: Esta capa se divide en dos partes fundamentales, las acciones y el controlador frontal.

- Controlador Frontal: Es exclusivo en cada aplicación. Este realiza tareas comunes a los demás controladores que puedan existir en la aplicación como el manejo de las peticiones del usuario, el manejo de la seguridad y la carga de la configuración de la aplicación. Entre sus principales ventajas se encuentra la de constituir un punto de entrada único para toda la aplicación. Así, en caso de que sea necesario impedir el acceso a la aplicación, solamente es necesario editar el script correspondiente al controlador frontal. Tiene una clase llamada sfController que se encarga de decodificar la petición y transferirla a la acción correspondiente.

- Acciones: Se encargan de obtener los resultados del modelo y definen variables para la vista. Constituyen métodos con el nombre executeNombreAccion de la clase NombreModuloAction

que a su vez hereda de la clase sfActions, es importante acotar que el prefijo execute de las acciones es obligatorio.

La vista: Contiene tres partes fundamentales, el layout (conocido como plantilla global), el complemento de las acciones (llamado plantilla) y las páginas con sus formularios.

- El layout: almacena el código HTML que es común a todas las páginas de la aplicación. El contenido de la plantilla se integra en el layout, o si se mira desde el otro punto de vista, el layout decora la plantilla.

- El complemento de las acciones o plantilla: son las encargadas de, con los resultados de la acción, insertarse en el cuerpo de la layout y generar finalmente la página web resultado de la petición de un usuario. Esta plantilla tiene el mismo nombre que el sufijo de la acción correspondiente y termina en la palabra Success. Ejemplo: para una acción *executeIndex* existirá una plantilla *indexSuccess.*

- Páginas clientes y formularios: son las páginas que se generan finalmente y con las que el usuario interactúa.

Vista de Procesos

La Vista de Procesos soporta algunos RNF. Esta vista también específica que hilo de control ejecuta cada operación identificada en las clases de la vista lógica, se centra por tanto en la concurrencia y distribución de procesos.

Las aplicaciones Web se basan en la tecnología cliente-servidor, donde cada instancia del sistema en el lado cliente es independiente de la ejecución de otra instancia en el lado servidor, es decir no existen hilos de controles entre las mismas. La concurrencia de utilización de los Servidores Web y los servidores de BD no se manejan desde la aplicación sino a través de los propios servidores. A su vez la aplicación no tiene tareas que requieran ejecutarse de forma periódica sin la intervención del cliente. Debido a estas características, se llega a la conclusión de que no se requiere una Vista de Procesos.

Vista de Despliegue

La vista física se centra en los RNF. Suele utilizarse cuando el sistema está distribuido puesto que cada componente físico debe estar almacenado en un nodo, esto incluye también la asignación de tareas provenientes de la vista de procesos en los nodos.

Figura 5: Vista de Despliegue (Elaboración propia)

Descripción de los nodos expuestos e interfaces de comunicación

PC_Cliente: Desde este nodo se podrá acceder a la aplicación con la utilización de un navegador.

Servidor Web: En este nodo se ejecutarán todas las funcionalidades del servidor Web, entre ellas se encuentra la construcción de interfaces de usuarios y el procesamiento de datos.

Servidor de Base de Datos: En este nodo estará ejecutándose el servidor PostgreSQL aunque la lógica del tratamiento de los datos no se implementará en este sino en el servidor Web en la misma aplicación.

Protocolo TCP/IP: El protocolo de control de transmisión (TCP) garantiza que los datos sean entregados en su destino sin errores y en el mismo orden en que se transmitieron. El protocolo de Internet (IP) es usado tanto por el origen como por el destino para la comunicación de datos a través de una red.

Protocolo HTTPS: Protocolo seguro de transferencia de hipertexto, utiliza el cifrado de paquetes, garantizando así un canal seguro de comunicación entre la computadora cliente y los servidores apache. Garantiza la interoperabilidad con disimiles sistemas gestores de base de datos. Se integra perfectamente con PostgreSQL.

Vista de Implementación

Esta vista describe la descomposición del software en componentes y subsistemas de implementación organizados en capas y jerarquías, así como las relaciones entre estas. Básicamente se detalla la trazabilidad desde los paquetes y clases del modelo de diseño hasta subsistemas y componentes físicos.

