Universidad de las Ciencias Informáticas

Facultad 9

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN INFORMÁTICA

Título: **Desarrollo de una aplicación para la automatización de la evaluación de la Arquitectura de Software en los Proyectos Productivos de la UCI.**

Autores: **Damaris Batista González**
Isael Bozán Acosta

Tutor: **Ing. Julio Alberto Leyva Durán**

Co-Tutor: **Ing. Liester Cruz Castro**

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

Declaración de Autoría

Declaración de Autoría

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

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

Damaris Batista González

_____

(Firma del Autor)

Ing. Julio Alberto Leyva Durán

_____

(Firma del Tutor)

Isael Bozán Acosta

_____

(Firma del Autor)

Dedicatoria

A mis padres, que han depositado en mí su confianza y cariño.
A mis hermanos.
A Fidel Castro, por su fe ciega en la juventud cubana.
Damaris

A mi madre por siempre confiar en mí, y darme apoyo en todo, por sufrir conmigo cada momento desagradable, por siempre esperarme y despedirme con aquellos platos de comida que cualquiera desearía, hechos con todo el amor del mundo, por todo el amor brindado en estos 23 años y sobre todo, por darme la vida y hacer de mi lo que hoy soy, un ingeniero, por ser aquella persona que si alguna vez me faltara, no sabría cómo vivir.

A mi padre que junto a mi madre son las personas que más quiero en este mundo, por brindarme todo el amor que un padre le puede brindar a su hijo, por pasar tantas malas noches conmigo cuando cogía aquellas crisis de asma que terminaba en una bronconeumonía y pasaba días en el Hospital. Por confiar siempre en mí.

A mis Hermanas que son mi vida también, a Tay por ser como es: tan especial, alegre, jodedora, por siempre estar alegrándome la vida cuando estoy por casa, y a Chave, por ser siempre el modelo profesionalmente a seguir y estar tan unidos a nosotros a pesar de encontrarse algo alejada.

Isael

RESUMEN

La arquitectura de software es fundamental durante el desarrollo de un sistema, ya que la misma posee un gran impacto sobre la calidad del producto. Por tales razones se creó en la Universidad de las Ciencias Informáticas (UCI) una metodología que detalla la manera de llevar a cabo una evaluación arquitectónica luego que esta haya sido establecida. Para ello se elaboran un conjunto de documentos que describen este proceso. Teniendo en cuenta que la documentación que se obtiene de una correcta evaluación a la arquitectura de un software es relativamente extensa, se hace necesario realizar un almacenamiento correcto de la misma, así como facilitar el acceso a los resultados que se obtienen en ella.

Es por ello que surge la necesidad de desarrollar un sistema automatizado para la gestión de la información que se genera durante la evaluación de la Arquitectura de Software en los Proyectos Productivos de la UCI, con el que se pretende organizar y centralizar la información que se maneja durante este tipo de evaluaciones, tratando con ello de simplificar este procedimiento evaluativo. Para el desarrollo de este sistema se hizo uso de las distintas tendencias y tecnologías actuales, las que permitieron desarrollar un conjunto de artefactos que fundamentan el proceso de desarrollo, tales como: el modelado del negocio, un esbozo de la arquitectura del sistema mediante diagramas del diseño, la implementación del producto, así como pruebas de caja negra que se aplicaron al sistema una vez concluido.

ÍNDICE

INTRODUCCIÓN

El creciente desarrollo de la Informática y las Telecomunicaciones ha dado lugar a que nadie se encuentre exento de los avances que ha enfrentado la sociedad mundial. Por tal motivo, la producción de software, proceso que representa un papel fundamental en esta esfera, se vuelve cada vez más exigente. A medida que transcurre el tiempo las demandas son cada vez mayores y los requisitos para satisfacerlas totalmente, más difíciles de alcanzar.

En la elaboración de un producto de software "la sofisticación y la complejidad pueden producir resultados deslumbrantes cuando un sistema tiene éxito, pero también pueden suponer grandes problemas para aquellos que deben construir sistemas complejos." **(Pressman, 2002)** Es por ello que los desarrolladores de estos sistemas deben sentirse firmemente comprometidos con su trabajo y poner conocimientos y recursos en función de alcanzar un producto de calidad.

Teniendo en cuenta que "La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario" **(Std.IEEE, 610-1990)**, todos los esfuerzos que se pongan en marcha serán pocos cuando realmente se ambiciona insertarse entre los primeros y mejores desarrolladores de software de un mundo donde la calidad del producto es uno de los principales escalones para conseguir el éxito.

