Universidad de las Ciencias Informáticas

Facultad 3

Título: Arquitectura del Tutor Virtual de Evaluación para el Aprendizaje Autónomo de Idiomas (VIRTEVALL)

Trabajo de Diploma para optar por el título de
Ingeniero Informático

Autora: Odelkis Rodriguez Irsula

Tutores: MSc. Yoan Martínez Márquez
Ing. Reinier Castillo González
Lic. Moisés A. Mayet Solano

Ciudad de La Habana, Junio 2010
Año 52 de la Revolución

DECLARACIÓN DE AUTORÍA

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

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

_____

Odelkis Rodriguez Irsula

Autor

_____

MSc. Yoan Martínez Márquez

Tutor

_____

Ing. Reinier castillo González

Tutor

_____

Lic. Moisés Alain Mayet Solano

Tutor

DATOS DE CONTACTO

Tutor: MSc. Yoan Martínez Márquez

Especialidad de graduación:

- Licenciado en Educación. Especialidad: Lengua Inglesa. 2004.
- Máster en Ciencias de la Educación. Especialidad: Tecnologías en los Procesos Educativos. 2007

Categoría docente: Asistente

Años de experiencia en el tema: 6 años

Año de graduado: 2004

Correo electrónico: yoanm@uci.cu

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

Tutor: Ing. Reinier Castillo González

Especialidad de graduación: Ingeniería en Ciencias Informáticas

Categoría docente: Instructor recién graduado en Adiestramiento

Años de experiencia en el tema: 2 años

Año de graduado: 2008

Correo electrónico: rcgonzalez@uci.cu

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

Tutor: Lic. Moisés Alain Mayet Solano

Especialidad de graduación: Licenciado en Ciencia de la Computación

Categoría docente: Instructor

Años de experiencia en el tema: 5 años

Año de graduado: 2005

Correo electrónico: mmayets@uci.cu

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

"Programar sin una arquitectura en mente es como explorar una gruta sólo con una linterna: no sabes dónde estás, dónde has estado ni hacia dónde vas."

Danny Thorpe

Agradecimientos

A mi abuelita Ederia, que a pesar de decir adiós cuando más lo necesitaba, ha sido la luz que me guía en la recta final.

A mi mamita por todo su amor y comprensión en cada paso de mi vida, por ser la estrella que me alumbra en las noches oscuras.

A mi papi por crear en mí el lado fuerte y seguro que me ha hecho crecer cada día y convertirme en la mujer que nunca soñó.

A mi hermanito, por confiar en su tata siempre, por ser el granito de amor que necesito para ser feliz. No te fallaré nunca.

A mi hermosa y gran familia, por apoyarme en todo lo que hago.

A esa persona que llegó a mi vida sin ser esperada: mi novio Hamlet; muchas gracias por enseñarme a amar y ser mi todo, por tus sabios consejos y tu amor. Te quiero, tito.

Gracias a las hermanitas que la vida me ha regalado por estar siempre a mi lado tanto en las buenas como en las malas: Maylín, Nallelys, la FlaK e Isabelita.

A todos los amigos que he tenido el gran placer de conocer durante toda mi vida, en especial a aquel que ha sido mi ejemplo a seguir desde mi primer año en la Universidad y se ha convertido en el hermano mayor que no tengo: Yandryto.

A mis tutores por su preocupación y su ayuda incondicional en los tiempos difíciles.

Al equipo de desarrollo de VIRTEVALL: sin ustedes nada hubiera sido posible.

A todos, MUCHAS GRACIAS de todo corazón.

Dedicatoria

A todos los que de una forma u otra son parte de mi vida.

Resumen

En los sistemas de Educación Asistida por Computadora (EAC), la máquina puede funcionar como herramienta, alumno o maestro. En este último caso, los más sofisticados son los Sistemas Tutores Inteligentes (STI) que actúan como un tutor particular del estudiante, de acuerdo a las necesidades que posea éste. Por tal motivo se busca diseñar un sistema adaptable de acuerdo a los conocimientos previos y a la capacidad de evolución de cada estudiante en el aprendizaje de idioma.

En el presente trabajo de diploma se describe la arquitectura de un STI. Para su realización se estudiaron las principales metodologías, herramientas y tecnologías de software con el objetivo de obtener una arquitectura de software que potencie los requisitos no funcionales, con el objetivo de lograr una mayor flexibilidad, escalabilidad y robustez.

La arquitectura definida es descrita a partir de los artefactos correspondientes al rol de arquitecto de la metodología RUP, basados en el modelado de vistas arquitectónicas. Una vez realizada la descripción de la arquitectura, ésta es validada con el fin de determinar el cumplimiento con los atributos de calidad establecidos a partir de escenarios.

Palabras Claves: aprendizaje, arquitectura de software, evaluación, idioma, Tutor Inteligente, virtual.

Índice

Índice de Figuras

Índice de Tablas

Introducción

En la actualidad, los sitios Web además de ser una fuente de información, se han convertido en el medio adecuado para realizar diversas acciones docentes. La educación, a su vez, ha avanzado en la aplicación de las Tecnologías de la Información y las Comunicaciones (TIC) en el proceso de aprendizaje, proporcionando nuevas opciones educativas integradas a los sitios Web, con la intención de personalizar las actividades en la Enseñanza Asistida por Computadoras (EAC).

La EAC es un "*ambiente de aprendizaje caracterizado por la interacción educativa entre la computadora y el estudiante*" [1]. En la actualidad, los sistemas de EAC más complejos y con mayor potencial son los Sistemas Tutores Inteligentes (STI). Un STI es un sistema capaz de comportarse de manera similar a un tutor humano; o sea, que se adapta a lo que el estudiante realmente necesita e identifica cómo éste resuelve un problema para luego brindarle ayudas cognitivas cuando lo requiera.

Un tutor inteligente, por lo tanto: "*es un sistema de software que utiliza técnicas de Inteligencia Artificial (IA) para representar el conocimiento e interactúa con los estudiantes para enseñárselo*" [2]. Wolf define los STI como: "*sistemas que modelan la enseñanza, el aprendizaje, la comunicación y el dominio del conocimiento del especialista y el entendimiento del estudiante sobre ese dominio*". [3]

El abastecimiento de información y el mejoramiento de la experiencia educativa a través de nuevos recursos pedagógicos ha contribuido a facilitar el proceso cognitivo del estudiante. A esto, se le puede añadir el entorno flexible respecto al acceso a la red desde cualquier punto donde el STI se encuentre almacenado. Esto no sólo significa una reducción de costos importantes (en materiales y tiempo) sino que constituye una mejora en el uso de las plataformas para la educación a distancia.

A esta alternativa, se puede sumar que la educación basada en Web es cada vez más atractiva para las instituciones académicas. Se podría citar como ejemplo que para lograr certificaciones de Cisco Systems[1], como la "Cisco Certified Network Associate" (CCNA), la empresa cuenta con un campus virtual, con todo el material de la certificación almacenado en varios idiomas, existiendo un expediente del alumno y los

[1] La empresa Cisco Systems realiza hardware para redes y software para manipular este hardware. Las certificaciones CCNA, entre otras, como la Cisco Certified Internetwork Expert (CCIE), Cisco Certified Security Professional (CCSP), Cisco Certified Network Professional (CCNP), Cisco Certified Voice Professional (CCVP), Cisco Certified Design Associate (CCDA) utilizan un sistema de educación a similar.

cuestionarios a resolver. Con el uso de este sistema se centraliza todo el proceso, permitiendo que los alumnos tomen el curso en sus hogares o en cualquier establecimiento habilitado para luego rendir los exámenes, también en línea. [4]

El desarrollo de tales sistemas es hoy una posibilidad en Cuba, dado a que en todas las universidades se cuenta con el equipamiento necesario para ello. Uno de los más significativos es el Sistema Tutorial Inteligente para el Diagnóstico y Tratamiento de las Infecciones de Transmisión Sexual (STIITS), que como aplicación es considerado el primero de su tipo en Cuba. Este STI fue desarrollado por la Universidad de Cienfuegos en conjunto con especialistas de segundo grado del Hospital Dr. Gustavo Aldereguía Lima y presentado en el VII Congreso Internacional de Informática en Salud en el mes de febrero del 2009. STIITS responde a una necesidad social y tiene como valor agregado la posibilidad de ser empleado en la docencia.

La Universidad de las Ciencias Informáticas (UCI) cuenta con disímiles herramientas que contribuyen al proceso de auto-aprendizaje de los estudiantes. La materia de Idioma Extranjero está integrada al uso de una estas herramientas, denominada Entorno Virtual de Aprendizaje (EVA), que permite a los estudiantes la realización de numerosos diagnósticos de ejercitación y la consolidación de los conocimientos, recibiendo una calificación de éstos. No obstante, el EVA no es capaz de guiar y monitorear el avance que posee el estudiante durante su aprendizaje, ni de reconocer el estilo de aprendizaje y el nivel académico de éste. Para darle solución a las desventajas que presenta el EVA se comenzó a desarrollar el *Centro Virtual de Auto-aprendizaje de Lenguas Extranjeras (CEVALE)* que surgió a partir del re-diseño de la estrategia curricular de la disciplina de Idioma Inglés en la Facultad 9, sin embargo, no pudo ser desarrollado exitosamente.

En el año 2009, se crea la aplicación VIRTEVALL *(Virtual Tutor of Evaluation for Autonomous Learning of Languages*, Tutor Virtual de Evaluación para el Aprendizaje Autónomo de Idiomas en español), como aporte práctico de un Proyecto de Innovación Pedagógica que tiene como nombre: “Metodología de Evaluación para el Aprendizaje Autónomo de Idiomas” que responde a una tesis doctoral y a su vez la aplicación constituirá dos tesis de maestría y cuatro tesis de diploma en su primera versión. En este STI se pretende que los estudiantes sean autónomos en cuanto a la evaluación para el aprendizaje de las Lenguas Extranjeras. Esta evaluación se realizará de acuerdo a sus debilidades y recibirán un tratamiento para erradicarlas. El sistema tendrá como referencia el TOEFL (*Test of English as a Foreign Language*),

TOEIC (*Test of English for International Communication*) e IELTS (*International English Language Testing System*), los cuales son considerados como exámenes internacionales estandarizados para la certificación de nivel de idiomas. La UCI, como infraestructura docente-productiva no posee un sistema con semejantes características dirigido a la evaluación para el aprendizaje de las Lenguas Extranjeras.

Sistemas como estos necesitan de una Arquitectura de Software (AS) que le propicie robustez, dado que de las decisiones arquitectónicas correctas que se tomen en el proceso depende en gran medida el éxito del desarrollo de la solución. Una buena AS significa una estructura del sistema estable que puede adaptarse a los cambios de requisitos y de tecnologías.

Una vez analizado lo anterior, se plantea como **Problema a Resolver** durante la investigación: La inexistencia de una línea base o arquitectura que satisfaga los requisitos que se desean desarrollar para el Tutor Virtual de Evaluación para el Aprendizaje Autónomo de Idiomas.

El **Objeto de Estudio** está orientado al Proceso de Desarrollo de Software, el cual tiene enmarcado como **Campo de Acción** la Arquitectura de Software de los Sistema Tutor Inteligente.

A partir del problema referenciado anteriormente se puede enunciar la siguiente **Idea a Defender**: Si se logra diseñar la línea base de la arquitectura para el Tutor Virtual de Evaluación para el Aprendizaje Autónomo de Idiomas entonces se potenciará el rendimiento de las tecnologías definidas para el proyecto.

Para dar solución a la problemática, se define como **Objetivo General** diseñar la arquitectura para el Tutor Virtual de Evaluación para el Aprendizaje Autónomo de Idiomas.

El cumplimiento del objetivo general se basará en los siguientes **Objetivos Específicos**:

- Analizar las arquitecturas utilizadas en los Sistema Tutor Inteligente.
- Definir una arquitectura para el Tutor Virtual de Evaluación para el Aprendizaje Autónomo de Idiomas.
- Evaluar la arquitectura definida.

Se realizarán las siguientes **Tareas de la Investigación** que harán posible que se lleven a cabo los objetivos anteriormente expuestos:

- Elaboración del diseño teórico de la investigación.
- Estudio del estado del arte de las arquitecturas en los Sistema Tutor Inteligente.

- Valoración del estado del arte de los estilos y patrones arquitectónicos.
- Selección de las tecnologías, frameworks y herramientas de desarrollo a utilizar en la solución.
- Establecimiento de la línea arquitectónica a utilizar en el Tutor Virtual de Evaluación para el Aprendizaje Autónomo de Idiomas.
- Diseño de los artefactos correspondientes al rol Arquitecto en el desarrollo de la propuesta.
- Evaluación de la arquitectura diseñada para el desarrollo del Tutor Virtual de Evaluación para el Aprendizaje Autónomo de Idiomas mediante métodos de evaluación.

Con la realización de estas tareas se espera obtener como **Posible Resultado** la Arquitectura del Tutor Virtual de Evaluación para el Aprendizaje Autónomo de Idiomas.

Esta investigación está estructurada en 3 Capítulos fundamentales.

Capítulo 1. Fundamentación Teórica

En este capítulo se realiza un estudio referencial de las arquitecturas más utilizadas en este tipo de sistema para la selección de la que llevará el sistema. Además, de los estilos arquitectónicos, patrones, frameworks, sistemas gestores de base de datos y herramientas de desarrollo a tener en cuenta en la elaboración de la arquitectura del sistema.

Capítulo 2. Descripción Arquitectónica

En este capítulo se hace una propuesta de solución arquitectónica para el desarrollo del software. Se expone cómo estará estructurado el sistema, cuáles serán los requisitos no funcionales, además de los artefactos correspondientes al rol de Arquitecto.

Capítulo 3. Evaluación de la Arquitectura.

En este capítulo se analizan los resultados obtenidos con la propuesta de arquitectura, además de evaluarla mediante el uso de un método de evaluación.

Capítulo 1: Fundamentación Teórica

En este capítulo se realiza un estudio de los conceptos y temas relacionados con la Arquitectura de Software, partiendo de una versión bibliográfica de la arquitectura de un Sistema Tutor Inteligente, logrando su entendimiento. Se investigan los estilos y patrones arquitectónicos posibles a utilizar en esta línea base. Además, se seleccionan la metodología, herramientas y tecnologías a utilizar, fundamentando sus características, ventajas y desventajas de su uso en el desarrollo del software.

1.1. Arquitectura de STI

Para obtener un STI con módulos intercambiables que puedan interactuar entre sí, se debe partir del modelo tripartito propuesto por Carbonell [5] en la que cada módulo posee una función específica.

Al incorporar elementos de la inteligencia artificial, los sistemas tutoriales inteligentes (STI) son los sistemas más complejos y poderosos en el campo de la EAC [6]. Un sistema tutorial se puede considerar "inteligente" según la medida en que cumpla tres criterios [7]:

1. El dominio a enseñar debe ser conocido por el sistema. Es decir, éste debe poder hacer inferencias o resolver problemas en este dominio.
2. El sistema debe poder deducir la cercanía del conocimiento del estudiante con el propio.
3. El sistema debe poder adaptar su estrategia de enseñanza para reducir la distancia entre su conocimiento y el del estudiante.

Estos criterios se han implementado, tradicionalmente, en tres módulos: Dominio, Tutor y Estudiante. Pero un sistema tutorial inteligente debe tener también una interfaz, y puede beneficiarse de estar inmerso en un complejo ambiente educativo.

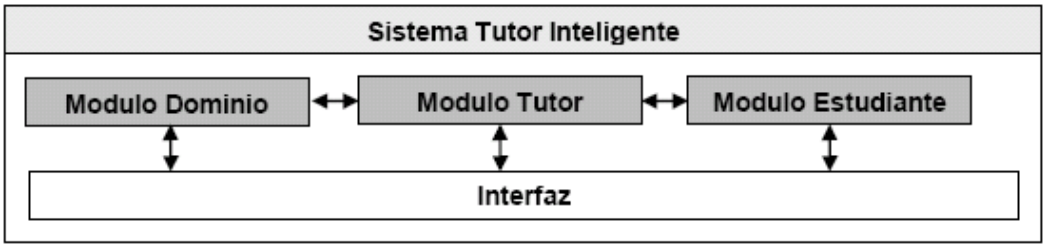

Figura 1. Estructura clásica de un STI propuesta por Carbonell.