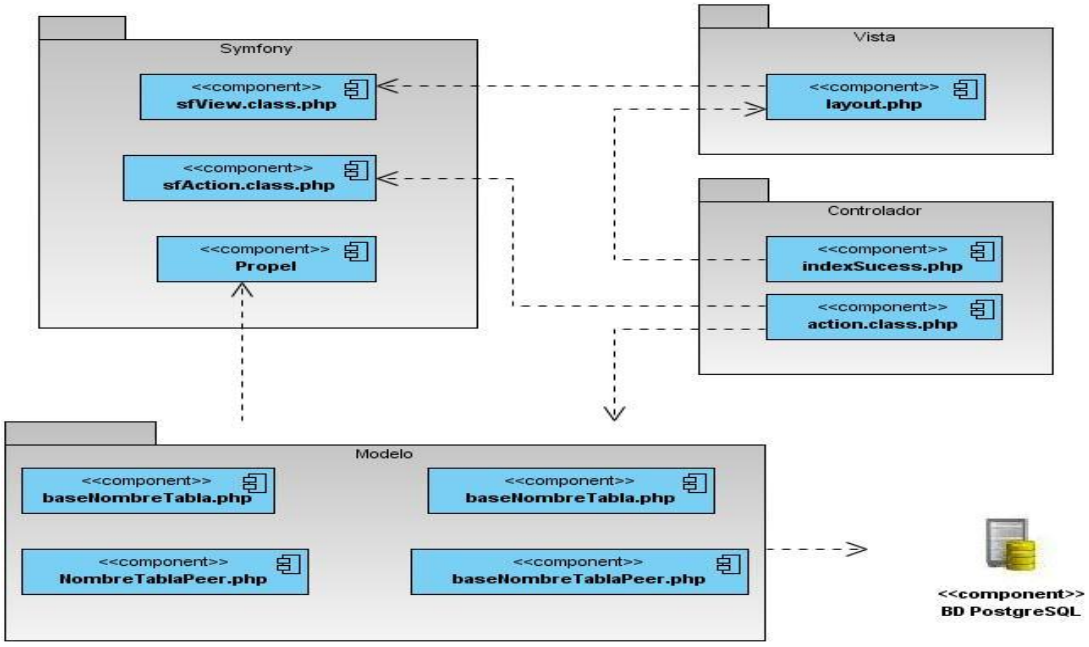

Figura 6: Vista de Implementación (Elaboración propia)

3.4- Conclusiones Parciales

En el presente capítulo quedó establecida una arquitectura candidata concebida a través de las metas, restricciones y de la descripción detallada de las vistas de la arquitectura. El diseño de dichas vistas se conformó de acuerdo a las características propias del sistema a desarrollar y a las del framework Symfony. Se concluye que la implementación utilizando dicho framework disminuye considerablemente el tiempo utilizado por el equipo de desarrollo pues parte del trabajo es realizado de antemano, al utilizar este framework solo es necesario concentrarse en las cuestiones propias del problema. Se diseñó una arquitectura robusta, flexible y que responde a las necesidades de los clientes.

Capítulo 4: Evaluación de la Arquitectura Propuesta

4- Introducción

En el presente capitulo se expondrán las etapas en las que se evalúa la arquitectura de software, así como las técnicas utilizadas para dicha evaluación. Luego se procederá a evaluar la arquitectura que se ha propuesto con el fin de conocer si es la adecuada para el Portal del Grupo Calidad de la Facultad 9. Considerando que la calidad de un software depende en gran medida de la calidad de la arquitectura que se proponga para el mismo se tendrán en cuenta los atributos de calidad asociados a esta propuesta.

4.1- Necesidad de Evaluar la Arquitectura

La evaluación de la arquitectura permite mitigar riesgos asociados con el desarrollo del software, cuando los errores son detectados en etapas tempranas, menos costosos serán los mismos en términos económicos y de tiempo requerido para solucionarlos. Una arquitectura robusta debe cumplir con los requisitos funcionales del sistema, al menos los básicos para una primera iteración, y con los atributos de calidad esperados del mismo. La parte ligada a estos atributos de calidad son los requisitos no funcionales del sistema, que poseen gran relevancia ya que especifican características que se esperan del mismo como: que sea robusto, fiable, que posea portabilidad, usabilidad, rendimiento, etc.