En vista de alcanzar este producto de calidad, es fundamental durante la construcción del mismo, diseñar una arquitectura que le de soporte al sistema que se va a implementar. Es esta etapa de diseño una de las más importantes durante todo el período de elaboración del producto, debido a que "La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución." **(Std.IEEE, 1471-2000)** y establece además los fundamentos necesarios para que analistas, diseñadores, implementadores y equipo en general trabajen en una línea común que les permita alcanzar los objetivos propuestos.

Se debe tener en cuenta que no siempre la arquitectura diseñada consigue satisfacer las necesidades del sistema que se desarrolla o solo logra un cumplimiento parcial de las mismas, lo que provoca que el

sistema no responda a las expectativas que se vislumbraron inicialmente. Y es entonces que surge esta interrogante: ¿Cómo asegurar que la arquitectura elegida es la correcta para el software a desarrollar? Una respuesta viable a dicha interrogante es evaluar el diseño arquitectónico propuesto antes que se lleve a cabo la implementación del producto. Evaluar una arquitectura de software constituye poner a prueba el potencial de la arquitectura diseñada y cómo esta logra dar respuesta a los atributos de calidad requeridos por el sistema en cuestión. Por tanto, "si las decisiones arquitectónicas determinan los atributos de calidad del sistema, entonces es posible evaluar las decisiones arquitectónicas con respecto a su impacto sobre dichos atributos." **(Camacho, 2004)** La evaluación de la Arquitectura de Software contribuye a encontrar debilidades, y mientras más temprano se evalúe, más probabilidades existen de evitar cualquier problema que se derive del mal diseño arquitectónico.

En la Universidad de las Ciencias Informáticas (UCI), se trabaja no solo para formar profesionales en la rama de la informática de cualquier parte del país, sino también busca ser un pilar fundamental en el desarrollo y comercialización de software tanto a nivel nacional como internacional. Para dar cumplimiento a esta meta, la universidad trabaja constantemente en vista de desarrollar un producto con calidad, que satisfaga cada una de las necesidades de sus clientes. Aun así, es válido señalar que existen dificultades con relación a la evaluación de la Arquitectura de Software y a la forma en que se documenta la misma.

En la mayoría de los proyectos productivos que se desarrollan hoy en la UCI no se ha fomentado una cultura de evaluación de la arquitectura, lo que provoca que esta se lleve a cabo en menos de la mitad de la producción en dicha institución, generando como resultado de la misma, muy poca o ninguna documentación. Para un mayor control de la evaluación de la arquitectura de los proyectos de la universidad, se desarrolló en el 2009 como Trabajo de Diploma, una metodología que describe como llevar a cabo este procedimiento y da fe del mismo a través de la elaboración de un conjunto de planillas. Esta metodología se puso en práctica en el proyecto Captura y Catalogación de Medias del Departamento Señales Digitales de la Facultad 9, y se pretende con ella evaluar la Arquitectura de Software de los demás proyectos que se desarrollan hoy en la UCI.

Pero aún así la documentación que se generaría como parte del procedimiento de la evaluación de la arquitectura para todos los proyectos productivos de la universidad sería lo suficientemente extensa como para dificultar su organización y centralización así como el acceso continuo a la misma.

Por tales motivos se identificó como **Problema a resolver:** La poca centralización de la documentación que se genera durante la evaluación de la Arquitectura de Software en los proyectos productivos de la UCI trae consigo atraso, desorganización y dificulta el acceso a dicha información.

Después de analizar el problema científico se ha llegado a la conclusión que el **Objeto de estudio** se basa en: La Evaluación de la Arquitectura de Software en la UCI. Del cual se obtiene como **Campo de acción:** Las aplicaciones de gestión de información para la evaluación de la Arquitectura de Software.

En vista de solucionar el problema planteado anteriormente se tiene como **Objetivo general:** Desarrollar una aplicación informática para automatizar la evaluación de la Arquitectura de Software en los Proyectos Productivos en la UCI.

Para el cumplimiento de dicho objetivo se propone la siguiente **Idea a defender**: La implementación de una aplicación para automatizar la evaluación de la Arquitectura de Software en los proyectos de la UCI, permite agilizar la gestión de dicho proceso.