A pesar de la existencia de una arquitectura común para los STI, existen diferentes puntos de vista como lo es la propuesta de una arquitectura general en Clancey [8] la cual se resume a continuación:

- Dos sistemas expertos: Uno que se encarga de diagnosticar el estado actual del estudiante y el otro, de decidir la mejor forma de retroalimentarlo al momento.
- Un sistema experto de diagnóstico: Encargado de escoger la mejor estrategia de retroalimentación a través de un algoritmo de decisión, basándose en lo conocimientos del estudiante.

Módulo del Estudiante

El diseño del modelo del estudiante está centrado en lo que se desea que el estudiante sepa acerca de la manera de resolver un problema y el tipo de conocimiento que tiene para resolverlo. De forma tal, que si un estudiante elige examinar un componente se asume que conoce algo del componente.

Este módulo tiene como objetivo principal realizar interpretaciones teniendo como base las acciones del estudiante, para luego inferir lo que el estudiante conoce. Por lo tanto, es importante conocer el estado y el estilo de aprendizaje de cada uno de los alumnos para que el módulo tutor pueda tomar las decisiones pedagógicas adecuadas. [9]

Módulo del Tutor

La idea principal del módulo es utilizar la computadora como herramienta de aprendizaje, para la cual toma decisiones pedagógicas en función de las interacciones con el estudiante; interacciones que se encuentran derivadas del conocimiento del tutor con respecto al dominio y se encuentran representadas de forma definida en el sistema. El tutor debe utilizar la información que obtiene del comportamiento del estudiante con el sistema para proveerlo de información útil para este. En resumen, el tutor debe ser capaz de responder las interrogantes: *¿Cuándo es necesario instruir? ¿Qué tipo de instrucción debe darse?* [9]

Módulo del Dominio

En el módulo del dominio se debe descomponer el conocimiento de los temas en partes identificables. Entre estos se encuentra el conocimiento dependiente del dominio, el cual lo componen las definiciones, los conceptos fundamentales y las agrupaciones de conceptos que conforman los temas. Por otra parte está el comportamiento independiente del dominio, compuesto por diferentes parámetros del sistema que

se requieren para el correcto funcionamiento del mismo y por último se encuentran los elementos didácticos que se utilizan para mejorar y facilitar la experiencia del proceso enseñanza-aprendizaje. [9]

Módulo de Interfaz

La interfaz se encarga de presentar en tiempo y forma los elementos pedagógicos de cada sesión (ejercitación, material multimedia, textos, etc.). A pesar de existir en todos los STI, no es considerado como un módulo fundamental, dado a que se encuentra centrada en el aspecto gráfico y no en los contenidos.

Es posible generar una interfaz que satisfaga al usuario, donde este pueda establecer parámetros, como la letra, el formato de la letra, el color de fondo, etc., para constituir un ambiente de trabajo cómodo que mejore la sesión educativa. Debido a que el diseño depende en gran medida del usuario, este se convierte en una parte importante de la interfaz [9].

1.2. Arquitectura de Software

La Arquitectura de Software (AS) tiene sus inicios desde la década de 1960, donde el holandés Edsger Dijkstra[2], de la Universidad Tecnológica de Eindhoven en Holanda, propuso que los sistemas fueran estructurados correctamente antes de ser programados. En 1992, surge originalmente el término "arquitectura de software". Sus autores Perry y Wolf definen tres componentes: *elementos, forma* y *razón*.

La arquitectura no se inscribe en ninguna metodología de desarrollo sino que es una disciplina que se desarrolla durante todo el ciclo de desarrollo del software constituyendo la organización del sistema al nivel más alto de abstracción.

La arquitectura de software abarca decisiones importantes sobre: [10]

- La organización del sistema de software.
- Los elementos estructurales que compondrán el sistema y de interfaces, junto con sus comportamientos, tal y como se especifica en las colaboraciones entre estos elementos.

[2] Dijkstra estudió física teórica en la Universidad de Leiden. Entre sus contribuciones a la informática está el algoritmo de caminos mínimos; también conocido como Algoritmo de Dijkstra.

- La composición de estos elementos estructurales y del comportamiento en subsistemas progresivamente más grandes.
- El estilo de la arquitectura que guía esta organización, los elementos y sus interfaces, sus colaboraciones y su composición.

Sin embargo, la arquitectura de software está afectada no sólo por la estructura y el comportamiento del sistema, sino también por el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de comprensión, las restricciones y compromisos económicos y tecnológicos, y la estética.[10]

Todo sistema de software tiene una arquitectura. Una buena arquitectura sienta las bases para un sistema exitoso. Una mala arquitectura generalmente se convierte en resultados no favorables, teniendo luego que invertir tiempo y recursos innecesarios e imprevistos para continuar con el desarrollo del sistema.

Arquitectura de software en RUP

La arquitectura se representa mediante vistas o modelos que encierran la información necesaria para establecer la línea base a partir de la cual se desarrollará el sistema: una vista del modelo de casos de uso, una vista del modelo de análisis, una vista del modelo de diseño, etc. Este conjunto de vistas concuerda con las 4+1 vistas discutidas en [11].

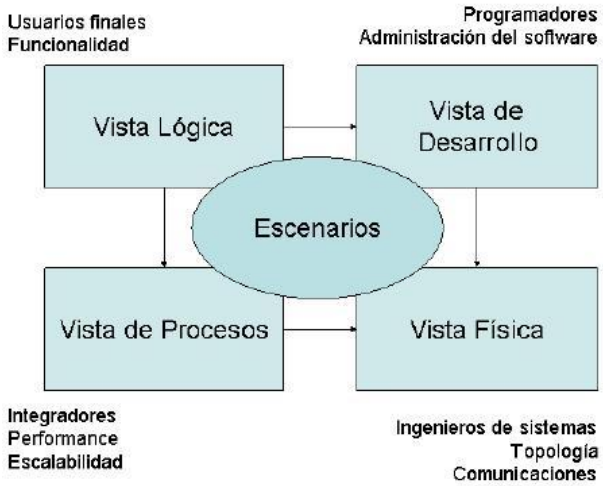

Figura 2. Modelo de "4+1" vistas

Se puede organizar la descripción de las decisiones de arquitectura en estas cuatro vistas, y luego ilustrarlas con un conjunto reducido de casos de uso o escenarios, los cuales constituyen la quinta vista. La arquitectura evoluciona parcialmente a partir de estos escenarios.

1.3. Estilos Arquitectónicos

Desde el preciso momento en que surge la AS se observaron algunas irregularidades de configuración en la práctica del diseño y la implementación que aparecían en ocasiones reiteradas respondiendo a situaciones similares. A estas muy pronto se les llamó estilos, debido a la semejanza con el uso del término en la arquitectura de edificios.

En 1992, Dewayne Perry y Alexander Wolf establecen que: [12]

> *"Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales"*

Mientras que para Mary Shaw y Paul Clements [13] los estilos arquitectónicos son un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo. Los componentes, incluyendo los subsistemas encapsulados, se pueden distinguir por la naturaleza de su computación: por ejemplo, si retienen estado entre una invocación y otra, y de ser así, si ese estado es público para otros componentes.

Los estilos sirven para sintetizar estructuras de soluciones, o sea, tener un lenguaje que describa la estructura de la solución que luego será refinada a través del diseño. Toda estructura de una aplicación corresponde a un estilo bien conocido. Los estilos definen los patrones posibles de las aplicaciones y aplican los conocimientos para evitar errores arquitectónicos comunes. La clave del trabajo arquitectónico se encuentra en la elección del estilo adecuado.

A continuación se describen los estilos más representativos y vigentes en el mundo de la AS.

Estilo Modelo Vista Controlador (MVC).

Separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes: Modelo, Vista y Controlador.[14]

Estilos Centrados en Datos.

Esta familia de estilos enfatiza la integrabilidad de los datos. Se estima apropiada 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.

Estilos de Llamada y Retorno.

Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en 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.

Estilo Arquitecturas en Capas

Garlan y Shaw [15] definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.

1.4. Patrones

Para Christopher Alexander: [16]

> *"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo siquiera dos veces de la misma forma"*

Basándose en la definición de Alexander y ya referida ya al ámbito del software, Buschman propone: [17]

> *"Un patrón describe un problema de diseño recurrente, que surge en contextos específicos de diseño, y presenta un esquema genérico probado para la solución de este. El esquema de la solución describe*

un conjunto de componentes, responsabilidades y relaciones entre de éstos, y formas en que dichos componentes colaboran entre sí."

Son soluciones generales a problemas comunes, además describen parte del sistema. Entre los principales patrones se encuentran: [18]

- Patrones de arquitectura: Son los relacionados a la interacción de objetos dentro o entre los niveles arquitectónicos. Se aplican durante la fase de diseño inicial para resolver problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, performance (rendimiento), modularidad y acoplamiento. Constituyen patrones de llamada entre objetos, similar a los patrones de diseño, decisiones y criterios arquitectónicos.
- Patrones de diseño: Se aplican durante la fase de diseño detallado para solucionar problemas de claridad del diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes.
- Patrones de análisis: Usualmente son específicos de la aplicación. Se aplican durante el análisis para solucionar problemas relacionados con el modelo de dominio, completitud, integración y equilibrio de objetivos múltiples, planeamiento para capacidades adicionales comunes.

1.4.1. Patrones Arquitectónicos

Su relación con los estilos arquitectónicos es perceptible, pero indirecta y variable incluso dentro de la obra de un mismo autor. Constituyen plantillas para arquitecturas de software concretas. Especifican las propiedades de la estructura global del sistema y tienen un impacto en la arquitectura de cada subsistema. La selección de un patrón de arquitectura o la combinación de varios es solo el primer paso cuando se diseña la arquitectura de un sistema software.

Buschmann propone que los patrones de arquitectura "*expresan un esquema de organización estructural para los sistemas de software. Proporcionan un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y lineamientos para organizar la relación entre ellos*".

Entre los principales patrones arquitectónicos se encuentran los que a continuación se describen.

Patrón arquitectónico Vista de Plantilla

La idea básica de la vista de plantilla es incrustar marcadores en una página HTML cuando ésta es escrita. Cuando la página se utiliza para solicitar un servicio, los marcadores se sustituirán por los

resultados obtenidos de acuerdo con el servicio que brinda la página, como por ejemplo, el resultado de una consulta obtenida en una base de datos.

Patrón arquitectónico Página de Control

La idea básica detrás de una página de control es tener un módulo en el servidor web, dígase un controlador para cada página en el sitio web. En la práctica, no funciona exactamente un controlador por página, ya que a veces se le puede realizar un enlace a cualquier página dinámica mediante alguna función de información dinámica.

Patrón arquitectónico Modelo Vista Controlador

Separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes [14]:

- Modelo: Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- Vista: Maneja la visualización de la información.
- Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

1.4.2. Patrones de Análisis

Los patrones de análisis son un conjunto de clases y relaciones entre ellas, que tienen algún sentido en el contexto de una aplicación. Representan una estructura que puede ser válida para otras aplicaciones.[19]

Describen un conjunto de prácticas destinadas a elaborar modelos de los conceptos principales de la aplicación que se va a construir. La intención principal de estos patrones es ayudar a las personas que realizan el trabajo de modelado, pues no siempre tienen experiencia al respecto y, en la mayoría de los casos, construyen sus modelos sin referencia alguna.

1.4.3. Patrones de Diseño

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. [17]

- **Patrón GRASP**

GRASP (*General Responsibility Assignment Software Patterns*) es un acrónimo que llevado al español corresponde a los Patrones Generales de Software de Asignación de Responsabilidades. Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. A continuación se describen los patrones básicos de asignación de responsabilidades:

Patrón Experto

Surge como principio fundamental que hay que tener en cuenta siempre cuando se esté asignando una responsabilidad a una clase. La respuesta es asignar la responsabilidad a la clase que contenga la información necesaria para cumplir la responsabilidad, o sea, la clase debe ser la experta en la información. La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos).

Patrón Creador

La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. En consecuencia, conviene contar con un principio general para asignar las responsabilidades concernientes a ella. Por tanto ¿Quién debería ser responsable de crear una nueva instancia de alguna clase?

Este patrón es muy simple y su mayor beneficio es que contribuye a soportar el bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización.

Patrón Bajo Acoplamiento

El acoplamiento es una medida de la fuerza con que una clase está relacionada a otras clases. Una clase con bajo o débil acoplamiento no depende de muchas otras. Por el contrario, una clase con alto o fuerte acoplamiento recurre a muchas otras. Este tipo de clases no es conveniente, porque un cambio en las

clases que utiliza ocasionaría cambios locales en la clase dependiente, además son más difíciles de entender cuando están aisladas y son más difíciles de reutilizar porque se requiere la presencia de otras clases de las que dependen.

Patrón Controlador

¿Quién debería encargarse de atender un evento del sistema?

Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas.

Asignar la responsabilidad del manejo de mensajes de los eventos del sistema a una clase que represente alguna de las siguientes opciones:

- El sistema global.
- La empresa u organización global.
- Algo activo en el mundo real que pueda participar en la tarea.
- Un manejador artificial de todos los eventos del sistema de un caso de uso (controlador de casos de uso)

Patrón Alta Cohesión

Cada elemento del diseño debe realizar una labor única dentro del sistema, lo cual expresa que la información que almacena una clase debe de ser coherente y está en la mayor medida de lo posible relacionada con la clase. . La solución es asignar a una clase responsabilidades que trabajen sobre una misma área de la aplicación y que no tengan mucha complejidad.

Además, existen cuatro patrones GRASP adicionales:

- Fabricación Pura.
- Polimorfismo.
- Indirección.
- No hables con extraños.

- **Patrones GoF**

El catálogo de patrones de diseño más famoso es el contenido en el libro bien conocido Libro GOF (*Gang-Of-Four Book*) [20]. Según este libro existen 3 tipos de patrones:

1. De Creación: Muestran la guía de cómo crear objetos cuando sus creaciones requieren tomar decisiones. Estas decisiones normalmente serán resueltas dinámicamente decidiendo qué clases instanciar o sobre qué objetos un objeto delegará responsabilidades.
2. Estructurales: Describen la forma en que diferentes tipos de objetos pueden ser organizados para trabajar unos con otros.
3. De Comportamiento: Se utilizan para organizar, manejar y combinar comportamientos

Patrones Creacionales	Patrones Estructurales	Patrones de Comportamiento
• Abstract Factory (Fábrica Abstracta) • Builder (Constructor Virtual) • Factory Method (Método de fabricación) • Prototype (Prototipo) • Singleton (Instancia única)	• Adapter (Adaptador) • Bridge (Puente) • Composite (Objeto Compuesto) • Decorator (Envoltorio) • Facade (Fachada) • Flyweight (Peso Ligero) • Proxy	• Chain of Responsibility (Cadena de responsabilidad) • Command (Orden) • Interpreter (Intérprete) • Iterator (Iterador) • Mediator (Mediador) • Memento (Recuerdo) • Observer (Observador) • State (Estado) • Strategy (Estrategia) • Template Method (Método plantilla) • Visitor (Visitante)

Tabla 1. Tabla de patrones GoF.

1.5. Metodología de desarrollo

El desarrollo de un software se torna cada vez más difícil teniendo en cuenta la complejidad que pueda presentar el mismo. Otro factor importante es la necesidad de que los desarrolladores posean un único lenguaje de entendimiento, el cual influye en la calidad del trabajo y a su vez en la satisfacción del cliente en cuanto al resultado. La aplicación de una metodología sería una solución para lograr lo anteriormente expresado.

Cuando se trata de proyectos pequeños (dos o tres meses de duración aproximadamente) no se suele tener en cuenta el uso de una apropiada metodología. Por otra parte, si los proyectos a desarrollar son de una mayor complejidad, es imprescindible tener una metodología de desarrollo que se adecue a estos; ya sea una existente o una diseñada por los propios desarrolladores.

Las metodologías de desarrollo de software se han clasificado en dos grupos: las ágiles y las tradicionales. Las metodologías *ágiles* enfatizan la comunicación con el cliente mientras que suelen ser criticadas por la falta de documentación técnica. Aquellas con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, reciben el apelativo de metodologías *tradicionales o pesadas* [21].