Las evaluaciones de la arquitectura tienden a aumentar la calidad, el control de gastos y el presupuesto. La arquitectura es el marco para todas las decisiones técnicas y tiene un importante impacto en el costo de los productos y la calidad. Una evaluación de la arquitectura no garantiza la alta calidad o el bajo costo a la hora de desarrollar un producto, pero puede señalar áreas de riesgo que de ser tratadas garantizan una mayor calidad del producto final disminuyendo además el costo del desarrollo del mismo.

4.2- Etapas de Evaluación de la Arquitectura

La evaluación de la arquitectura se puede realizar en cualquier etapa a lo largo del proceso de desarrollo, pero se debe enfatizar cuando en el proyecto se comienzan a tomar decisiones que dependen de la misma. Un aspecto importante a tener en cuenta es que cuando el costo de evaluar una arquitectura excede el costo de enmendar un error encontrado en dicha arquitectura, no se debe proceder a la

evaluación. Existen dos variantes que agrupan todos los momentos en que se puede evaluar una arquitectura, estas son:

- Evaluación temprana: En esta variante no es necesario que la arquitectura no esté completamente especificada para ser evaluada. Abarca desde las fases tempranas del desarrollo del proyecto hasta el final del mismo. Este puede sufrir cambios en la arquitectura en cualquier nivel puesto que pueden existir cambios, producto de una evaluación realizada.

- Evaluación tardía: En esta se establece la evaluación cuando la arquitectura está definida y el proyecto se encuentra implementado. A este nivel es muy importante la evaluación debido a que da una medida del cumplimiento de los atributos de calidad en el sistema y da una medida de cómo será su comportamiento (Bosch, 2000).

4.3- Atributos de Calidad

Para evaluar una arquitectura se necesita saber qué cualidades de esta se tomaran en consideración. Estas cualidades son llamados atributos de calidad. Los atributos a los que se hace referencia son requerimientos o características que el sistema debe tener, diferentes a los requerimientos funcionales. Estos se dividen en dos, los observables vía ejecución que son aquellos que se determinan del comportamiento del sistema en tiempo de ejecución y los no observables vía ejecución que se establecen durante el desarrollo del sistema.

Descripción de atributos de calidad observables vía ejecución.

Atributos	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 (Functionality)	Habilidad del sistema en la realización el trabajo para el cual fue concebido.

Desempeño (Performance)	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria.
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 4: Descripción de atributos de calidad observables vía ejecución (Elaboración propia)

Descripción de atributos de calidad no observables vía ejecución

Atributos	Descripción
Configurabilidad (Configurability)	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.
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, además de lograr que sea de forma 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 el que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
Capacidad de Prueba (Testability)	Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas.

Tabla 5: Descripción de atributos de calidad no observables vía ejecución (Elaboración propia)

4.4- Técnicas de Evaluación de Arquitecturas

Las técnicas de evaluación de la arquitectura se clasifican en cualitativas (escenarios, cuestionarios y listas de chequeo) y cuantitativas (métricas, normas, máximos y mínimos teóricos, simulaciones, prototipos, etc.). Estas técnicas permiten mediante la evaluación de la arquitectura establecer comparaciones entre arquitecturas candidatas para determinar cuál satisface más un atributo de calidad específico. En vista de que el interés es tomar decisiones de tipo arquitectónico en las fases tempranas del desarrollo, son necesarias técnicas que requieran poca información detallada y puedan conducir a resultados relativamente precisos. Las técnicas que posibilitan lo antes planteado son las cualitativas, que a su vez son las más utilizadas por los arquitectos debido a su bajo costo. Estas técnicas son:

- Evaluación basada en escenarios: Este consiste en crear escenarios donde exista un contexto determinado y una respuesta. Un escenario es una breve descripción de la interacción de uno de los involucrados en el desarrollo del sistema con este. Dentro de la evaluación por escenarios se utiliza lo que se conoce como el Utility Tree que no es más que un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle, el nivel de prioridad de cada uno.

- Evaluación basada en simulación: Esta evaluación está ligada al contexto de evaluación a partir de un prototipo funcional de la aplicación o el sistema a desarrollar donde se activaran los escenarios y se evaluaran en dependencia de los resultados los atributos de calidad esperados bajo ciertas condiciones impuestas básicamente para probar el desempeño de la aplicación.

- Evaluación basada en modelos matemáticos: Se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico, y se presentan como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados, utilizando los resultados de uno como entrada para el otro.