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

1. Definir la documentación asociada a la evaluación de la Arquitectura de Software y a las aplicaciones que tratan la gestión de la información que se genera en dichas evaluaciones.
2. Identificar las necesidades automatizables de la metodología de Evaluación de la Arquitectura.
3. Realizar las actividades de análisis y diseño necesarias para llevar a cabo la implementación del producto.
4. Implementar un producto que cumpla con las especificaciones identificadas y de acuerdo al diseño elaborado.
5. Validar el trabajo mediante el uso de técnicas para este fin.

Con el desempeño exitoso de las tareas anteriores se identifican los **Posibles resultados:**

- Una aplicación capaz de automatizar la evaluación de la Arquitectura de Software para los Proyectos Productivos de la UCI.
- La documentación técnica asociada al desarrollo de la aplicación anterior.

En vista de llevar a cabo las tareas científicas propuestas y alcanzar los resultados esperados se utilizaron los siguientes **Métodos Científicos**:

Métodos Teóricos: “permiten revelar las relaciones esenciales del objeto de investigación, no observables directamente. Participan en la etapa de asimilación de hechos, fenómenos, procesos y en la construcción del modelo e hipótesis de investigación.” **(Alvarez de Sayaz, 1995)**

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

- Histórico-Lógico: que permite conocer y comprender de qué forma se ha desarrollado la evaluación de la Arquitectura de Software, así como sus principales tendencias, y las causas que han dado origen a su comportamiento evolutivo.

Métodos Empíricos: “Describen y explican las características fenomenológicas del objeto, representan un nivel de la investigación cuyo contenido procede de la experiencia y es sometido a cierta elaboración racional.” **(Hernández, 2002)**

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

CAPÍTULO 1: Fundamentación teórica

1.1 Introducción

En el transcurso de este capítulo se realizará una breve valoración sobre la evaluación de la arquitectura de software, así como los conceptos básicos relacionados con el tema. Se pretende resumir en él, la importancia de este tipo de evaluaciones, así como su impacto en el proceso de desarrollo de un software. Se expondrán además, las razones por las cuales se hace tan necesario llevar a cabo este proceso evaluativo en los proyectos de la universidad.

1.2 Arquitectura de Software

Los antecedentes de la Arquitectura de Software se remontan hasta la década de 1960, años en los que solo se acariciaba su concepto. Se conoce que en sus inicios Edsger Dijkstra propuso que se estableciera una estructuración correcta de los sistemas de software antes de proceder a la programación de los mismos; y aunque no utilizó el término arquitectura para describir el diseño conceptual del software, sus conceptos sentaron las bases para lo que luego opinarían otros conocedores del tema.

Es en 1989 con la publicación del libro, “Sistemas de Gran Escala Requieren de un Alto Nivel de Abstracción” (“*Larger Scale Systems Require Higher-Level Abstractions*”), de Mary Shaw, donde se dan los primeros pasos por definir “arquitectura de software”, donde Shaw la define de la siguiente manera:

“…Arquitectura de software es el estudio de la estructura a gran escala y el rendimiento de los sistemas de software. Aspectos importantes de la arquitectura de un sistema; incluye la división de funciones entre los módulos del sistema, los medios de comunicación entre módulos, y la representación de la información compartida.” **(Shaw, 1989)**

Pero es en la década de 1990 donde la Arquitectura de Software alcanza mayor popularidad y donde, con el lanzamiento del libro Introducción a la Arquitectura de Software, (“*An Introduction to Software Architecture*”), de Mary Shaw y David Garlan, se introduce la necesidad de la creación de la Arquitectura de Software como disciplina científica.

La historia de la Arquitectura de Software no se ha desarrollado de manera continua y el gran número de definiciones que se han ido generando con el paso del tiempo amenazan hoy con alcanzar los cuatro dígitos .Se sabe además que ninguna de estas definiciones han sido respaldadas unánimemente, lo que evidencia que su conceptualización sea aún tema de discusión para los arquitectos. Por tal motivo se hace necesario exponer a continuación algunos de los planteamientos más reconocidos.

Entre las definiciones más conocidas se encuentra la dada por Len Bass: "La arquitectura de software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos". **(Bass, 2005)**

En el año 2000 se publica la definición que se ha acordado tomar como la más oficial, publicada en el documento IEEE[1] Std 1471-2000 y adoptada por Microsoft, que procura homogeneizar y ordenar la nomenclatura de descripción arquitectónica y homologa los estilos como un modelo fundamental de representación conceptual: "La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución."