Entre las metodologías *ágiles* se destacan XP (*eXtreme Programming*), OpenUP (*Open Unified Process*), Scrum y Crystal. Así mismo se puede mencionar RUP (*Rational Unified Process*) y MSF (Microsoft Solution Framework) como las más destacadas en las metodologías *pesadas*.

A continuación se abordarán características fundamentales de dos de las metodologías más usadas actualmente. Como principal ejemplo de las metodologías ágiles estará XP, mientras que RUP será la representante de las metodologías tradicionales por ser una de las más generalizadas y usadas en estas.

eXtreme Programing

XP (*eXtreme Programming, Programación Extrema*) se ha convertido en la metodología ágil más popular, y posiblemente en la más transgresora de la ortodoxia basada en procesos. Su creador fue Kent Beck, impulsor del manifiesto ágil [22]. XP consiste en una programación rápida o extrema, cuyo rasgo característico es tener como parte del equipo de desarrollo al usuario final (cliente), pues es uno de los requisitos para llegar al éxito del proyecto.[23]

Esta programación considera que con un poco de planificación, codificación y unas pruebas, se puede decidir si se está siguiendo el camino correcto o el equívoco, evitando una marcha atrás demasiado tarde.

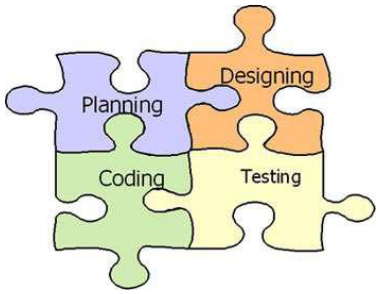

Figura 3. Metodología eXtreme Programming

Esta metodología se basa en: [23]

- *Pruebas Unitarias:* Pruebas realizadas a los principales procesos.
- *Refabricación*: Reutilización de código. Son creados patrones o modelos estándares que traen consigo una mayor flexibilidad al cambio.
- *Programación en pares*: Dos desarrolladores participan 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.

El ciclo de desarrollo consiste (agrandes rasgos) en los siguientes pasos:[24]

- El cliente define el valor de negocio a implementar.
- El programador estima el esfuerzo necesario para su implementación.
- El cliente selecciona qué construir, de acuerdo a sus prioridades y las restricciones de tiempo.
- El programador construye ese valor de negocio.
- Vuelve al paso 1.

El ciclo de vida ideal de XP consiste de seis fases: Exploración, Planificación de la Entrega (*Release*), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto. [24]

Rational Unified Process

Rational Unified Process, o RUP, es una plataforma flexible de procesos de desarrollo de software que ayuda brindando guías consistentes y personalizadas de procesos para todo el equipo de proyecto. RUP describe cómo utilizar de forma efectiva reglas de negocio y procedimientos comerciales probados en el desarrollo de software para equipos de desarrollo de software, conocidos como “mejores prácticas”. Captura varias de las mejores prácticas en el desarrollo moderno de software en una forma que es

aplicable para un amplio rango de proyectos y organizaciones. Es una guía de cómo utilizar de manera efectiva UML (*Unified Modeling Language*). Provee a cada miembro del equipo fácil acceso a una base de conocimiento con guías, plantillas y herramientas para todas las actividades críticas de desarrollo. Crea y mantiene modelos, en lugar de enfocarse en la producción de una gran cantidad de papeles de documentación.[21]

Los autores de RUP destacan que el proceso de software propuesto por RUP tiene tres características esenciales [25]:

- Proceso dirigido por casos de uso.
- Está centrado en la arquitectura.
- Es iterativo e incremental.

Una de las mejores prácticas centrales de RUP es la noción de desarrollar iterativamente. RUP organiza los proyectos en términos de disciplinas y fases, consistiendo cada una en una o más iteraciones. Con esta aproximación iterativa, el énfasis de cada flujo de trabajo variará a través del ciclo de vida.[21]

La metodología RUP divide en 4 fases el desarrollo del software [23]:

- Inicio: Determinarla visión del proyecto.
- Elaboración: Determina la arquitectura óptima.
- Construcción: Obtiene la capacidad operacional inicial.
- Transmisión: Obtiene .el *release*[3] del proyecto

Cada una de estas fases se desarrolla mediante el ciclo de iteraciones, las cuales tienen sus objetivos en función de la evaluación de las iteraciones precedentes. El ciclo de vida que se desarrolla por cada iteración es llevada bajo dos disciplinas: [23]

1. Disciplina de Desarrollo: Modelamiento del Negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas.
2. Disciplina de Soporte: Despliegue, Administración de configuración y cambios, Administración de Proyecto, Ambiente.

[3] Release: liberación, puesta en circulación.

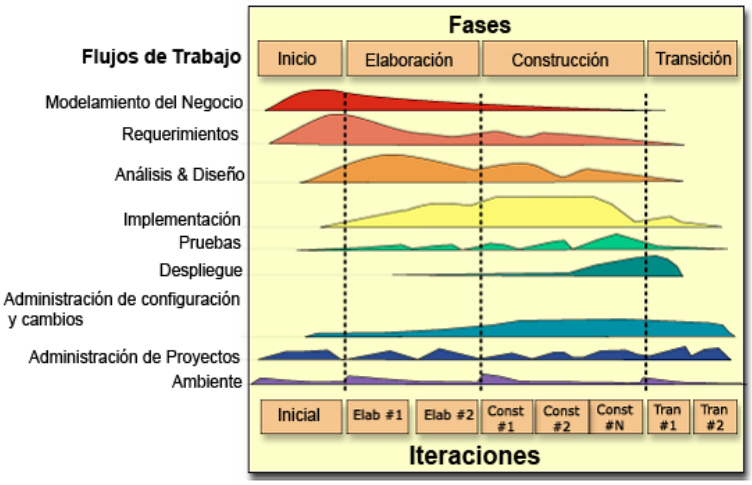

Figura 4. Fases e iteraciones de la metodología RUP

Selección de la metodología a utilizar

Luego del estudio de las metodologías más utilizadas mundialmente, se llega ha llegado a la conclusión que la más indicada para guiar el desarrollo de este sistema es RUP.

RUP es una plataforma flexible de procesos de desarrollo de software que ayuda brindando guías consistentes y personalizadas de procesos para todo el equipo de proyecto. Es una guía de cómo utilizar de manera efectiva UML. Provee a cada miembro del equipo fácil acceso a una base de conocimiento con guías, plantillas y herramientas para todas las actividades críticas de desarrollo. Crea y mantiene modelos, en lugar de enfocarse en la producción de una gran cantidad de papeles de documentación.

Una de las mejores prácticas centrales de RUP es la noción de desarrollar iterativamente. La aproximación iterativa ayuda a mitigar los riesgos en forma temprana y continua, con un progreso demostrable y frecuentes *releases* ejecutables. Además provee un entorno de proceso de desarrollo configurable basado en estándares; permite tener claro y accesible el proceso de desarrollo que se sigue y que este sea configurado a las necesidades de la organización y del proyecto.

1.6. Lenguaje de Programación

Un lenguaje de programación es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Permite crear programas mediante un conjunto de instrucciones, operadores

y reglas de sintaxis. Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Por lo tanto, un lenguaje de programación es un modo práctico para que los programadores puedan dar instrucciones a un equipo.

Java

Java es un lenguaje de programación orientado a objetos desarrollado por *Sun Microsystems* con el cual se puede realizar cualquier tipo de programa. Entre sus características resaltan: poseer un mecanismo robusto de manejo eficiente de excepciones, ser fuertemente tipiado, la ausencia de los punteros, ser multiplataforma, el soporte para la programación asincrónica y la utilización del enlace dinámico en tiempo de ejecución. [26]

El poder reducir los problemas de acceso a memoria y liberación automática hacen de Java un lenguaje poco apropiado para desarrollar aplicaciones de base como Sistemas Operativos.

La compatibilidad es total:

- A nivel de fuentes: El lenguaje es exactamente el mismo en todas las plataformas.
- A nivel de bibliotecas: En todas las plataformas están presentes las mismas bibliotecas estándares.
- A nivel del código compilado: El código intermedio que genera el compilador es el mismo para todas las plataformas. Lo que cambia es el intérprete del código intermedio.

PHP

PHP (Hypertext Preprocessor, Preprocesador de Hipertexto) se trata de un lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor. Está muy orientado al desarrollo de aplicaciones web y permite insertar contenidos dinámicos en las páginas.[27]

- Es multiplataforma. Puede ser utilizado en cualquiera de los principales sistemas operativos incluyendo Linux, variantes Unix (HP-UX, Solaris y OpenBSD), Microsoft Windows, entre otros.
- *Soporta la mayoría de servidores web.* Incluyendo Apache, Microsoft Internet Information Server, Personal Web Server, Netscape, y muchos otros.

- No se encuentra limitado a resultados en HTML. Puede presentar resultados como XHTM y archivos XML; autogenerarlos y luego almacenarlos en el sistema de archivos en vez de presentarlos en la pantalla.
- Soporta una gran cantidad de Bases de Datos, entre ellas: MySQL, ODBC, PostgreSQL, Oracle (OCI7 y OCI8), entre otras.

Tiene las siguientes ventajas:

- Capacidad de expandir su potencial utilizando la enorme cantidad de módulos (o extensiones).
- Posee una amplia documentación.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite las técnicas de Programación Orientada a Objetos.
- Biblioteca nativa de funciones sumamente amplia e incluida.
- No requiere definición de tipos de variables.
- Tiene manejo de excepciones (desde PHP5).

1.7. Frameworks de Desarrollo

Los frameworks de desarrollo son soluciones que implementan patrones y ayudan a desarrollar funciones dentro de las aplicaciones a un nivel superior. Abstraen de complejas implementaciones y hacen el desarrollo más rápido, ya que proveen soluciones a problemas comunes a la hora de lidiar con la tecnología, son como un conjunto de librerías que se usan en las aplicaciones.

Zend Frameworks

Zend Framework se trata de un framework para desarrollo de aplicaciones Web y servicios Web con PHP, te brinda soluciones para construir sitios web modernos, robustos y seguros. Además es Open Source y trabaja con PHP 5. Su principal ventaja es que es desarrollado por Zend[4].[28]

Sus principales características son:

- Trabaja con MVC (Model-View-Controller)
- Cuenta con módulos para manejar archivos PDF, canales RSS, Web Services, entre otros.

[4] Zend: Empresa que respalda comercialmente a PHP.

- El Marco de Zend también incluye objetos de las diferentes bases de datos, por lo que es extremadamente simple para consultar su base de datos, sin tener que escribir ninguna consulta SQL.
- 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.

Symfony

Es un framework para el desarrollo de aplicaciones web de forma rápida. Es la respuesta de PHP para Ruby on Rail[5]. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Symfony está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel.

Symfony utiliza los siguientes componentes:

- Propel: ORM[6] para el acceso a Base de datos.
- Creole: Maneja la capa de abstracción de BD.
- Phing: Mapeador XML.
- Pake: Gestión de ejecución de Scripts.

A continuación se muestran algunas de sus características [29]:

- 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
- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos.

[5] Ruby on Rails, también conocido como RoR o Rails es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC).

[6] ORM: Object-Relational Mapping, mapeo de objetos a Base de Datos.

- Basado en la premisa de "convenir en vez de configurar", en la que el desarrollador solo debe configurar aquello que no es convencional
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web
- Preparado para aplicaciones empresariales y adaptable 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
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros.

1.8. Sistema Gestor de Base de Datos

Un Sistema Gestor de Base de Datos (SGBD) es un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Tienen como propósito general el manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante para una organización.

PostgreSQL

Es un servidor de bases de datos relacional libre, ofrecido bajo la licencia BSD (*Berkeley Software Distribution*) [30]. PostgreSQL incluye características de la orientación a objetos, como puede ser: la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional.

Las principales características de PostgreSQL son: [30]

- Atomicidad: Asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.
- Consistencia: Asegura que sólo se empiece aquello que se puede acabar. Se ejecutan las operaciones que no van a romper la reglas y directrices de integridad de la base de datos.
- Aislamiento: Asegura que una operación no puede afectar a otras. Esto asegura que dos transacciones sobre la misma información nunca generará ningún tipo de error.
- Durabilidad: Asegura que una vez realizada la operación, esta persistirá y no se podrá deshacer.
- Soporte de casi todos los sistemas operativos: Linux, Unix, Mac OS, Beos, Windows, etc.

- Documentación muy bien organizada, pública y libre.
- Altamente adaptable a las necesidades del cliente
- Soporte nativo para los lenguajes más populares del medio: PHP, C, C++, Perl, Python.
- Soporte de todas las características de una base de datos profesional.

MySQL

Es un SGBD relacional que almacena la información en tablas separadas en vez de ponerla toda en un gran almacén; añadiéndole velocidad y flexibilidad en su funcionamiento. Se encuentra liberados bajo la licencia GPL (G*eneral Public License*). Es catalogado como el gestor más rápido. Puede trabajar en modo cliente/servidor o en sistemas embebidos.

Las principales características de MySQL son:

- Trabaja en varias plataformas.
- Completamente multihilo, utilizando hilos del núcleo. Puede fácilmente utilizar múltiples CPU en caso de que estén disponibles.
- Las funciones SQL están implementadas utilizando una librería de clases optimizada para que sea tan rápida como sea posible. Usualmente no hay asignación de memoria.
- Provee motores de almacenamientos transaccionales y no transaccionales.

1.9. Herramienta CASE[7]

Una herramienta CASE consiste en una o varias herramientas que permiten organizar y manejar cierta información de un proyecto informático.

Visual Paradigm

Visual Paradigm es una herramienta que soporta el ciclo de vida completo del desarrollo de software. Este software de modelado UML ayuda a que la construcción de aplicaciones de calidad sea más rápida, mejor y un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

[7] CASE: Computer Aided Software Engineering (Ingeniería de Software Asistida por Computación).

Las principales características son:

- Licencia: Gratuita y Comercial.
- Soporta aplicaciones Web.
- Varios idiomas.
- Fácil de instalar y actualizar.
- Soporta notación UML 2.x.
- Importación desde Rational Rose
- Generación de código e ingeniería inversa a la vez de los lenguajes: Java, C++, CORBA IDL, PHP, XML Schema, Ada y Python.
- Soporta la generación de código en: C#, VB .NET, Object Definition Language (ODL), Flash ActionScript, Delphi, Perl, Objective-C y Ruby.
- Soporte de ingeniería inversa de: Java class, .NET (dll, exe), JDBC y ficheros mapeados de Hibernate

Rational Rose

Rational Rose es una herramienta CASE que da soporte al modelado visual con UML ofreciendo distintas perspectivas del sistema [31]. Da soporte a RUP para el desarrollo de los proyectos de software, desde la etapa de Ingeniería de Requerimientos hasta la etapa de pruebas. Para cada una de estas etapas existe una herramienta que ayuda en la administración de los proyectos, Rose es la herramienta de Rational para la etapa de análisis y diseño de sistemas. [21]

- Modelado de Negocio
- Captura de Requisitos (parcial)
- Análisis y Diseño (Completo)
- Implementación (como ayuda)
- Control de Cambio y gestión de configuración.

Rational Rose ofrece un diseño dirigido por modelos que redundan en una mayor productividad de los desarrolladores, admitiendo el lenguaje de modelado UML y técnicas de modelado de objetos (OMT). Utiliza un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación. Posee capacidades de ingeniería inversa. Además se encuentra disponible en múltiples plataformas.

1.10. Conclusiones del Capítulo

Como resultado de la investigación en este capítulo, se puede concluir que:

- Se realizó un estudio de las Arquitecturas de Software de los STI.
- Del estudio realizado a los estilos arquitectónicos se definió como estilo rigente el Modelo - Vista - Controlador, incluyendo patrones arquitectónicos que contribuirán una exitosa solución.
- La información recopilada sobre las herramientas y tecnologías constituye la base para la toma de decisiones respecto a la manera de estructurar la solución.

Capítulo 2. Descripción Arquitectónica

Este capítulo tiene como propósito describir la arquitectura definida para la aplicación VIRTEVALL con el objetivo de solucionar el problema planteado en el marco teórico. Se confeccionan los artefactos acorde a la metodología RUP para el proceso de desarrollo. Estos artefactos constituyen 4 de las 5 vistas arquitectónicas donde se representa la arquitectura en casos de usos, paquetes, subsistemas, componentes y demás elementos que posibiliten un entendimiento común entre los desarrolladores.