- Evaluación basada en experiencia: Establece que en muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evaluación de decisiones erradas de diseño. Aunque todas estas experiencias se basan en evidencia anecdótica; es decir, basada en factores subjetivos como la intuición y la experiencia. Sin embargo, la mayoría de ellas puede ser justificada por una línea lógica de razonamiento, y pueden ser la base de otros enfoques de evaluación. Existen dos tipos de evaluación basada en experiencia: la evaluación informal, que es realizada por los arquitectos de software durante el proceso de diseño, y la realizada por equipos externos de evaluación de arquitecturas.

4.5- Métodos de Evaluación de Arquitecturas

Los métodos de análisis de arquitectura están dados por una serie de actividades que se deben desarrollar por varios roles o implicados para evaluar el desempeño en cuanto a atributos de calidad de la arquitectura desarrollada. Entre estos métodos fundamentalmente se encuentran:

4.5.1- Método de Análisis de Arquitecturas de Software (SAAM)

El Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) es el primero que fue ampliamente promulgado y documentado. Según Kazman “Originalmente fue creado para el análisis de modificabilidad de la arquitectura, pero ha demostrado ser muy útil para evaluar ciertos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad” (R. Kazman, 2001). Este método se enfoca en la enumeración de un conjunto de escenarios que representan los cambios posibles a los que estará sometido el sistema en el futuro. Como entrada principal, es necesaria alguna forma de descripción de la arquitectura a ser evaluada.

Para la realización de las pruebas SAAM define tres roles que son los interesados externos representados por: la organización, el cliente, usuarios finales, administradores del sistema, etc. Están también los interesados internos: Arquitectos de Software, analistas, especialista en requerimientos. Por último se encuentra el equipo SAAM: Líder, expertos en el dominio de la aplicación, expertos externos en arquitectura y secretario.

La evaluación de este método está compuesta por seis (6) pasos, los cuales son:

1. Desarrollo de escenarios.
2. Descripción de la arquitectura.
3. Clasificación y asignación de la prioridad de los escenarios.
4. Evaluación individual de los escenarios indirectos.
5. Evaluación de la interacción entre los escenarios.
6. Creación de la evaluación global.

Y produce salidas como: La proyección sobre la arquitectura de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema y el entendimiento de la funcionalidad del sistema, e incluso una comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación.

El método se puede aplicar para evaluar una arquitectura o para evaluar varias, si el objetivo de la evaluación es una sola arquitectura se obtienen los puntos en los que la misma puede fallar, pero si el objetivo son varias arquitecturas el método permite definir qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones.

4.5.2- Método de Análisis de Acuerdos de Arquitectura (ATAM)

El nombre Método de Análisis de Acuerdos de Arquitectura (ATAM por sus siglas en ingles) surge del hecho de que revela la forma específica en que una arquitectura cumple con algunos atributos de calidad, así como la interacción de estos con otros(R. Kazman, 2001). Está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado

anteriormente. El método se centra en la búsqueda de enfoques arquitectónicos que son los medios empleados por la arquitectura para alcanzar los atributos de calidad.

La metodología de ATAM comprende nueve pasos divididos en cuatro fases.

Fase 1: Presentación.

1. Presentación del ATAM.
2. Presentación de las metas del negocio.
3. Presentación del negocio.

Fase 2: Investigación y análisis.

4. Identificación de los enfoques arquitectónicos.
5. Generación del Utility Tree12.
6. Análisis de los enfoques arquitectónicos.

Fase 3: Pruebas.

7. Tormenta o lluvia de ideas y establecimientos de la prioridad de los escenarios.
8. Análisis de los enfoques arquitectónicos.

Fase 4: Reportes.

9. Presentación de los resultados.

4.5.3- Método de Análisis de Diseños Intermedios (ARID)

El método de Análisis de Diseños Intermedios (ARID por sus siglas en ingles) fue creado para evaluar diseños parciales de la arquitectura en las etapas tempranas del desarrollo. ARID es un híbrido entre los métodos ATAM; visto anteriormente; y Active Design Review (ADR). En el caso de ADR proporciona una documentación detallada del diseño y completan cuestionarios. En el caso de ATAM se realiza una evaluación de la arquitectura en su conjunto. El método de evaluación comprende nueve pasos divididos en dos fases.

Fase 1: Actividades previas.