Si se analizan detalladamente cada uno de las definiciones anteriores resulta claramente visible que existen relaciones entre ellos, complementándose unos con otro, con puntos de vista e ideas en común que convergen en que: La Arquitectura de Software es una estructura abstracta del sistema, compuesta por los componentes del mismo, sus propiedades y la manera en que estos interactúan para garantizar el cumplimiento de los requerimientos de dicho sistema.

1.3 Evaluación de la Arquitectura de Software

"La necesidad del manejo de la arquitectura de un sistema de software nace con los sistemas de mediana o gran envergadura, que se proponen como solución para un problema determinado. En la medida que los sistemas de software crecen en complejidad, bien sea por número de requerimientos o por el impacto de los mismos, se hace necesario establecer medios para el manejo de esta complejidad." **(Hofmeister,**

[1] Instituto de Ingenieros Electricistas y Electrónicos.

2000) Por tales motivos la Arquitectura de Software del sistema a desarrollar constituye un importante factor para lograr que este tenga un alto nivel de calidad, así como determina el éxito o fracaso de dicho sistema de software, en la medida que esta cumpla o no con sus objetivos.

Por tanto, "Para reducir tales riesgos, y como buena práctica de ingeniería, es recomendable realizar evaluaciones a la arquitectura." **(Salvador, 2007)**

1.3.1 ¿Qué es una evaluación a la arquitectura?

"Evaluar una arquitectura es realizar una valoración del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos e identificar cuáles son los puntos de riesgos del diseño evaluado." **(Kazman, 2007)**

La arquitectura de software posee un gran impacto sobre la calidad de un sistema, por lo que es muy importante estar en capacidad de tomar decisiones acertadas sobre ella, en diversos tipos de situaciones, entre las cuales destacan **(Bengtsson, 1999)**:

- Comparación de alternativas similares
- Comparación de la arquitectura original y la modificada
- Comparación de la arquitectura de software con respecto a los requerimientos del sistema
- Comparación de una arquitectura de software con una propuesta teórica
- Valoración de una arquitectura en base a escalas específicas

En fin, evaluar una arquitectura de software consiste en poner a prueba el potencial del diseño arquitectónico propuesto, con el propósito de lograr dar respuesta a los atributos de calidad requeridos por el sistema.

1.3.2 ¿Por qué evaluar una Arquitectura de Software?

La necesidad de evaluar la arquitectura de software nace del impacto que esta posee sobre la calidad del sistema. Si se tiene en cuenta que del buen diseño arquitectónico depende el cumplimiento de los atributos de calidad y el éxito en sí del producto; entonces es preciso que se tomen las medidas necesarias para garantizar que este diseño, haya sido desarrollado de la manera correcta. La principal

tarea que se debe llevar a cabo para asegurar que la arquitectura propuesta cumple con las necesidades del sistema, es evaluar dicha arquitectura.

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

1.3.2.1 Atributos de calidad

De acuerdo al estándar IEEE 610.12-1990, "un atributo de calidad es una característica que afecta la calidad de un elemento." Es válido aclarar que en la definición anterior, el término "característica" se refiere a aspectos no funcionales, mientras que el término "elemento" se refiere a un componente o sistema.

A grandes rasgos, Bass establece una clasificación de los atributos de calidad en dos categorías **(Camacho, 2004)**:

- **Observables vía ejecución:** aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución. Por ejemplo: rendimiento, confiabilidad, disponibilidad, tolerancia a fallas.
- **No observables vía ejecución:** aquellos atributos que se establecen durante el desarrollo del sistema. Por ejemplo: facilidad de modificación, facilidad de re-utilización, flexibilidad.

1.4 Técnicas de Evaluación de la Arquitectura

"Las técnicas de evaluación de arquitectura son herramientas que predicen el comportamiento de la arquitectura de software durante su diseño. Existen dos clasificaciones de estas, las técnicas cuantitativas y las cualitativas." **(Bosch, 2006)** Dentro de las técnicas de evaluación cuantitativas se encuentran las métricas, los prototipos, las basadas en simulación, las basadas en experiencias y basadas en modelos matemáticos. Por otro lado, dentro de las técnicas cualitativas, están las representadas por escenarios, las listas de verificación y los cuestionarios.

A continuación se explicarán algunas de las técnicas de evaluación anteriormente mencionadas, después de hacer una selección entre los dos grupos, con el objetivo analizar las principales características de ambos enfoques.