2.1. Representación Arquitectónica

Para la realización de la arquitectura de VIRTEVALL se tuvo en cuenta el diseño de las capas lógicas a través de una propuesta de diseño base. Ésta brinda una estructura física capaz de soportar el código, creando un esqueleto base definido por el framework a utilizar. Se basa en los estilos de llamada y retorno, fundamentalmente el referente al MVC, debido a que esta familia de estilos enfatiza la modificabilidad, la escalabilidad, la reusabilidad y la usabilidad del sistema.

Mediante la metodología de desarrollo RUP, estará guiada la representación arquitectónica expuesta. Esta metodología representa la arquitectura haciendo uso del Modelo 4+1 vista. Es importante señalar que la vista de procesos no se representa debido a que en el sistema no existen procesos concurrentes significativos.

El sistema estará dividido en módulos, para lograr reusabilidad y facilitar el mantenimiento del mismo ante cambios que se puedan realizar en los procesos de negocio:

1. Módulo Autenticación: En gran parte, la seguridad del sistema se encuentra en la autenticación. Cada usuario posee un acceso de acuerdo al rol que desempeñe. Este módulo es el responsable de la autenticación y el registro a la aplicación.
2. Módulo Actividades: En este módulo se encuentran las actividades que realizará el estudiante. Estas actividades son todas las propuestas por el profesor: encuestas, diagnósticos, ejercicios. Además, le ofrece al estudiante el método de enseñanza que se corresponde a sus necesidades.
3. Módulo Administración: Es la encargada de gestionar los usuarios que van a administrar la aplicación: estudiantes y profesores. Los estudiantes tendrán por defecto un rol de usuario que

sólo les permitirá interactuar con la aplicación, con el mínimo acceso permisible. A los profesores se les asignará un rol, de acuerdo a este tendrán acceso a determinadas acciones.

4. Módulo Configuración: Se encarga de realizar el diseño previo de los diagnósticos, encuestas, actividades. Tiene como principal participante al profesor, el cual de acuerdo al rol que desempeña crea y/o configura las actividades. Contiene el conocimiento del profesor, y las reglas de producción fundamentalmente.
5. Módulo Presentación: Esta es la que permite una interacción del usuario con el sistema. Contiene la página de inicio, donde se le presenta al usuario noticias, informaciones, etc. Además, le presenta al usuario todo el contenido del Módulo Configuración.
6. Módulo Diagnóstico: Contiene el diagnóstico inicial general que se le realiza al estudiante, lo cual permite conocer el nivel que posee el estudiante en cuanto al idioma. Estas preguntas son adquiridas de los resultados arrojados por la encuesta.
7. Módulo Estudiante: Es la que representa todas las acciones que realiza el estudiante. Incluye las características individuales, además del perfil de usuario que se crea dado su conocimiento en cuanto a las actividades propuestas por el profesor.
8. Módulo Encuesta: Contiene las encuestas que permiten conocer las características de los estudiantes, sus preferencias, estilos de aprendizajes.

Restricciones de Diseño

- El Lenguaje de Programación será: PHP 5.0.
- El framework base de desarrollo que se utilizará es: Symfony 1.2.8 y sub-frameworks asociados.
- El SGDB deberá ser PostgreSQL 8.3.

El repositorio principal, el entorno de prueba y el servidor de Base de Datos estarán montados sobre Windows.

2.2. MVC en Symfony

Symfony toma lo mejor de la arquitectura MVC y la implementa de forma tal que el desarrollo de aplicaciones sea rápido y sencillo.

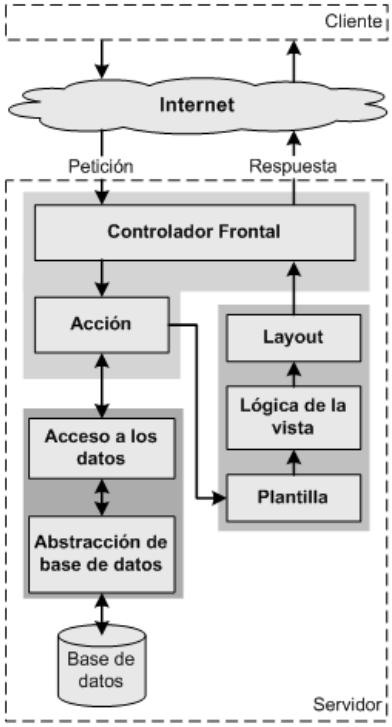

Figura 5. Flujo de trabajo de Symfony

El *controlador frontal* y el *layout* son comunes para todas las acciones de la aplicación. Se pueden tener varios controladores y varios *layouts*, pero solamente es obligatorio tener uno. El controlador frontal es un componente que sólo tiene código relativo al MVC, por lo que no es necesario crear uno, ya que Symfony lo genera de forma automática. [29]

Las clases de la *capa del modelo* también se generan automáticamente, en función de la estructura de datos de la aplicación. La librería Propel se encarga de esta generación automática, ya que crea el *esqueleto* o estructura básica de las clases y genera automáticamente el código necesario. [29]

La *abstracción de la base de datos* es completamente transparente para el programador, ya que se realiza de forma nativa mediante Creole. Así, si se cambia el sistema gestor de bases de datos en cualquier momento, no se debe reescribir ni una línea de código, ya que tan sólo es necesario modificar un parámetro en un archivo de configuración. [29]

Por último, la lógica de la vista se puede transformar en un archivo de configuración sencillo, sin necesidad de programarla.

2.3. Patrones de diseño utilizados

- **Patrones GRASP**

Creador

Las acciones definidas para cada módulo se encuentran en la clase *nombremoduloActions.* Las acciones contienen toda la lógica de la aplicación. En las acciones se crean los objetos de las clases que representan las entidades y de los objetos de otras clases que intervienen en la lógica del módulo.

Experto

Este patrón es uno de los más utilizados. La implementación que realiza Symfony de la arquitectura MVC incluye varias clases, en las cuales se evidencia el uso de este patrón:

- sfController: Clase perteneciente al controlador frontal que se encarga de decodificar la petición y transferirla a la acción correspondiente.
- sfRequest: Almacena los elementos que forman la petición (parámetros, cookies, cabeceras, etc.)
- sfResponse: Contiene las cabeceras de la respuesta y los contenidos. El contenido de este objeto se transforma en la respuesta HTML que se envía al usuario.

Alta Cohesión

En cada clase *Actions* se definen las acciones para las plantillas, además estas colaboran con otras para realizar diferentes operaciones, se instancian objetos, se acceden a las *properties*; o sea, en una clase *Actions* se utilizan diferentes funcionalidades estrechamente relacionadas entre sí, lo que proporciona un software flexible ante los cambios. Symfony agrupa las clases por funcionalidades que son fácilmente reutilizables, bien por su uso directo o por herencia.

Controlador

Todas las peticiones Web son manejadas por un solo controlador frontal (*sfActions*), el cual es el único punto de entrada de toda la aplicación en un determinado entorno. 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

A cada clase Symfony le asigna una responsabilidad, de forma tal que mantiene pocas dependencias entre las mismas.

- **Patrones GoF**

Singleton (Instancia única)

El patrón *Singleton* garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. A través del controlador frontal de Symfony se realiza una llamada a sfContext::createInstance(). En una acción, el método getContext() devuelve el singleton. Se trata de un objeto que guarda una referencia a todos los objetos del núcleo de Symfony relacionados con una petición dada, y ofrece un método que permite el acceso para cada uno de ellos:

- sfController: Objeto controlador (->getController())
- sfRequest: Objeto de la petición (->getRequest())
- sfResponse: Objeto de la respuesta (->getResponse())
- sfUser: Objeto de la sesión del usuario (->getUser())
- sfDatabaseConnection: Conexión a la base de datos (->getDatabaseConnection())
- sfLogger: Objeto para los logs (->getLogger())
- sfI18N: Objeto de internacionalización (->getI18N())
- Desde cualquier parte del código se puede llamar al singleton sfContext::getInstance().

Abstract Factory (Fábrica abstracta)

Este patrón permite trabajar con objetos de distintas familias de manera que éstas no se mezclen entre sí, haciendo transparente el tipo de familia concreta que se esté usando. Por ejemplo, cuando el framework necesita crear un nuevo objeto para una petición, pues busca en la definición de la factoría el nombre de la clase que se debe utilizar para esta tarea. La definición por defecto de la factoría para las peticiones es *sfWebRequest*, por lo cual Symfony crea un objeto de esta clase para tratar con las peticiones.

Decorator (Envoltorio)

Con el uso del *Decorator* se añade funcionalidad a una clase dinámicamente. El archivo layout.php (o plantilla global) almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el *layout*, lo cual implica que éste decora la plantilla.

Composite (Objeto compuesto)

El *Composite* permite tratar objetos compuestos como si de uno simple se tratase. Dada la composición recursiva y la estructura en forma de árbol que posee Symfony, se provee la construcción de objetos complejos a partir de otros más simples y similares entre sí; permitiendo que el tratamiento de los objetos creados se simplifique ya que al poseer todos una interfaz común se tratan de la misma manera.

2.4. Requisitos No Funcionales

Como anteriormente se ha visto, la AS está centrada en los requisitos no funcionales (RNF) de la solución. Estos RNF, se han de tener en cuenta para la implementación del sistema.

Requisitos de Seguridad

- La información manejada por el sistema podrá ser accedida por las personas que tienen los permisos para ello, por lo estará protegida de acceso no autorizado y divulgación.
- Los usuarios accederán a la información correspondiente a su rol.

Requisitos de Integridad

- Los datos contenidos en la Base de Datos no deberán ser redundantes o falsos.
- La información generada por el sistema será objeto de un alto nivel de protección contra estados inconsistentes y corrupción.

Requisitos de Disponibilidad

- A los usuarios autorizados se les deberá garantizar el acceso a la información solicitada en todo momento.

Requisitos de Confiabilidad

- Se garantizará la llegada de los datos de forma íntegra y segura al destino.
- Los datos serán almacenados de forma segura en una Base de Datos utilizando métodos de encriptación.

Requisitos de Portabilidad

- La aplicación podrá ser usada en sistemas operativos Windows o Linux.

Requisitos de Usabilidad

- La aplicación que se utilizará debe ser lo más interactiva posible, brindará una interfaz simple y amigable para que el usuario no tenga dificultad al utilizarla.

Requisitos de Rendimiento

- El sistema debe ser lo más eficiente posible para poder lograr un tiempo de respuesta de 0.1 a 0.7 segundos ante la concurrencia de peticiones.

Requisitos de Soporte

- El sistema contará con un grupo de soporte destinado a brindar consultorías y asesoramiento técnico.
- Se confeccionarán manuales de usuarios y guías para una mejor comprensión del usuario.

Requisitos de Software

- El servidor central de procesamiento deberá tener instalado:
  - Sistema Operativo Windows o Linux.
  - Framework Symfony.
  - Servidor WAMP (Windows) o LAMP (Linux).
- Las PC Clientes deberán tener instalado:
  - Sistema Operativo Windows o Linux.
  - Navegadores Web Internet Explorer 5.0 o Mozilla Firefox 3.5.
- El servidor central de bases de datos deberá tener instalado:

  - Sistema Operativo Windows o Linux
  - Gestor de Base de Datos PostgreSQL 8.3.
- El servidor de respaldo de bases de datos deberá tener instalado:
  - Sistema Operativo Windows o Linux
  - Gestor de Base de Datos PostgreSQL 8.3.

Requisitos de Hardware

- Todas las computadoras deberán tener una tarjeta de red con una velocidad de transmisión de 100Mbps.
- El Servidor central de procesamiento requiere:
  - Memoria RAM 1 GB DDR2 (recomendado 8 GB)
  - Procesador de 1.90GHZ
  - 80 GB de capacidad de almacenamiento mínimo (recomendado 250 GB).
- Las PC Clientes requieren:
  - Memoria RAM 128MB DDR2 (recomendado 2 GB)
  - Procesador de 1.90GHZ
- El servidor central de bases de datos requiere:
  - Memoria RAM 1 GB DDR2 (recomendado 8 GB)
  - Procesador de 1.90GHZ
  - 80GB de capacidad de almacenamiento mínimo (recomendado 1 TB).
- El servidor de respaldo de bases de datos requiere:
  - Memoria RAM 1 GB DDR2 (recomendado 8 GB)
  - Procesador de 1.90GHZ
  - 80 GB de capacidad de almacenamiento mínimo (recomendado 1 TB).

2.5. Vistas Arquitectónicas

Las vistas arquitectónicas, a través de distintos niveles de abstracción, resaltan diversos aspectos que conciernen a los involucrados en el desarrollo. Resulta interesante analizar estas perspectivas de una arquitectura, y la forma en que éstas ayudan a todos los involucrados en el proceso de desarrollo a razonar sobre los atributos de calidad del sistema.

2.5.1. Vista de Casos de Uso

Esta vista brinda información de los casos de usos arquitectónicamente significativos, los cuales contienen las funcionalidades críticas para el sistema. Se refiere a críticos cuando son: funciones que son las más importantes, la razón de existir del sistema, o que tienen la mayor frecuencia de uso, o que presentan cierto riesgo técnico que debe ser mitigado, o que sirven para validar la arquitectura propuesta por el uso de la mayoría de los elementos de la misma. [11]

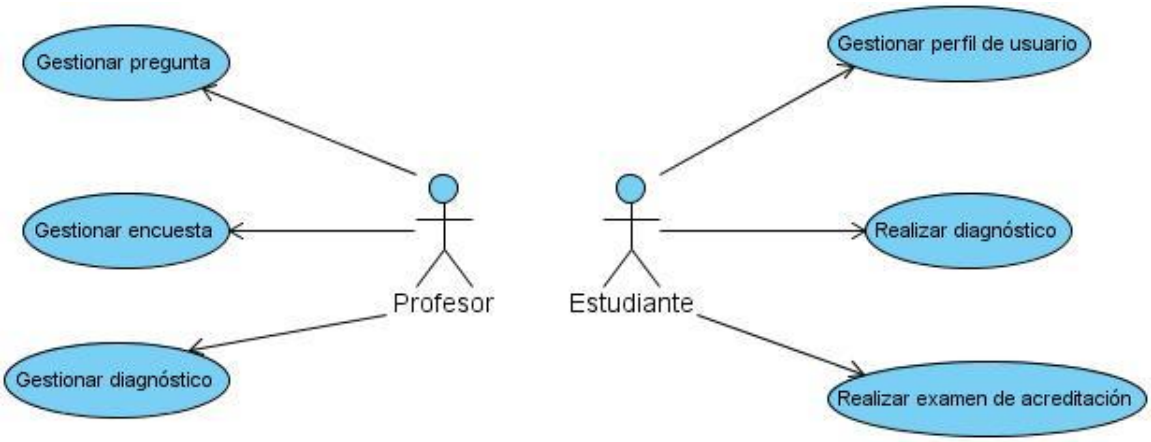

Figura 6. Diagrama de Casos de Uso.

Actores del Sistema

Profesor: Es la persona que contribuye a la facilitación del auto-aprendizaje de los estudiantes.

Estudiante: Es aquel que se beneficia con la interacción con la plataforma, ya que le permite realizar el auto-aprendizaje.

Casos de uso arquitectónicamente significativos

Gestionar preguntas: El caso de uso se inicia cuando el profesor selecciona la opción *Gestionar Preguntas*. El sistema realiza la acción seleccionada por el usuario y muestra las posibles opciones a seleccionar, finalizando el caso de uso.

Gestionar encuesta: El caso de uso se inicia cuando el profesor selecciona la opción *Gestionar Encuesta.* El sistema realiza la acción seleccionada por el usuario y muestra las posibles opciones a seleccionar, finalizando el caso de uso.

Gestionar diagnóstico: El caso de uso se inicia cuando el profesor selecciona la opción *Gestionar Diagnóstico*. El sistema realiza la acción seleccionada por el profesor y muestra las posibles opciones a seleccionar, finalizando el caso de uso.