1. Identificación de los encargos de la revisión.
2. Preparar el informe de diseño.
3. Preparar los escenarios bases.
4. Preparar los materiales.

Fase 2: Revisión.

5. Presentación del ARID.
6. Presentación del diseño.
7. Lluvia de ideas y establecimiento de prioridad de escenarios.
8. Aplicación de los escenarios.
9. Resumen.

Entre los resultados arrojados por el método pueden encontrarse esencialmente: el conjunto de escenarios y sus prioridades, el árbol de utilidad, los riesgos descubiertos, los no riesgos documentados y los puntos de sensibilidad y los acuerdos de puntos encontrados.

4.6- Evaluación de la propuesta arquitectónica.

En función de evaluar la propuesta arquitectónica y teniendo en cuenta que la arquitectura es un producto temprano del diseño, se decidió utilizar el método ARID. Entre las ventajas de este método se nombran su facilidad de uso, su aplicación poco costosa y de gran beneficio y el logro de una mejor evaluación de la factibilidad de la arquitectura. ARID utiliza la técnica de evaluación basada en escenarios con el instrumento perfiles, donde se hace un análisis de los principales escenarios arquitectónicos. Dada la realidad de que el arquitecto de software fue el encargado de la definición de la arquitectura y entiende los procesos descritos en la misma ya que ha sido parte del equipo de desarrollo de la aplicación, se consideró omitir los pasos de la primera fase del método, pasando directamente a aplicarse los tres últimos pasos de la Fase 2.

A continuación se muestran diferentes tablas que resumen los pasos finales de la fase última del método ARID formado por: el establecimiento de prioridad de escenarios, la aplicación de estos y el resumen de la relación entre los atributos de calidad y dichos escenarios. Estos escenarios han sido escogidos por ser primordiales para la primera iteración del Portal del Grupo Calidad de la Facultad 9.

Escenario # 1	Duplicación de los datos
Perfil	Seguridad
Atributo de Calidad	Funcionalidad – Integridad
Relación Atributo-Escenario	
El framework Symfony cuenta con un controlador frontal que es el encargado de velar que la petición del usuario que llega desde la vista, sea correcta y se pueda responder. Si esta solicitud presenta algún tipo de error, el controlador envía un mensaje a la vista, y esta a su vez se encarga de mostrarlo al usuario. De esta forma si el usuario pide insertar un dato ya creado, se mostrara un mensaje comunicando que dicha información ya ha sido creada y se le dará la posibilidad de modificarlo, leerlo o eliminarlo. Lógicamente esto solo sucederá si dicho usuario tiene los permisos necesarios para realizar este tipo de operaciones. Todas estas acciones están encaminadas a evitar la duplicación de los datos ya creados.	

Tabla 6: Escenario # 1. Duplicación de los datos (Elaboración propia).

Escenario # 2	Migración del sistema gestor de base de datos
Perfil	Configuración - Mantenibilidad
Atributo de Calidad	Modificabilidad – Configurabilidad

Relación Atributo-Escenario
El framework Symfony divide la capa del modelo en dos partes, una encargada del acceso a datos y otra que se ocupa de la abstracción de la base de datos. La abstracción de la base de datos es gestionada por Creole que es usado por Propel, el cual es un ORM encargado de generar el modelo del MVC. De tal manera, si se hace necesario cambiar el SGBD solo habría que configurar un parámetro en la clase conexión, sin que esto afecte los datos almacenados o demás capas de la aplicación.

Tabla 7: Escenario # 2. Migración del sistema gestor de base de datos (Elaboración propia).

Escenario # 3	Migración de Sistema Operativo
Perfil	Portabilidad
Atributo de Calidad	Portabilidad
Relación Atributo-Escenario	
El sistema será desarrollado utilizando como lenguaje PHP5, el entorno de ejecución será el contenedor web Apache, para la persistencia de datos se utilizará el sistema gestor de base de datos PostgreSQL y para la interfaz de usuario, se utilizará para su desarrollo lenguaje HTML y JavaScript. Cada uno de los elementos seleccionados para el desarrollo de la aplicación son compatibles con la mayoría de los sistemas operativos más populares; la migración, de PostgreSQL, puede realizarse sin que afecte el rendimiento; mientras que para la capa de interfaz de usuario solamente requerirá de un navegador web utilizado por el cliente.	

Tabla 8: Escenario # 3. Migración de Sistema Operativo (Elaboración propia).