1.4.1 Técnica de evaluación basada en escenarios

Se conoce que "un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con éste." **(Kazman, 2001)** Según lo planteado por Kazman está compuesto por tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario donde se detalla lo que el involucrado hace para iniciar la interacción con el sistema. El contexto describe qué sucede en el sistema al momento del estímulo .La respuesta describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado.

Los escenarios proveen un vehículo que permite concretar y entender atributos de calidad. Varios autores coinciden en la importancia del uso de los escenarios, no sólo para efectos de la evaluación de arquitecturas de software.

Dentro de las principales ventajas de su uso se encuentran:

- Son simples de crear y entender.
- Son poco costosos y no requieren mucho entrenamiento.
- Son efectivos.

Las técnicas de evaluación basadas en escenarios cuentan actualmente con dos relevantes instrumentos: el Árbol Utilidad de (*Utility Tree*) propuesto por Kazman, y los Perfiles (*Profiles*), propuestos por Bosch. Debido a su sencillez y a la representación jerárquica de los atributos de calidad que inciden en un escenario, se reconoce actualmente que el instrumento más utilizado es el Árbol Utilidad.

1.4.2 Técnica de evaluación basadas en simulación

"La evaluación basada en simulación utiliza una implementación de alto nivel de la arquitectura de software. El enfoque básico consiste en la implementación de componentes de la arquitectura y la

implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse." **(Bosch, 2000)** Su propósito es evaluar el comportamiento de la arquitectura bajo circunstancias diversas. Una vez disponibles estas implementaciones, se usarán los respectivos perfiles para la evaluación los atributos de calidad.

En términos de los instrumentos asociados a las técnicas de evaluación basadas en simulación, se encuentran los lenguajes de descripción arquitectónica y los modelos de colas.

La técnica de evaluación basada en simulación presenta varias dificultades, una de ellas es que la exactitud de los resultados de la evaluación, depende de la exactitud del perfil utilizado para evaluar el atributo de calidad, y de la precisión con la que el contexto del sistema simula las condiciones del mundo real, lo que tributa a que la evaluación no tenga un nivel alto de precisión, debido a que sus resultados dependen de factores subjetivos, que pueden ser logrados o no.

Otra de las desventajas de esta técnica de evaluación consiste en que su aplicación durante un proceso de desarrollo de software provoca un aumento considerable de esfuerzo y tiempo, por el alto nivel de especificación que se necesita en cada uno de sus pasos y los conocimientos avanzados con que debe contar el equipo de evaluación para su aplicación. Además esta técnica es utilizada para evaluar solo requerimientos de calidad operacional, como desempeño y confiabilidad.

1.4.3 Técnica de evaluación basada en modelos matemáticos

"La evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico, y se presentan como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados, utilizando los resultados de uno como entrada para el otro." **(Bosch, 2000)**

Dentro de los instrumentos con que cuentan las técnicas basadas en modelos matemáticos para la evaluación de las arquitecturas de software, se encuentran las Cadenas de Markov y los Diagramas de Bloque de Fiabilidad (*Reliability Block Diagrams*).

Es importante resaltar que dentro de las desventajas que presenta esta técnica de evaluación se resalta la carencia de modelos matemáticos apropiados para los atributos de calidad que resultan relevantes para la arquitectura de software, además del hecho de que requiere esfuerzos sustanciales a la hora de desarrollar un modelo simulación completo, aparejado a la necesidad de un elevado nivel de conocimientos para la adopción de dichos modelos.

1.4.4 Técnica de evaluación basada en experiencia

"En muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño. Aunque todas estas experiencias se basan en evidencia anecdótica; es decir, basada en factores subjetivos como la intuición y la experiencia. Sin embargo, la mayoría de ellas puede ser justificada por una línea lógica de razonamiento, y pueden ser la base de otros enfoques de evaluación." **(Bosch, 2000)**

Las técnicas de evaluación basadas experiencia se dividen en dos tipos: la evaluación formal, que es aquella que se realiza por los arquitectos de software durante el proceso de diseño y la realizada por equipos externos de evaluación de arquitecturas.

Ambas técnicas de evaluación de arquitecturas carecen de basamentos teóricos, por lo que se apoyan en factores específicos de las personas, factores que pueden o no ser efectivos. La dificultad fundamental de esta técnica consiste en la variabilidad y desconfianza de los resultados que se obtienen. Es de vital importancia para su uso, la existencia de personas capacitadas y con experiencia, que hayan acumulado los conocimientos suficientes que les permita brindar un razonamiento lógico a las decisiones arquitectónicas tomadas. O sea, sería muy útil como criterio durante la evaluación de la arquitectura de un software, pero insuficiente para tomar decisiones basándose únicamente en este tipo de análisis.