Gestionar perfil de usuario: El caso de uso se inicia cuando el estudiante selecciona la opción *Crear Perfil*, al introducir usuario y contraseña y no encontrarse en la Base de Datos. Cuando el usuario desee modificar o visualizar su perfil, selecciona la opción *Gestionar Perfil de Usuario*. El sistema realiza la acción seleccionada por el usuario y muestra las posibles opciones a seleccionar, finalizando el caso de uso.

Realizar diagnóstico: El caso de uso se inicia cuando el estudiante selecciona la opción *Comenzar el Diagnóstico de Idioma*, lo responde y envía las respuestas del diagnóstico. El sistema realiza la acción seleccionada por el usuario y muestra el nivel obtenido por este, finalizando el caso de uso.

Realizar examen de acreditación: El caso de uso se inicia cuando el estudiante selecciona la opción *Seleccionar Actividades del Examen de Acreditación*. El sistema realiza la acción seleccionada por el usuario y muestra las posibles actividades a seleccionar, finalizando el caso de uso.

2.5.2. Vista Lógica

Esta vista describe las clases más importantes que formarán parte del ciclo de desarrollo del sistema de software. Se describen también los paquetes o subsistemas que conforman el sistema y las relaciones de dependencia o de uso que existen entre ellos. Esta descomposición en clases y subsistemas se hace para potenciar el análisis funcional y sirve para identificar mecanismos y elementos de diseño comunes a diversas partes del sistema.

Haciendo uso del MVC como arquitectura rigente para VIRTEVALL, se realiza una distribución por capas de acuerdo a la misma:

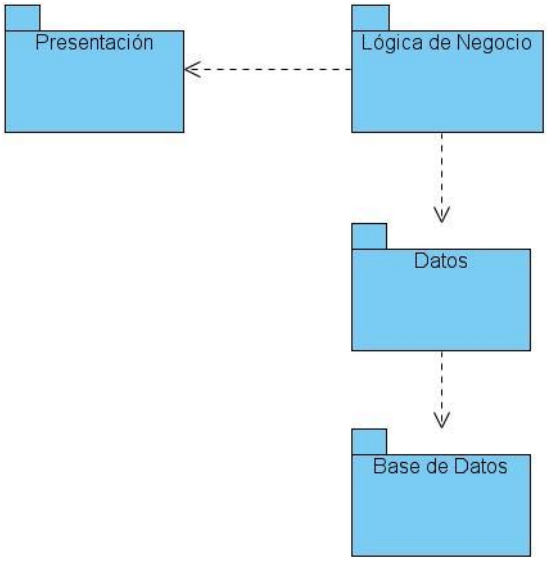

Figura 7 Distribución de la aplicación en capas usando MVC

Donde:

- Presentación (Vista): Conjunto de archivos que componen las páginas clientes del sistema.
- Lógica de Negocio (Controlador): Conjunto de clases y archivos de configuración encargados de gestionar todos los procesos de la lógica de negocio.
- Datos (Modelo): Conjunto de clases que controlan el tratamiento de los datos.
- Base de Datos: Conjunto de tablas relacionadas que contienen los datos de la aplicación.

Capa de Presentación (Vista)

La Capa de Presentación está dividida en dos partes:

1. La **vista global de la aplicación** contiene todo lo que es común para toda la aplicación: el layout (decora la plantilla del los módulos), las librerías ExtJS, los CSS y archivos .js con la presentación ExtJS de cada modulo:
    - **JS**: Conjunto de archivos y clases JavaScript comunes en toda la aplicación, además de los archivos .js de cada módulo.
    - **ext**: Librerías y clases con las API

- **Actividades, Administración, Autenticación, Configuración, Diagnóstico, Encuesta, Estudiante, Presentación:** Conjunto de archivos .js con la presentación ExtJS de cada uno de los módulos.
- **CSS**: Conjunto de las hojas de estilo común en toda la aplicación.
- **ext**: Contiene las hojas de estilo del ExtJS. (ext-all.css)
- **layout.php**: Contiene el código común a todas las páginas clientes.

2. La **vista de los módulos** contiene todas las plantillas de estos, donde se cargan los .js con la presentación ExtJS de cada uno, así como los slot, partial y los componentes.
   - **template**: Contiene las plantillas php en las que se cargará la presentación ExtJS del módulo correspondiente.

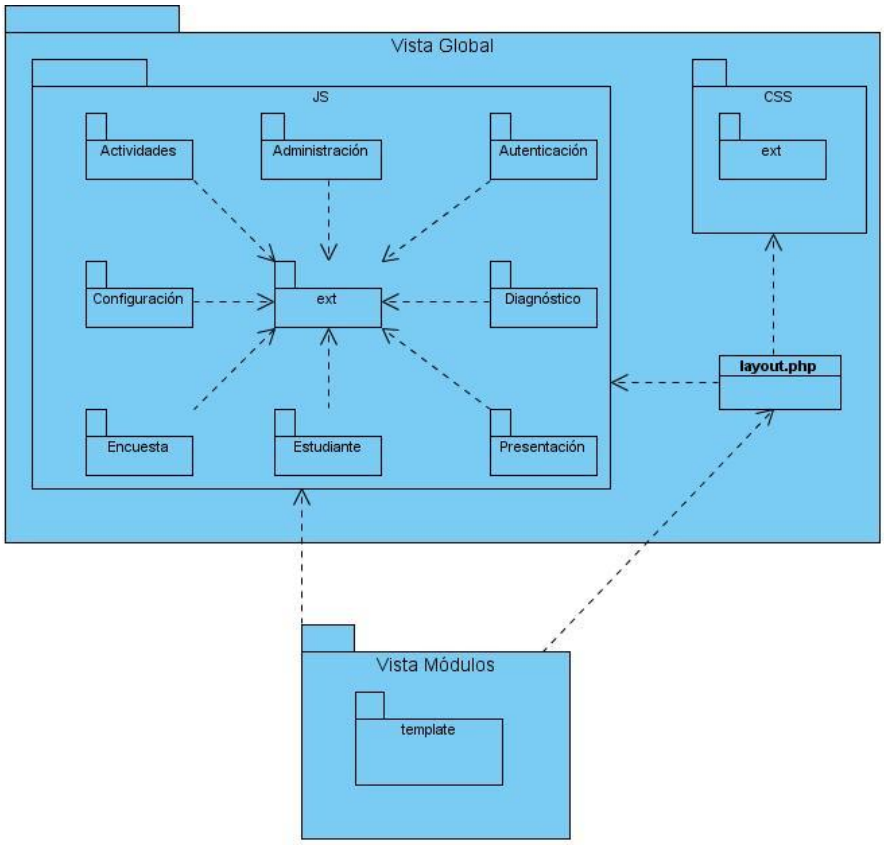

Figura 8. Vista Lógica - Capa Vista

Capa Lógica de Negocio (Controlador)

La Lógica de Negocio también se divide en dos partes:

1. El controlador de la Aplicación contiene los controladores frontales que son la vía de entrada y salida de la aplicación y los encargados de cargar todas las configuraciones de la aplicación.
   - **nombreaplicación_dev.php:** Controlador frontal para el entorno de desarrollo.

2. Los controladores de los módulos desarrollan toda la lógica del módulo.
   - **sfActions:** Clase del núcleo de Symfony, por la cual se accede a los objetos sfRequest, sfView, sfUser más usados.
   - **nombremoduloActions:** Clase responsable de la lógica de negocio del módulo.

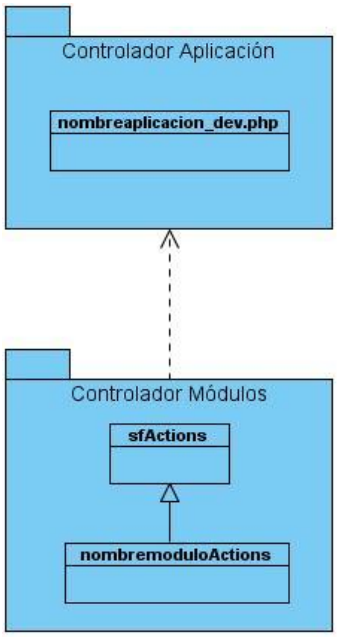

Figura 9. Vista Lógica - Capa Controlador

Capa de Datos (Modelo)

Propel: Contiene todas las clases de mapeo de la BD generada por el Propel.

Criteria: Clase del núcleo de Symfony, empleada para la elaboración de consultas SQL que interactúa con las clases del modelo de Propel.

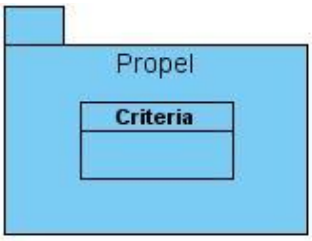

Figura 10. Vista Lógica - Capa Modelo

2.5.3. Vista de Despliegue

La vista de despliegue suministra una base para la comprensión de la distribución física de un sistema a través de nodos, y propone cómo se satisfacen los requisitos no funcionales de software y hardware e influye en el rendimiento.

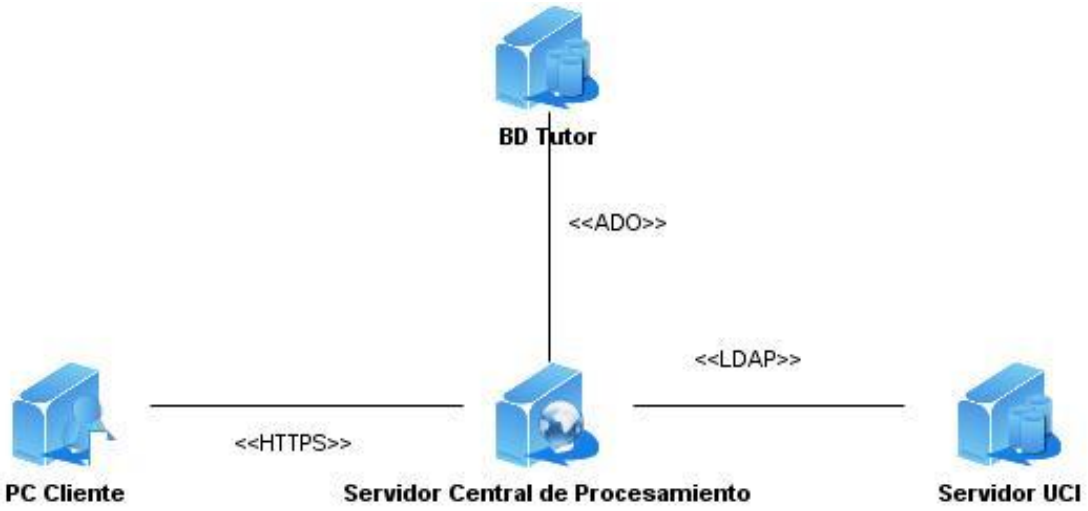

Figura 11. Diagrama de Despliegue

- **PC Cliente:** Este ordenador corresponde con el puesto de trabajo de los usuarios que utilizarán la aplicación, así como con el administrador del sistema.
- **Servidor Central de Procesamiento:** Nodo donde se encuentra la aplicación.
- **BD Tutor:** Servidor de BD en PostgreSQL que contiene la base de datos de la aplicación.
- **Servidor UCI:** Es el nodo al que la aplicación va acceder en el proceso de autenticación para consultar datos personales del usuario UCI.

Las PC Cliente se comunicarán con el Servidor Central de Procesamiento a través del protocolo HTTPS[8], permitiéndoles seguridad en el envío de los datos. Mientras, que para la comunicación entre los servidores se utilizará el mecanismo ADO[9].

2.5.4. Vista de Implementación

Esta vista describe la descomposición del software en capas y subsistemas de implementación, mostrando la colección de componentes de cada capa. También provee una vista de la trazabilidad de los elementos de diseño de la vista lógica ahora para la implementación.

Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes de Ada, bibliotecas cargadas dinámicamente, etc. Las relaciones de dependencia se utilizan en los diagramas de componentes para indicar que un componente utiliza los servicios ofrecidos por otro componente.

- **ext-all.js, ext-base.js:** Ficheros con las librerías ExtJS
- **ext-all.css:** Fichero con los estilos ExtJS
- **layout.css:** Fichero con los estilos del layout de la aplicación.

[8] *HTTPS: Hypertext Transfer Protocol (en español protocolo de transferencia de hipertexto).*

[9] *ADO: ActiveX Data Objects*

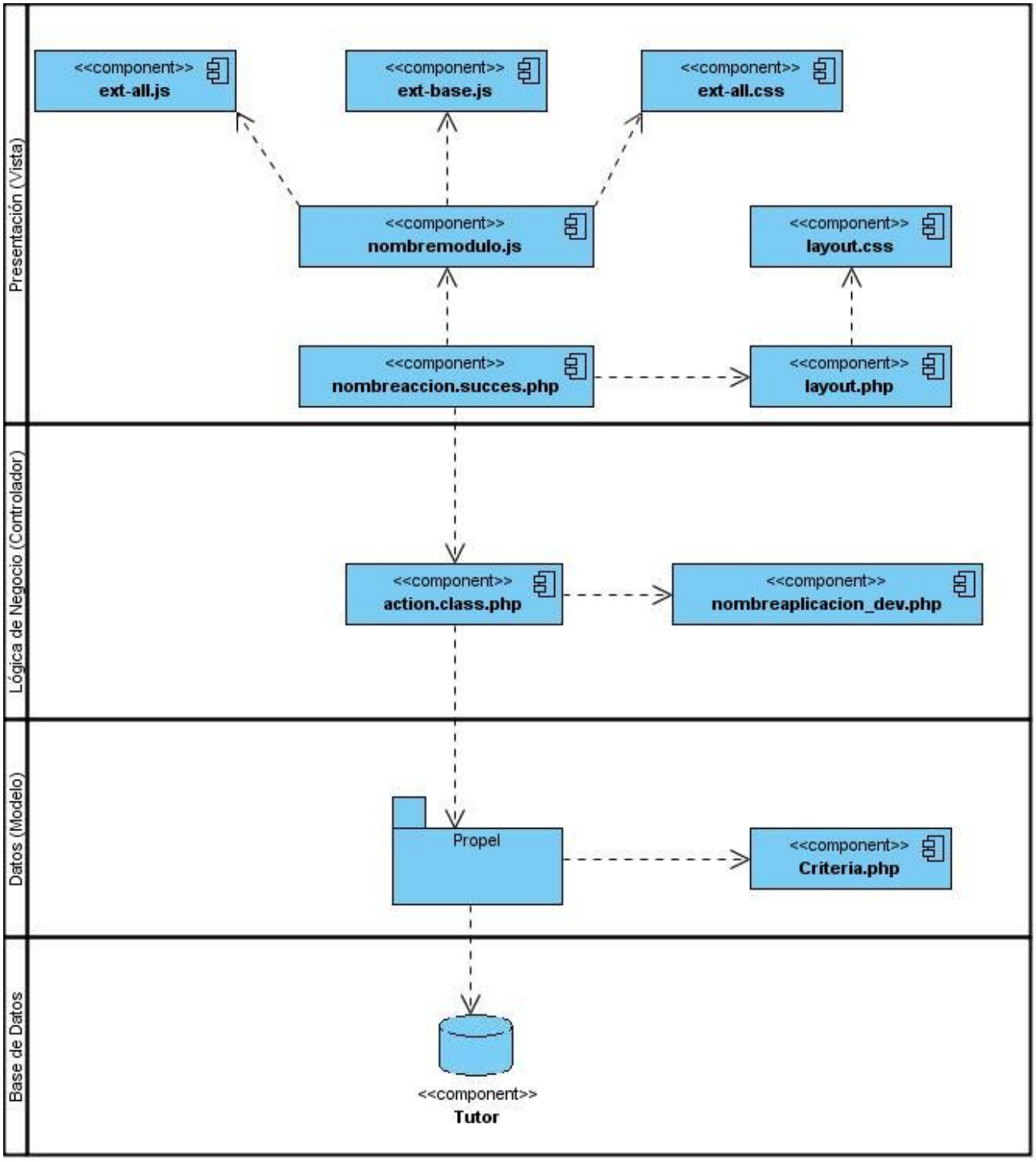

Figura 12. Diagrama de componentes global de la aplicación

2.6. Conclusiones del Capítulo

En este capítulo se realizó la descripción de la arquitectura del sistema haciendo uso del Modelo 4+1 vista que propone la metodología RUP, resumiendo que:

- Se identificaron los estilos y patrones arquitectónicos utilizados en la solución.
- Se establecieron los estándares de codificación, así como los requisitos no funcionales.
- Se fundamentaron los componentes que soportan el estilo Modelo-Vista-Controlador en cuatro de las cinco vistas que propone la metodología utilizada.