Escenario # 4	Intrusión de usuario
Perfil	Seguridad
Atributo de Calidad	Seguridad
Relación Atributo-Escenario	
Symfony provee mecanismos para la restricción de acceso al nivel más mínimo, donde cada acción antes de ser ejecutada pasa por un filtro especial que verifica si el usuario tiene privilegios para acceder a las misma. Estos privilegios están compuestos por dos partes: las acciones seguras requieren que los usuarios estén autenticados. Las credenciales son privilegios de seguridad agrupados bajo un nombre y permiten organizar la seguridad en grupos. De esta manera cuando un usuario intente acceder a una información para la cual no tiene los permisos necesarios se mostrara la página de autenticación conjuntamente con un mensaje de no permisibilidad a los datos. De cualquier otra forma dicho usuario podrá acceder a la información común para todos los interesados.	

Tabla 9: Escenario # 4. Intrusión de un usuario (Elaboración propia).

Escenario # 5	Aumento del modelo de datos
Perfil	Mantenibilidad
Atributo de Calidad	Escalabilidad - Modificabilidad
Relación Atributo-Escenario	
El SGBD PostgreSQL brinda la posibilidad de crear tantos clústeres de bases de datos como sea necesario. Esto permite aumentar la capacidad de almacenamiento y disminuir la sobrecarga del proceso servidor en caso que la concurrencia o el espacio sean insuficientes.	

Tabla 10: Escenario # 5. Aumento del modelo de datos (Elaboración propia).

La evaluación realizada refleja que la aplicación cumple con los atributos de calidad requeridos como son la escalabilidad, la modificabilidad, la seguridad, la configurabilidad, la portabilidad, funcionalidad e integridad. A su vez se arrojan resultados que evidencian la resolución de la situación problemática que da vida a esta investigación, en este caso se encuentra la disponibilidad de los documentos para los interesados en todo momento, la eliminación de la duplicación de los mismos, entre otros.

Por lo anteriormente planteado se puede afirmar que el diseño de la arquitectura propuesta, dentro de una etapa temprana de desarrollo, es adecuado.

4.7- Conclusiones Parciales

En el presente capítulo se destacaron las ventajas de realizar una evaluación, ya que 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 analizaron brevemente los atributos de calidad de la arquitectura propuesta. Se explicaron algunos de los métodos de evaluación de arquitectura y se valoró la aplicación de uno de ellos a la solución propuesta. Se arrojaron los resultados encontrados como parte de la evaluación de la arquitectura. Pudiéndose concluir que la arquitectura propuesta cumple de manera satisfactoria los requerimientos de calidad y es adecuada para el desarrollo del Portal del Grupo Calidad de la Facultad 9.

Conclusiones Generales

Al llegar al desenlace de esta investigación se puede arribar a las siguientes conclusiones:

El recorrido por la historia de los principales conceptos relacionados con la arquitectura, tales como patrones, estilos arquitectónicos y frameworks para el desarrollo del software, realizado mediante el método científico análisis histórico- lógico, permitió la selección de los mejores y más apropiados.

El estudio de las tecnologías utilizadas a nivel mundial y el conocimiento de los requisitos de los clientes, condujo a que se definiera la metodología a utilizar, así como los lenguajes, IDE, objetivos y restricciones a tener en cuenta para la realización de la aplicación. Todas las elecciones fueron fundamentadas.

Se describió la arquitectura mediante las vistas definidas por RUP (Caso de uso, Lógica, Implementación y Despliegue), lo que garantizó que se tuviera una visión de todos los elementos arquitectónicamente significativos para el proceso de implementación y que se pudiera llegar a un entendimiento conjunto en el equipo de desarrolladores.

Se realizó un estudio detallado de los principales métodos de evaluación de arquitecturas software. Evaluándose la propuesta mediante el método ARID, siendo considerado el más efectivo en este caso debido a que la propuesta arquitectónica se hace en una fase temprana del desarrollo del sistema. Esta evaluación condujo a que se considerara adecuada la arquitectura propuesta para el desarrollo del sistema.

La idea de que el diseño de una arquitectura, permitiría que se agilizara el manejo de la información y el flujo de trabajo en el proyecto, planteada como propuesta de solución inmediata al problema científico de la investigación, ha quedado validada de manera satisfactoria dados los resultados de la descripción de la arquitectura y el comportamiento de los atributos de calidad en la solución propuesta, dándole cumplimiento de esta manera al objetivo general de la investigación.