Capítulo 3. Evaluación de la Arquitectura

Una vez definida la arquitectura es recomendable realizar valoraciones que permitan tener la medida en la que la solución cumple con los parámetros de calidad, demostrando que la arquitectura anteriormente descrita contribuye con un desarrollo de software viable. La evaluación de la AS permite mitigar los riesgos asociados al desarrollo del software, mejorar la visión de los procesos críticos y validar las decisiones de diseño que se tomaron.

Para conocer si una AS cumplió o no con al menos un atributo de calidad de los que se especificaron en los requisitos no funcionales, se utilizan métodos de evaluación. De este cumplimiento o no con uno o varios atributos de calidad, depende el buen diseño de la AS, lo cual determina el éxito o el fracaso de un sistema de software.

En este capítulo se hace un análisis de los métodos de evaluación para luego definir uno por el cual se realizará la evaluación de la AS. Luego, se procede a la aplicación de este método.

3.1. Evaluación de la Arquitectura

El éxito o fracaso de un software está centrado en la arquitectura que éste posea. Una arquitectura bien diseñada garantiza el cumplimiento del sistema con varios atributos de calidad; como por ejemplo confiabilidad, seguridad.

El **objetivo** de evaluar una arquitectura es saber si ésta puede habilitar los requisitos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los *stakeholders* [32]. Estas evaluaciones sirven para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante, verificar que los requisitos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad. [33]

Características de una Evaluación de Arquitectura [32]

- Es uno de los principales puntos de evaluación dentro del proyecto, ya que errores en ella, pueden traer que el proyecto fracase.

- Puede ser realizada por personas interna o externa al proyecto, aunque lo más interesante es que sea realizada por personas externa (Mentores o Arquitectos del Área).

Evaluar una arquitectura de software sirve para prevenir todos los posibles desastres de un diseño que no cumple con los requisitos de calidad y para saber qué tan adecuada es la arquitectura de software diseñada para el sistema. Una evaluación de una arquitectura no da un sí o un no, si es buena o mala, o una calificación, expresa donde está el riesgo; es decir, fortalezas y debilidades identificadas de la arquitectura. Después de la evaluación de una arquitectura de software, se pueden tomar algunas decisiones como: si se puede seguir el proyecto con las áreas de debilidad dadas en la evaluación, si hay que reforzar la arquitectura de software o si hay que comenzar de nuevo toda la arquitectura.[34]

¿Cuándo evaluar la arquitectura?

Una buena regla para decidir cuándo hay que realizar una evaluación de una arquitectura es realizarla cuando el equipo de desarrollo comience a tomar decisiones que dependan de la arquitectura y el costo de deshacer dichas decisiones sea mayor al costo de realizar la evaluación.[34]

Es posible realizar la evaluación en cualquier momento según Kazman, el cual propone dos variantes que agrupan dos etapas distintas [33]:

1. Evaluación temprana: No necesita que la arquitectura esté completamente especificada. Esta evaluación abarca desde las fases tempranas de diseño y a lo largo del desarrollo, puesto que pueden aparecer cambios arquitectónicos, producto de una evaluación en función de los atributos de calidad esperados.
2. Evaluación tardía: Se realiza cuando la arquitectura se encuentra establecida y la implementación completada. Este es el caso general que se presenta al momento de la adquisición de un sistema ya desarrollado, donde puede observarse el cumplimiento de los atributos de calidad asociados al sistema y como será su comportamiento general.

3.2. Técnicas de Evaluación

Las técnicas de evaluación están clasificadas en cualitativas o cuantitativas [32]:

- **Cualitativas o de cuestionamiento**. Utilizan cualitativas para preguntarle a la arquitectura.
  - Cuestionarios. Abiertas. Temprana.

  - Checklist: Específico del Dominio de la aplicación
  - Escenarios. Específicas del Sistema. Arquitectura avanzada.
- **Cuantitativas**. Sugiere hacerle medidas cuantitativas a la arquitectura
  - Utiliza métricas arquitectónicas, como acoplamiento, coherencia en los módulos, profundidad en herencias, modificabilidad.
  - Simulaciones, Prototipos, y Experimentos

Generalmente, las técnicas de evaluación *cualitativas* son utilizadas cuando la arquitectura está en construcción, mientras que las técnicas de evaluación *cuantitativas*, se usan cuando la arquitectura ya ha sido implantada.[35]

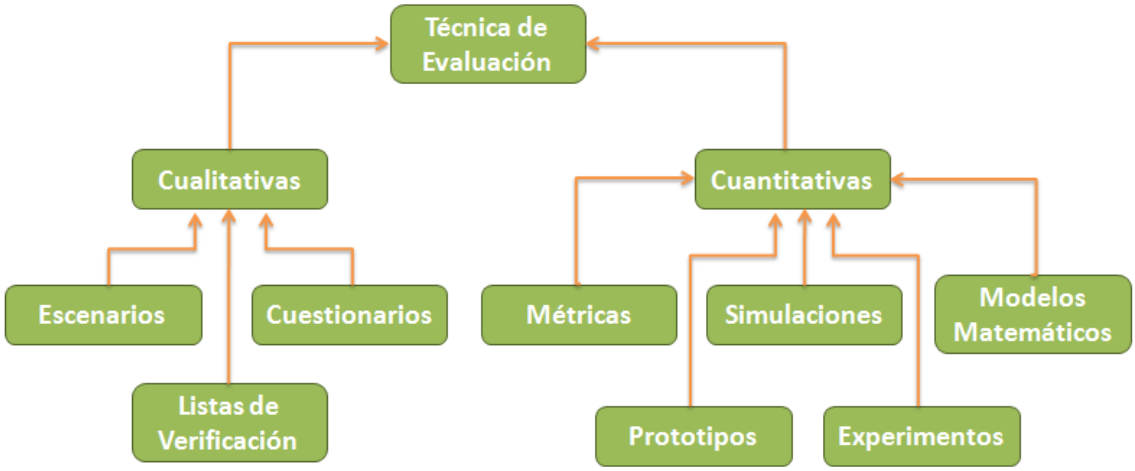

Figura 13. Clasificación de las Técnicas de Evaluación.

3.3. Métodos de Evaluación

Hasta hace poco no existían métodos de utilidad general para evaluar arquitecturas de software. Si alguno existía, sus enfoques eran incompletos y no repetibles, lo que no brindaba mucha confianza. En virtud de esto, múltiples métodos de evaluación han sido propuestos. Un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema para la búsqueda de conflictos que puede presentar una arquitectura y sus soluciones. A continuación se explican algunos de los más importantes.

Software Architecture Analysis Method (SAAM)

El Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) fue el primero en documentarse. Originalmente fue creado para el análisis de la modificabilidad de una arquitectura, pero al llevarse a la práctica demostró su utilidad en la evaluación de distintos atributos de calidad como: modificabilidad, portabilidad, escalabilidad e integrabilidad. Su principal propósito está enfocado en la enumeración de un conjunto de escenarios que representan los posibles cambios a los que estará sometido el sistema en un futuro.[33]

Para la evaluación de una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requisitos de modificabilidad. Mientras que en caso aplicarse el método para varias arquitecturas candidatas, se produce una escala relativa que permite observar qué opción satisface mejor los requisitos de calidad con la menor cantidad de modificaciones.

El método de evaluación SAAM comprende 6 pasos:

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

Architecture Trade-off Analysis Method (ATAM)

El Método de Análisis de Acuerdos de Arquitectura (Architecture Trade-off Analysis Method, ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM. Los estilos arquitectónicos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, integrarse con otros sistemas, entre otros. [33]

ATAM es un método de evaluación de arquitectura basado en cuestionarios y escenarios, que permite evaluar qué tan bien un atributo de calidad es soportado por la arquitectura y (permitiendo reconocer las

decisiones de arquitectura que afectan a más de un atributo de calidad) provee un entendimiento en cómo los atributos de calidad interactúan (tradeoff,). [32]

El método consta de nueve pasos, divididos en cuatro grupos:

1. **Presentación**
   Paso 1: Presentar el ATAM
   Paso 2: Presentar las pautas del negocio
   Paso 3: Presentar la arquitectura
2. **Investigación y Análisis**
   Paso 4: Identificar las propuestas arquitectónicas.
   Paso 5: Generar el árbol de utilidad de los atributos de calidad
   Paso 6: Analizar las propuestas arquitectónicas
3. **Pruebas**
   Paso 7: Lluvia de ideas y priorización de escenarios.
   Paso 8: Analizar las propuestas arquitectónicas.
4. **Informes**
   Paso 9: Presentar los resultados

Active Reviews for Intermediate Designs (ARID)

El Método de Análisis de Diseños Intermedios (Active Reviews for Intermediate Designs, ARID) es un método de bajo costo y gran beneficio, el mismo es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo [33]. ARID es un híbrido entre ADR (Active Design Review) y ATAM: en el caso de ADR, los involucrados reciben documentación detallada y completan cuestionarios, cada uno por separado; y en el caso de ATAM, está orientado a la evaluación de toda una arquitectura [36].

ARID se basa en ensamblar el diseño de los *stakeholders* para articular los escenarios de usos importantes o significativos, y probar el diseño para ver si satisface los escenarios. Como resultado de la aplicación de dicho procedimiento se obtiene un diseño de alta fidelidad acompañado de una alta familiarización con el diseño de los *stakeholders*. [37]

El método de evaluación ARID comprende 9 pasos, agrupados en 2 fases:

- **Actividades Previas**

  Paso 1: Identificar los encargados de la revisión.

  Paso 2: Preparar el informe de diseño.

  Paso 3: Preparar los escenarios base.

  Paso 4: Preparar los materiales
- **Revisión.**

  Paso 5: Presentar el ARID.

  Paso 6: Presentar el diseño.

  Paso 7: Lluvia de ideas y establecimiento de prioridades de escenarios.

  Paso 8: Aplicar los escenarios.

  Paso 9: Resumen

Método de Diseño de Arquitecturas de Software propuesto por Bosch (2000)

El Método de Diseño de Arquitecturas de Software propuesto por Bosch plantea que: “El proceso de evaluación debe ser visto como una actividad iterativa, que forma parte del proceso de diseño, también iterativo. Una vez que la arquitectura es evaluada, pasa a una fase de transformación, asumiendo que no satisface todos los requisitos. Luego, la arquitectura transformada es evaluada de nuevo”.[33]

Metodología del método propuesto por Bosch

- **Etapa 1**

  Paso 1: Selección de atributos de calidad.

  Paso 2: Definición de los perfiles.

  Paso 3: Selección de una técnica de evaluación
- **Etapa 2**

  Paso 4: Ejecución de la evaluación.

  Paso 5: Obtención de resultados.

3.3.1. Selección del método de evaluación

A continuación se muestra una tabla con la comparación entre los métodos de evaluación estudiados.

Métodos	SAAM	ATAM	ARID	Bosch (2000)
Atributos de Calidad	- Modificabilidad - Funcionalidad	- Modificabilidad - Seguridad - Confiabilidad - Desempeño	Conveniencia del diseño evaluado	Seleccionados por el arquitecto, de acuerdo a la importancia sobre el sistema
Objetos Analizados	- Documentación - Vistas Arq.	- Estilos Arq. - Documentación - Flujo de Datos - Vistas Arq.	Especificación de los componentes	- Estilos Arq. - Vistas Arq. - Patrones Arq. - Patrones de Diseño - Patrones de Idioma
Etapas del Proyecto en las que se Aplica	Luego que la arquitectura cuenta con funcionalidad ubicada en módulos	Luego que el diseño de la arquitectura ha sido establecido	A lo largo del diseño de la arquitectura	Luego que el diseño de la arquitectura ha sido establecido
Enfoques Utilizados	- Lluvia de ideas para escenarios y articular los requisitos de calidad. - Análisis de los escenarios para verificar funcionalidad o estimar el costo de los cambios.	- Árbol de Utilidad y lluvia de ideas para articular los requisitos de calidad. - Análisis arquitectónico que detecta puntos sensibles, puntos de balance y riesgos.	Revisiones de diseño, lluvia de ideas para obtener escenarios.	Análisis de perfiles

Tabla 2. Comparación de métodos de evaluación.

La solución de arquitectura que se propone en este trabajo se encuentra especificada a alto nivel, es decir, un diseño poco detallado. Esta especificación a alto nivel impide la selección de escenarios que permitan validar si la arquitectura diseñada satisface a requisitos específicos. Como resultado de esto se

realizará la evaluación atendiendo a cómo responden algunos elementos ante los principales atributos de calidad. Para ello se utilizó el método de evaluación ATAM (Analysis Tradeoff Architecture Method).

3.4. Atributos de calidad en la arquitectura de software

Para determinar los atributos de calidad que debe lograr la AS definida, fue necesario guiarse por los requisitos no funcionales y por los modelos de calidad. Mediante estos atributos será posible saber si la arquitectura seleccionada es correcta evaluando el resultado que han producido dichas decisiones sobre estos.

Entre los modelos de calidad más importantes propuestos se encuentran: McCall (1977), FURPS (1987), Dromey (1996) e ISO/IEC 9126, este último adaptado para arquitecturas de software propuesto por Losavio en el año 2003.

ISO/IEC 9126

Este estándar ha sido desarrollado en un intento de identificar los atributos clave de calidad para un producto de software[38]. El mismo es una simplificación del Modelo de McCall[39], e identifica seis características básicas de calidad que pueden estar presentes en cualquier producto de software, las mismas se relacionan a continuación: Funcionalidad, Confiabilidad, Usabilidad, Eficiencia, Mantenibilidad y Portabilidad.

ISO/IEC 9126 adaptado para arquitecturas de software

Losavio [38] en el año 2003 propone una adaptación del modelo ISO/IEC 9126 del año 2001 de calidad de software para efectos de la evaluación de AS. El modelo se basa en los atributos de calidad que se relacionan directamente con la arquitectura: funcionalidad, confiabilidad, eficiencia, mantenibilidad y portabilidad.

La presente investigación utilizará los atributos de calidad del modelo ISO/EC 9126 adaptado para arquitecturas de software para la evaluación del diseño arquitectónico debido a la correspondencia de dichos atributos con las necesidades del sistema. La tabla que se muestra a continuación presenta estos atributos de calidad planteados por Losavio [38] que poseen sub-características asociadas con elementos de tipo arquitectónicos.

Características	Sub-características	Elementos de tipo arquitectónico
Funcionalidad	Adecuación	Refinamiento de los diagramas de secuencia.
	Exactitud	Identificación de los componentes con las funciones responsables de los cálculos.
	Interoperabilidad	Identificación de conectores de comunicación con sistemas externos.
	Seguridad	Mecanismos o dispositivos que realizan explícitamente la tarea.
Confiabilidad	Tolerancia a fallas	Existencia de mecanismos o dispositivos de software para manejar excepciones.
	Recuperabilidad	Existencia de mecanismos o dispositivos de software para restablecer el nivel de desempeño y recuperar datos.
Eficiencia	Desempeño	Componentes involucrados en un flujo de ejecución para una funcionalidad.
	Utilización de recursos	Relación de los componentes en términos de espacio y tiempo.
Mantenibilidad	Acoplamiento	Interacciones entre componentes.
	Modularidad	Número de componentes que dependen de un componente.
Portabilidad	Adaptabilidad	Presencia de mecanismos de adaptación.
	Instalabilidad	Presencia de mecanismos de instalación.
	Coexistencia	Presencia de mecanismos que faciliten la coexistencia.
	Reemplazabilidad	Lista de componentes reemplazables para cada componente.

Tabla 3. Atributos de calidad - Modelo ISO/IEC 9126 adaptado para arquitecturas de software.

3.5. Aplicación del método seleccionado

Como anteriormente se expuso, el método de evaluación para la arquitectura será ATAM, utilizando los atributos de calidad ISO/IEC 9126 adaptado para arquitecturas de software y además se apoyará en las técnicas estudiadas con el instrumento de evaluación Utlily Tree.

A continuación se presenta la evaluación de la arquitectura propuesta según las fases del método de evaluación ATAM.

Fase 1: Presentación

Paso 1: Presentar el ATAM

Se presenta el método a los *stakeholders* explicando el proceso a seguir y el involucramiento y responsabilidad de cada uno en el proyecto. Se detallan los pasos a seguir, las técnicas a utilizar y los resultados a obtener.

Paso 2: Presentar las pautas del negocio

En este paso, el líder del proyecto presenta el sistema desde la perspectiva del negocio, al equipo de ATAM y a los *stakeholders*, donde quedan expuestos como principales funciones del sistema, los CU más significativos mostrados en el capítulo anterior. Por otra parte, se obtuvieron los requisitos de los atributos de calidad del sistema, los cuales guiarán el resto de la evaluación.

Paso 3: Presentar la arquitectura

El arquitecto presenta la Arquitectura de Software definida incluyendo por lo menos los estilos utilizados, otros sistemas con que se debe interactuar, restricciones técnicas de software como por ejemplo uso de sistema operativo. En el capítulo anterior se describe detalladamente la arquitectura a evaluar, razón por lo cual no es necesario su presentación en este paso.

Fase 2: Investigación y Análisis

Paso 4: Identificar las propuestas arquitectónicas.

Se le solicita al arquitecto las propuestas arquitectónicas o estilos de arquitectura utilizados, ya que éstos El árbol de utilidad generado se toma como un plan para el resto de la evaluación que realiza el método. Indica además al equipo evaluador dónde ocupar su tiempo y dónde probar aproximaciones y riesgos arquitectónicos.

Paso 5: Generar el árbol de utilidad de los atributos de calidad

Se genera el árbol de utilidad donde principalmente el arquitecto, identifica, prioriza y refina los requisitos de atributos de calidad más importantes del sistema, según se mencionó previamente, identificando los escenarios en el árbol. Cada uno de estos escenarios se ubicó en una hoja del árbol de utilidad, según el atributo de calidad que estuviera poniendo a prueba.

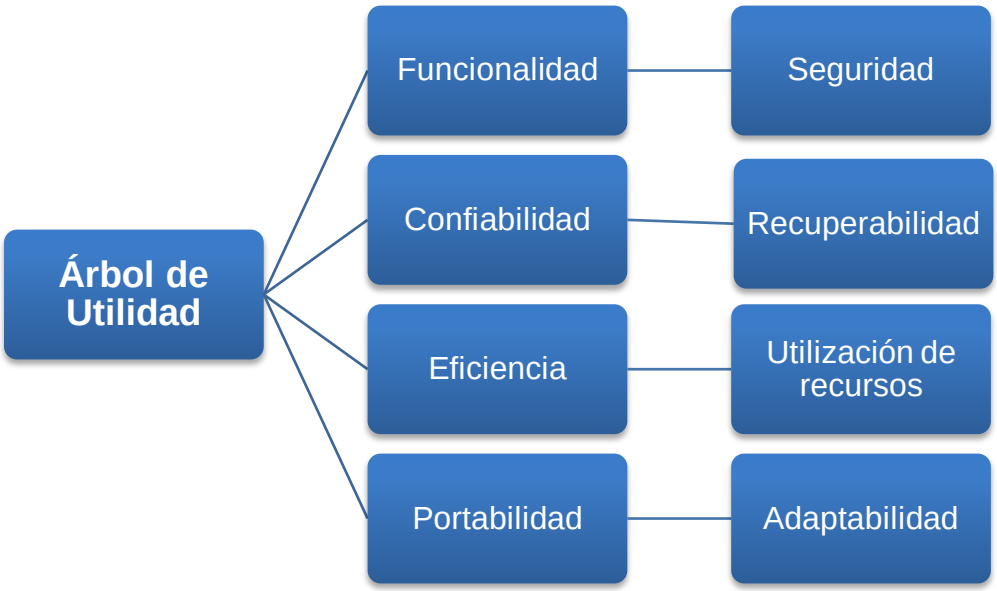

Figura 14. Árbol de Utilidad

Al construir el árbol de utilidad, se define la prioridad del escenario como un par (X, Y) donde; teniendo en cuenta la importancia de cada escenario para el éxito del sistema dada por el cliente se asignó la priorización en la primera dimensión, y según el grado de dificultad estimado por el grupo de evaluación para realizarlo, en la segunda. Los posibles valores para X e Y son A (Alto), B (Bajo) y M (Medio).

Funcionalidad	Seguridad	**(A, A)** El acceso a la información del sistema se les brindará a las personas que tienen los permisos para ello, donde especificarán en su autenticación el usuario y la contraseña.
		(A, M) La comunicación entre el cliente y el servidor se realizará a través de protocolos que le proporcionen una conexión segura.

		(A, B) La información almacenada en la Base de Datos utilizará métodos de encriptación con el objetivo de brindar mayor seguridad.
Confiabilidad	Recuperabilidad	**(A, M)** El servidor central de base de datos tendrá un servidor de respaldo que servirá de salva a la base de datos original.
		(M, A) Error del sistema operativo que reinicia el ordenador de la PC del cliente.
Eficiencia	Utilización de recursos	**(A, M)** Los tiempos de respuesta a las peticiones realizadas por los usuarios deberán estar entre 0.1 y 0.7.
Portabilidad	Adaptabilidad	**(A, B)** El sistema podrá ser usado tanto en Sistemas Operativos Windows como Linux.

Tabla 4. Prioridad de los escenarios.

A continuación se documentan detalladamente los escenarios que fueron ubicados en las hojas del árbol de utilidad y las propuestas arquitectónicas que cumplen con estos escenarios, con el objetivo de determinar cuán adecuados es el uno para el otro. Esto posibilita que la propuesta instanciada en la arquitectura que se está evaluando es la apropiada para satisfacer los requerimientos de un atributó en específico.

Escenario #1	Escenario: El acceso a la información del sistema se les brindará a las personas que tienen los permisos para ello, donde especificarán en su autenticación el usuario y la contraseña.			
Atributo(s)	Funcionalidad – Seguridad			
Entorno	Operación normal			
Estímulo	Un usuario que con autenticación incorrecta (ya sea el usuario o la contraseña) intenta acceder a las funcionalidades del sistema.			
Respuesta	Sólo se muestran las funcionalidades a los usuarios que accedan con un usuario y contraseña correcta, según el rol correspondiente a estos.			
Decisiones Arquitectónicas	**Punto de sensibilidad**	**Punto de trade-off**	**Riesgo**	**No Riesgo**
Archivo de configuración security.yml	S1			N1

Razonamiento	Para restringir el acceso a una acción se crea y se edita un archivo de configuración YAML llamado security.yml en el directorio config/ del módulo. En este archivo, se pueden especificar los requerimientos de seguridad que los usuarios deberán satisfacer para cada acción o para todas (all) las acciones. En Symfony, los 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 que permiten organizar la seguridad en grupos.

Tabla 5. Escenario #1.

Escenario #2	Escenario: La comunicación entre el cliente y el servidor se realizará a través de protocolos que le proporcionen una conexión segura.			
Atributo(s)	Funcionalidad – Seguridad			
Entorno	Operación normal			
Estímulo	Envío de información desde el cliente al servidor de aplicaciones.			
Respuesta	Se encripta la información y es enviada de forma segura al servidor de aplicaciones.			
Decisiones Arquitectónicas	**Punto de sensibilidad**	**Punto de trade-off**	**Riesgo**	**No Riesgo**
Protocolo Seguro HTTPS	S2			N2
Razonamiento	El uso del protocolo HTTPS proporciona una protección razonable a la información enviada a través de la red garantizándose la seguridad de la misma.			

Tabla 6. Escenario #2.

Escenario #3	Escenario: La información almacenada en la base de datos utilizará métodos de encriptación con el objetivo de brindar mayor seguridad.			
Atributo(s)	Funcionalidad – Seguridad			
Entorno	Operación normal			
Estímulo	Información almacenada en la base de datos			
Respuesta	Se encripta la información proporcionándole seguridad a los datos.			
Decisiones Arquitectónicas	**Punto de sensibilidad**	**Punto de trade-off**	**Riesgo**	**No Riesgo**
Uso del método md5	S3			N3

Razonamiento	El MD5 es un algoritmo de reducción criptográfico que permite encriptar la información que se ha de almacenar en la base de datos.

Tabla 7. Escenario #3.

Escenario #4	Escenario: El servidor central de base de datos tendrá un servidor de respaldo que servirá de salva a la base de datos original.			
Atributo(s)	Confiabilidad – Recuperabilidad			
Entorno	Operación normal			
Estímulo	Ante fallas en el servidor central de base de datos, salvar los datos del mismo.			
Respuesta	En caso de ocurrir una falla se recuperan los datos almacenados en una base de datos de respaldo.			
Decisiones Arquitectónicas	**Punto de sensibilidad**	**Punto de trade-off**	**Riesgo**	**No Riesgo**
Se implementará un sistema de réplicas entre la base datos, representándose en el despliegue.				N4
Razonamiento	Es de vital importancia la protección física de los datos en la seguridad del sistema. Con el uso de un servidor de respaldo para la base de datos se garantiza que en caso de que ocurra una falla estos puedan ser recuperados.			

Tabla 8. Escenario #4.

Escenario #5	Escenario: En caso de fallo, recuperar el sistema y los datos entrados sin enviar.			
Atributo(s)	Confiabilidad – Recuperabilidad			
Entorno	Operación normal			
Estímulo	Error del sistema operativo que reinicia el ordenador de la PC del cliente.			
Respuesta	No hay recuperación de los datos sin enviar.			
Decisiones Arquitectónicas	**Punto de sensibilidad**	**Punto de trade-off**	**Riesgo**	**No Riesgo**
No hay			R1	
Razonamiento	Como no se pueden recuperar los datos sin enviar una vez ocurrido un fallo externo, ya sea a causa del fluido eléctrico, un error del sistema operativo o cualquier fallo ajeno a la aplicación en la PC del cliente, se produce el mayor de los errores pues no existe ninguna			

	decisión arquitectónica capaz de solucionar esto.

Tabla 9. Escenario #5.

Escenario #6	Escenario: Los tiempos de respuesta a las peticiones realizadas por los usuarios deberán estar entre 0.1 y 0.7 segundos.			
Atributo(s)	Eficiencia – Utilización de recursos.			
Entorno	Conexión concurrente de múltiples usuarios.			
Estímulo	El usuario necesita realiza una acción en el sistema.			
Respuesta	Muestra los datos en un tiempo respuesta entre 0.1 y 0.7 segundos.			
Decisiones Arquitectónicas	**Punto de sensibilidad**	**Punto de trade-off**	**Riesgo**	**No Riesgo**
SGBD PostgreSQL	S4			N5
Implementar procedimientos almacenados				N6
Razonamiento	Ante una situación donde existen múltiples usuarios realizando peticiones concurrentemente en el sistema, las decisiones arquitectónicas que garantizan la respuesta del mismo en el tiempo que se desea son: • SGBD PostgreSQL en el manejo de las conexiones concurrentes de muchos usuarios de forma eficiente. • Implementar procedimientos almacenados permitiendo que las respuestas de las consultas a la base de datos sean más rápidas.			

Tabla 10. Escenario #6.

Escenario #7	Escenario: El sistema podrá ser usado tanto en Sistemas Operativos Windows como Linux.			
Atributo(s)	Portabilidad – Adaptabilidad.			
Entorno	Operación normal			
Estímulo	El usuario desea utilizar el sistema en un Sistema Operativo Windows o Linux.			
Respuesta	La aplicación se encuentra disponible a través de la web.			
Decisiones Arquitectónicas	**Punto de sensibilidad**	**Punto de trade-off**	**Riesgo**	**No Riesgo**
Tecnología Web	S5			N6

Razonamiento	Se utiliza la tecnología Web que está basada en la necesidad de garantizar el acceso al sistema desde cualquier sistema operativo, solo se necesita estar conectado a Internet y tener algún navegador Web instalado.

Tabla 11. Escenario #7.

Paso 6: Analizar las propuestas arquitectónicas

Se analizan las propuestas arquitectónicas según el árbol de utilidad generado. En este paso se evalúa la propuesta arquitectónica que influye en la obtención o no del atributo de calidad requerido, y se identifican los riesgos, no riesgos, puntos de sensibilidad y concesión asociados a dicha evaluación.

Fase 3: Pruebas

Paso 7: Lluvia de ideas y priorización de escenarios

Se confirman e identifican nuevos escenarios según varios *stakeholders* involucrados, los que también se priorizan y se comparan con los identificados en el árbol de utilidad. Pueden suceder tres casos: el escenario ya está en el árbol de utilidad, no está pero encaja en alguna rama y se convierte en una nueva hoja, no está y no encaja en ninguna rama, lo que significa que no había sido considerado previamente.

Paso 8: Analizar las propuestas arquitectónicas.

Se realiza lo mismo que en el sexto paso haciendo uso del paso anterior, para el nuevo árbol de utilidad con todos los escenarios incluidos.

Paso 9: Presentar los resultados

Se presentan los resultados a los *stakeholders* y entrega la documentación correspondiente a las salidas del ATAM. Finalmente, haciendo uso del método de evaluación seleccionado ATAM se resume y presenta; las más importantes salidas son:

Puntos de sensibilidad

S1. El desarrollo del archivo security.yml es un punto de sensibilidad para garantizar el acceso a la información solamente a aquellos roles que sean especificado en su interior.

S2. El uso del protocolo seguro HTTPS constituye un punto de sensibilidad al encriptar todo el tráfico de información.

S3. El uso del método md5 al encriptar la información a almacenar constituye un punto de sensibilidad.

S4. El normalizar la base de datos crea un punto de sensibilidad, para lograr que los tiempos de respuesta a las peticiones realizadas por los usuarios sean menores que 0.7 segundos.

S5. La decisión arquitectónica de utilizar tecnología Web es un punto de sensibilidad para poder acceder al sistema desde cualquier Sistema Operativo.

Puntos de Trade-off

No se detectaron puntos de trade-off ya que no existen decisiones que intervengan en la respuesta del sistema ante diferentes atributos.

Riesgos

R1. Al no existir decisiones arquitectónicas para contrarrestar cualquier fallo en el sistema puede traer consigo graves problemas, ya que los datos no podrán ser recuperados sin haber sido enviados previamente a la base de datos.

No Riesgos

N1. Cuando no existe el archivo security.yml o no se indica ninguna acción en ese archivo, todas las acciones son accesibles por todos los usuarios. Si existe un archivo security.yml, Syfmony busca por el nombre de la acción y si existe, verifica que se satisfagan los requerimientos de seguridad. Lo que sucede cuando un usuario trata de acceder una acción restringida depende de sus credenciales.

N2. El uso del protocolo HTTPS para la transferencia de la información permite que se cree un canal cifrado, permitiendo que la información sensible no sea interceptada por un atacante que sólo obtendrá será un flujo de datos cifrados.

N3. La codificación del MD5 de 128 bits es representada típicamente como un número de 32 dígitos hexadecimal, brindándole seguridad a la información almacenada.

N4. Se debe implementar en el despliegue un sistema de réplicas del servidor central base de datos, logrando que se salve de forma íntegra los datos en un servidor de respaldo ante cualquier falla.

N5. El uso del SGBD PostgreSQL permite que los usuarios accedan rápidamente a la información.

N6. Al implementar procedimientos almacenados se logra que las respuestas de las consultas de las base de datos sean más rápidas ante la concurrencia de peticiones por parte de los usuarios.

N7. Utilizar la tecnología Web garantiza que la aplicación esté disponible para cualquier Sistema Operativo.

3.6. Conclusiones del Capítulo

Al terminar este capítulo se puede concluir que:

- Se realizó una investigación de los métodos de evaluación para aplicar uno de estos.
- Se aplicó el método de evaluación de arquitecturas ATAM que está basado en la técnica de escenarios utilizando el Árbol de Utilidad.
- Se analizaron escenarios, identificándose riesgos y puntos de sensibilidad, lo cual contribuyó a refinar y perfeccionar las decisiones arquitectónicas.