Recomendaciones

Se recomienda:

Utilizar la arquitectura propuesta en otras aplicaciones con características semejantes.

El refinamiento constante de la arquitectura propuesta, durante el ciclo de desarrollo, ya que la arquitectura deberá adecuarse a los nuevos avances tecnológicos, orientada siempre al desarrollo de tecnologías libres.

Continuar profundizando en la validación de la arquitectura propuesta con un método de evaluación tardía, se propone realizar dicha validación aplicando el método ATAM.

Bibliografía y Referencias Bibliográficas.

Bustelo Ruesta, Carlota y Amarilla, Raquel. 2008. Inforarea Consultores de Información y Documentación. *Inforarea Consultores de Información y Documentación.* [En línea] 3 de febrero de 2008. [Citado el: 12 de diciembre de 2009.] http://www.inforarea.es/bedoc.asp.

Cid, Reynier Arias. 2009. *Propuesta de arquitectura de un subsistema de modelado de Diagramas de Flujo de Información.* Habana : s.n., 2009.

Clemens, Paul. 2008. *A Survey of Architecture Description Languages*. Proceedings of the International Workshop on Software Specification and Design. Alemania : s.n., 2008.

Cornejo, José. 2001. Arquitectura en Capas ~ DNA. Arquitectura en Capas ~ DNA. [Online] marzo 25, 2001. [Cited: enero 7, 2009.] http://www.docirs.cl/arquitectura_tres_capas.htm.

Eguiluz, Javier. 2007. Maestros del Web. *Maestros del Web.* [Online] 6 septiembre, 2007. [Cited: febrero 6, 2010.] http://www.maestrosdelweb.com/editorial/el-framework-symfony-una-introduccion-practica-i-parte/.

Fowler, Martin. 2002. *Patterns of Enterprise Application Architecture:* s.n.: Addison Wesley, 2002. 0-321-12742-0.

Gamma, Erich, et al. 2005. *Design Patterns: Elements of Reusable Object-Oriented Software.* 2005. 0-201-63361-2.

Gestión de información, gestión del conocimiento y gestión de la calidad de las organizaciones . **Quiroga, Lourdes Aja. 2005.** La Habana : Revista Cubana de los Profesionales de la Información y la Comunicación en Salud, 2005, Vol. 10.

IBM. 2003. *Ayuda de Rational Rose.* [IBM] s.n.: IBM, 2003.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Education S.A, 2000. 84-7829-036-2

R. Kazman, P. Clement, M. Client. 2001. *Evaluating Software architectures. Methods and cases studies.* s.n.: Addison Wesley, 2001.

Mark Klein, Rick Kazman. 1999. *Attribute-based architectural styles.* Carnegie Mellon University: Technical Report, CMU/SEI-99-TR-022, Octubre de 1999. ESC-TR-99-022.

Organización Internacional de Normalización. 2007. International Organization of Standarization. *International Organization of Standarization.* [En línea] 2007. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45991

Perry, Dewayne E. y L.Wolf, Alexander. 1992. *Foundation for the study of software architecture.* 1992.

Rabaix, Thomas. 2008. The Mail Archive. *The Mail Archive.* [En línea] 18 de Mayo de 2008. [Citado el: 15 de marzo de 2010.] http://www.mail-archive.com/symfony-users@googlegroups.com/msg07161.html.

Reynoso, Carlos Billy. 2004. Introduccion a la Arquitectura de Software. *Introduccion a la Arquitectura de Software.* Buenos Aires : s.n., 2004.

Reynoso, Carlos Billy. 2004. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires : s.n., 2004.

Sánchez, María A. Mendoza. 2004. *Metodologías De Desarrollo De Software.* Peru: s.n., 2004.

Shaw, Mary y Garlan, David. 1996. *Software Architecture: Perspectives on an emerging discipline.* Upper Saddle River, Prentice Hall, 1996.

Stig Sæther Bakken, Egon Schmid y Jim Winstead. 2001. *Manual de PHP.* 2001.

Sun Microsystems. 2008. *MySQL 5.0 Manual de Referencia.* 2008.

Zend Company. 2009. ZendFramework.com. [En línea] 2009. [Citado el: 12 de Febrero de 2009.] http://framework.zend.com/.