Conclusiones Generales

Durante el transcurso de esta investigación y a través de su desarrollo:

- Se definió la línea base de la arquitectura de VIRTEVALL, para la cual se realizó un estudio de estilos y patrones que le brindarán al sistema la calidad, sencillez y claridad requerida tanto en la estructura como en el diseño de la solución.
- Se generaron como artefactos el Documento de Arquitectura de Software y 4 de las vistas propuestas por la metodología RUP (Vista de Casos de Uso, Vista Lógica, Vista de Implementación, Vista de Despliegue) en la herramienta CASE seleccionada.
- Se evaluó la arquitectura definida mediante el método ATAM, contribuyendo al diseño de una arquitectura flexible, robusta y escalable.

Recomendaciones

Al finalizar este trabajo se tiene el propósito de ampliar y mejorar la documentación de la arquitectura presentada, para lograrlo se recomienda:

- Utilizar la arquitectura de VIRTEVALL en futuras aplicaciones de líneas de trabajo similares.
- Aplicar otros métodos de evaluación de la arquitectura para aumentar la fortaleza del sistema.

Referencias Bibliográficas

1. **Wright, E.B. y Forcier, R.C.** *The Computer: A Tool for the Teacher.* Wadsworth, Estados Unidos : s.n., 1985.

2. **Wenger, E.** Artificial intelligence and tutoring systems. Computational and Cognitive Approaches to the Communication of Knowledge. 1987.

3. **Wolf, B.** Context Dependent Planning in a Machine Tutor. 1984. University of Massachusetts, Amherst, Massachusetts..

4. **Cataldi, Zulma and Lage, Fernando J.** *Modelo de Sistemas Tutor Inteligente distribuido para educación a distancia.* Facultad Regional Buenos Aires. Universidad Tecnológica Nacional.

5. **Carbonell, J. R.** *AI in CAI: An artificial intelligence approach to computer assisted.* 1970. pp. 190-202. Vol. 11, IEEE transaction on Man Machine System.

6. **Casares Charles, Juan Pablo.** AMIVA: Ambiente para la Instrucción Visual de Algoritmos. [Tesis]. Mexico D.F : s.n., Julio 1999. p. 29.

7. **Burns, H. L. y Capps, C.G.** *Foundations of STI: An Introduction.* Estados Unidos : s.n., 1988.

8. **Clancey, W. J.** *Intelligent tutoring systems: A tutorial survey.* 1991.

9. **Salgueiro, Fernando A.** Sistemas inteligentes para el Modelado del Tutor. [Tesis]. Noviembre, 2005. Universidad de Buenos Aires. Argentina.

10. **Jacobson, Ivar and Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* Madrid, España : Addison Wesley, 2000. 84-7829-036-2.

11. **Kruchten, Philippe.** *El Modelo de "4+1" Vistas de la Arquitectura del Software.* s.l. : IEEE Software, Noviembre, 1995.

12. **Perry, Dewayne E. y Wolf, Alexander L.** *Foundations for the study of software architecture.* 1992. pp. 40-52.

13. **Clements, Paul y Shaw, Mary.** *A field guide to Boxology : Preliminary classification of architectural styles for software systems.* 1997.

14. **Burbeck, Steve.** *Application programming in Smalltalk-80: How to use Model-View-Controller (MVC).* 1992. University of Illinois in Urbana-Champaign, Smalltalk Archive, http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html..

15. **Garlan, David y Shaw, Mary.** *An introduction to software architecture.* 1994. CMU Software Engineering Institute.

16. **Alexander, Christopher, et al.** *A Pattern Language: Towns/Building/Construction.* 1977. Oxford University Press.

17. **Buschmann, F., et al.** *Pattern - Oriented Software Architecture. A System of Patterns.* Inglaterra : John Wiley & Sons, 1996.

18. **Sarver, Toby.** *Pattern Refactoring Workshop.* 2000.

19. **Carmona Ruiz, Alvaro Ernesto.** De los patrones de análisis y de integración a los componentes de negocio. [ppt]. Bogotá, Colombia : s.n., 2005. Software Architect Heinsohn Software House S.A..

20. **Gamma, E., et al.** *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley : s.n., 1995.

21. **Brito Acuña, Karenny.** Selección de Metodologías de Desarrollo para Aplicaciones Web en la Facultad de Informática de la Universidad de Cienfuegos. [Tesis]. Cienfuegos, Cuba : s.n., 2009. Texto completo en www.eumed.net/libros/2009c/584/.

22. **Palacio, Juan.** *Flexibilidad con Scrum.* 2007. Versión impresa, disponible en http://www.lulu.com.

23. **Mendoza Sanchez, María A.** *Metodologías De Desarrollo De Software.* 2004.

24. **Letelier, Patricio y Penadés, Mª Carmen.** *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* Universidad Politécnica de Valencia.

25. **Kruchten, Philippe.** *The Rational Unified Process: An Introduction.* s.l. : Addison Wesley, Marzo 2000. 0-201-70710-1.

26. **JavaTech.** An Introduction to Scientific and Technical Computing with Java. JavaTech. [Online] 2004. http://www.particle.kth.se/~lindsey/JavaCourse/Book/courseMap.html..

27. **Universidad de las Ciencias Informáticas.** Comunidad_PHP. *Portal Comunidad de PHP.* [Online] http://php.uci.cu.

28. **Magaña, Carlos Leopoldo.** Zend Framework, una introducción. [Online] 1 10, 2008. http://www.carlosleopoldo.com/post/zend-framework-una-introduccion/..

29. **Potencier, Fabien and Zaninotto, François.** *Symfony 1.2, la guía definitiva.* 2009. Se puede encontrar en http://www.librosweb.es/symfony_1_2.

30. **Quiñones, Ernesto.** Introducción a PostgreSQL. [Online] http://www.postgresql.org.pe/articles/introduccion_a_postgresql.pdf.

31. IBM Rational Software. [Online] IBM Corporation . http://www.rational.com.

32. **Brey, Gustavo Andrés, et al.** *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas.* Buenos Aires : s.n., 2005. Universidad Tecnológica Nacional. Facultad Regional de Buenos Aires - Departamento de Sistemas.

33. **Camacho, Erika, Cardeso, Fabio and Nuñez, Gabriel.** *Arquitecturas de Software. Guía de Estudio.* 2004.

34. **Toledo, Icedo, Vargas, Rosa Virginia y Trejo and Moisés, Jorge.** *Software Architecture Assesment.* 2003. CITMA.

35. **Gómez, Omar Salvador.** *Evaluando Arquitecturas de Software. Parte 1. Panorama General.* 2007. 1870-0888.

36. **Clement, Paul.** *SEI (Software Engeniering Institute).* 2000. Z39-18298-102.

37. **Gómez, Omar Salvador.** *Evaluando Arquitecturas de Software. Parte 2. Panorama General.* México : Brainworx S.A : s.n., 2007. 1870-0888.

38. **Losavio, Francisca, Chirinos, Ledis and Lévy, Nicole y Ramdane-Cherif, Amar.** *Quality Characteristics for Software Architecture. Quality Characteristics for Software Architecture.* 2003. http://www.jot.fm/issues/issue_2003_03/article2.

39. **Kazman, Rick and Clements, Paul y Klein, Mark.** *Evaluating Software Architectures. Methods and case studies.* s.l. : Addison Wesley, 2001.

Bibliografía

1. **Alexander, Christopher, y otros.** *A Pattern Language: Towns/Building/Construction.* 1977. Oxford University Press.
2. **Bass, Len y Clements, Paul y Kazman, Rick.** Software Architecture in Practice, Second Edition. s.l. : Addison Wesley, Abril 2003. 0-321-15495-9.
3. **Brey, Gustavo Andrés, y otros.** *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas.* Buenos Aires : s.n., 2005. Universidad Tecnológica Nacional. Facultad Regional de Buenos Aires - Departamento de Sistemas.
4. **Brito Acuña, Karenny.** Selección de Metodologías de Desarrollo para Aplicaciones Web en la Facultad de Informática de la Universidad de Cienfuegos. [Tesis]. Cienfuegos, Cuba : s.n., 2009. Texto completo en www.eumed.net/libros/2009c/584/.
5. **Burbeck, Steve.** *Application programming in Smalltalk-80: How to use Model-View-Controller (MVC).* 1992. University of Illinois in Urbana-Champaign, Smalltalk Archive, http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.
6. **Burns, H. L. y Capps, C.G.** *Foundations of STI: An Introduction.* Estados Unidos : s.n., 1988.
7. **Buschmann, F., y otros.** *Pattern - Oriented Software Architecture. A System of Patterns.* Inglaterra : John Wiley & Sons, 1996.
8. **Camacho, Erika, Cardeso, Fabio y Nuñez, Gabriel.** *Arquitecturas de Software. Guía de Estudio.* 2004
9. **Carbonell, J. R.** *AI in CAI: An artificial intelligence approach to computer assisted.* 1970. págs. 190-202. Vol. 11, IEEE transaction on Man Machine System.
10. **Carmona Ruiz, Alvaro Ernesto.** De los patrones de análisis y de integración a los componentes de negocio. [ppt]. Bogotá, Colombia : s.n., 2005. Software Architect Heinsohn Software House S.A
11. **Carrascoso Puebla, Yoan Arlet y Chaviano Gómez, Enrique y Céspedes Vega, Anisleydi.** Procedimiento para la Evaluación de Arquitecturas de Software basadas en Componentes. *Artículo para la publicación de investigaciones concluidas.* Universidad de las Ciencias Informáticas

12. **Casares Charles, Juan Pablo.** AMIVA: Ambiente para la Instrucción Visual de Algoritmos. [Tesis]. Mexico D.F : s.n., Julio 1999. pág. 29.
13. **Cataldi, Zulma y Lage, Fernando J.** *Modelo de Sistemas Tutor Inteligente distribuido para educación a distancia.* Facultad Regional Buenos Aires. Universidad Tecnológica Nacional.
14. **Clancey, W. J.** *Intelligent tutoring systems: A tutorial survey.* 1991.
15. **Clements, Paul.** *SEI (Software Engeniering Institute).* 2000. Z39-18298-102
16. **Clements, Paul y Kazman, Rick y Klein, Mark.** *Evaluating Software Architectures: Methods and Case Studies.* Marzo 2004. 0-201-70482-X.
17. **Clements, Paul y Shaw, Mary.** *A field guide to Boxology : Preliminary classification of architectural styles for software systems.* 1997
18. **Dávila, Mauricio, Germán, Martín y Crutas, Diego y García, Andrés.** *Evaluación de Arquitecturas de Software*
19. **Delgado, Andrea y Castro, Alberto y Germán, Martín.** *Evaluación de Arquitecturas de Software con ATAM (Architecture Tradeoff Analysis Method): un caso de estudio.* febrero 2007. Universidad de la República, Instituto de Computación, Grupo de Ingeniería de Software, Montevideo, Uruguay. 0797-6410
20. Especificación de Atributos de Calidad y QAW. *ppt.* Buenos Aires, Argentina : s.n., Marzo 2009. Ingenieria de Software 2. Primer Cuatrimestre de 2009.
21. **Fragoso Vázquez, Amado Victor y Alférez Salinas, Germán Harvey.** *Un Enfoque para la Aplicación de las Tácticas de la Arquitectura con el Fin de Ampliar la Documentación de sus Calidades.* México.
22. **Garlan, David y Shaw, Mary.** *An introduction to software architecture.* 1994. CMU Software Engineering Institute.
23. **Gamma, E., y otros.** *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley : s.n., 1995
24. **Gómez, Omar Salvador.** *Evaluando Arquitecturas de Software. Parte 1. Panorama General.* 2007. 1870-0888.
25. **Gómez, Omar Salvador.** *Evaluando Arquitecturas de Software. Parte 2. Panorama General.* México : Brainworx S.A : s.n., 2007. 1870-0888
26. IBM Rational Software. [En línea] IBM Corporation . http://www.rational.com

27. **Jacobson, Ivar y Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* Madrid, España : Addison Wesley, 2000. 84-7829-036-2.
28. **JavaTech.** An Introduction to Scientific and Technical Computing with Java. JavaTech. [En línea] 2004. http://www.particle.kth.se/~lindsey/JavaCourse/Book/courseMap.html.
29. **Kazman, Rick y Clements, Paul y Klein, Mark.** *Evaluating Software Architectures. Methods and case studies.* s.l. : Addison Wesley, 2001
30. **Kruchten, Philippe.** *El Modelo de "4+1" Vistas de la Arquitectura del Software.* s.l. : IEEE Software, Noviembre, 1995.
31. **Kruchten, Philippe.** *The Rational Unified Process: An Introduction.* s.l. : Addison Wesley, Marzo 2000. 0-201-70710-1.
32. **Letelier, Patricio y Penadés, Mª Carmen.** *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* Universidad Politécnica de Valencia.
33. **Losavio, Francisca, Chirinos, Ledis y Lévy, Nicole y Ramdane-Cherif, Amar.** *Quality Characteristics for Software Architecture. Quality Characteristics for Software Architecture.* 2003. http://www.jot.fm/issues/issue_2003_03/article2
34. **Magaña, Carlos Leopoldo.** Zend Framework, una introducción. [En línea] 10 de 1 de 2008. http://www.carlosleopoldo.com/post/zend-framework-una-introduccion/.
35. **Mazzini, Daniel.** Patrones de Diseño. *ppt.* Ubica Solutions
36. **Mendoza Sanchez, María A.** *Metodologías De Desarrollo De Software.* 2004
37. **Montaldo, Diego Fernando.** Patrones de Diseño de Arquitecturas de Software Enterprise. *Tesis.* Noviembre 2005. Departamento de Computación. Facultad de Ingeniería. Universidad de Buenos Aires
38. **Moreno, Ana M. y Sánchez-Segura, Maribel.** *Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el momento de Arquitectónico.* España : s.n
39. **Palacio, Juan.** *Flexibilidad con Scrum.* 2007. Versión impresa, disponible en http://www.lulu.com
40. *Patrones del "Gang of Four".* Unidad Docente de Ingeniería del Software. Facultad de informática - Universidad Politécnica de Madrid.
41. **Peralta, Mario.** *Estimación del esfuerzo basada en casos de uso.* Buenos Aires, Argentina : s.n. Centro de Ingeniería del Software e Ingeniería del Conocimiento (CAPIS). Escuela de Postgrado. Instituto Tecnológico de Buenos Aires. http://www.itba.edu.ar/capis/webcapis/planma.html.

42. **Perry, Dewayne E. y Wolf, Alexander L.** *Foundations for the study of software architecture.* 1992. págs. 40-52.
43. **Potencier, Fabien.** *El Tutorial JoBeet.* 2009. http://www.librosweb.es/jobeet_1_3
44. **Potencier, Fabien y Zaninotto, François.** *Symfony 1.2, la guía definitiva.* 2009. Se puede encontrar en http://www.librosweb.es/symfony_1_2.
45. **Quiñones, Ernesto.** Introducción a PostgreSQL. [En línea] http://www.postgresql.org.pe/articles/introduccion_a_postgresql.pdf
46. **Reynoso, Carlos Billy.** *Architect Academy:Seminario de Arquitectura de Software*
47. **Reynoso, Carlos Billy.** *Introducción a la Arquitectura de Software.* Marzo 2004. Universidad de Buenos Aires. Versión 1.0.
48. **Reynoso, Carlos y Kiccillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Marzo 2004. Universidad de Buenos Aires. Versión 1.0
49. **Salgueiro, Fernando A.** Sistemas inteligentes para el Modelado del Tutor. [Tesis]. Noviembre, 2005. Universidad de Buenos Aires. Argentina.
50. **Sarver, Toby***. Pattern Refactoring Workshop***.** 2000**.**
51. **Silva, Andrés.** Ingeniería de Requisitos. *ppt.*
52. **Toledo, Icedo, Vargas, Rosa Virginia y Trejo y Moisés, Jorge.** *Software Architecture Assesment.* 2003. CITMA.
53. **Universidad de las Ciencias Informáticas.** Comunidad_PHP. *Portal Comunidad de PHP.* [En línea] http://php.uci.cu
54. **Wenger, E.** Artificial intelligence and tutoring systems. Computational and Cognitive Approaches to the Communication of Knowledge. 1987.
55. **Wolf, B.** Context Dependent Planning in a Machine Tutor. 1984. University of Massachusetts, Amherst, Massachusetts.
56. **Wright, E.B. y Forcier, R.C.** *The Computer: A Tool for the Teacher.* Wadsworth, Estados Unidos : s.n., 1985.