1.4.5 Consideraciones sobre las Técnicas de Evaluación

Las técnicas de evaluación, tanto las cualitativas como las cuantitativas, posibilitan valorar el diseño arquitectónico desde las primeras fases de su desarrollo, y constituyen un medidor que permite conocer cuan acertada o no es la arquitectura evaluada, además de servir de apoyo al equipo de desarrollo en la toma de decisiones a través de la evaluación que las mismas realizan a los atributos de calidad. Se puede

llegar a la conclusión que las técnicas de evaluación cualitativas demandan menores esfuerzos durante su implementación que las técnicas cuantitativas, además de resultar más simples y entendibles para los involucrados en el proceso de evaluación.

"Las técnicas generalmente, no son utilizadas para realizar una evaluación arquitectónica de manera directa. Su principal utilidad es apoyar la evaluación realizada por un método."**(Bass, 2005)** Se considera que las más difundida y usada actualmente es la técnica de evaluación basada en escenarios, debido a su fácil implementación y simplicidad.

1.5 Métodos de Evaluación de la Arquitectura

"Los métodos de arquitectura surgen de la necesidad de evaluar el potencial de una arquitectura de software para soportar los requisitos de software. Muchos han sido los métodos propuestos para el análisis de las arquitecturas, en su mayoría adecuaciones a contextos específicos basados en métodos ya diseñados." **(Lane, 2006)** "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." **(Kazman, 2001)** Por tales razones, han sido propuestos múltiples métodos de evaluación. A continuación se expondrán algunos de los más importantes, teniendo en cuenta la representatividad que tienen estos en la actualidad.

1.5.1 Método de Análisis de Arquitectura de Software (SAAM)

"El Método de Análisis de Arquitecturas de Software (*Software Architecture Analysis Method*, SAAM) es el primero que fue ampliamente promulgado y documentado." **(Kazman, 2005)** El mismo "tiene como objetivo evaluar un grupo de atributos de calidad que debe lograr la arquitectura de software, a través del uso de escenarios. En la práctica ha demostrado ser útil para la rápida evaluación de atributos de calidad como: modificabilidad, portabilidad y extensibilidad. En resumen se puede afirmar, que al evaluar una arquitectura, SAAM indica los puntos de fortalezas y debilidades, junto con los puntos de la arquitectura que no cumple con los requisitos de modificabilidad." **(Kazman, 2007)**

El método SAAM se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro.

Como entrada principal se hace necesaria:

- Alguna forma de descripción de la arquitectura a ser evaluada.

Como salidas de la evaluación del método SAAM se obtiene **(Kazman, 2005)**:

- Una proyección sobre la arquitectura de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema.
- Entendimiento de la funcionalidad del sistema, e incluso una comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación.

1.5.2 Método de Análisis de Interacciones de la Arquitectura (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. El nombre del método ATAM surge del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros; esto es, los tipos de acuerdos que se establecen entre ellos."**(Kazman, 2001)**

"El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados (…) estos elementos 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, e integrarse con otros sistemas."**(Kazman, 2001)**

Este método de evaluación se encarga de analizar cómo la arquitectura logra satisfacer los atributos de calidad: modificabilidad, portabilidad, extensibilidad e integrabilidad. Proporciona además información sobre el cumplimiento de cada uno de estos atributos de calidad involucrados, teniendo en cuenta las dependencias que se crean entre ellos.

Este método es considerado como el más abarcador de todos los métodos de evaluación. Una de sus fortalezas consiste en el manejo de la identificación de los riesgos de la arquitectura diseñada, mediante la identificación de puntos de sensibilidad, de desventaja y de riesgo. Al igual que los demás métodos de evaluación no utiliza un modelo de calidad para la selección, refinamiento y medición de los atributos de calidad. No obstante, el instrumento del árbol de utilidad propone un nivel de refinamiento de los atributos

más abarcador, que incluye hasta la definición de métricas con criterios de máximos y mínimos, permitiendo un nivel de medición aceptable.

"Los beneficios de la utilización de este método, es la organizada interacción que se establece entre los actores, arquitectos y equipo de evaluación, así como toda la documentación arquitectónica que genera el proceso de evaluación." **(Ionita, 2006)**

1.5.3 Método de Análisis de Costo-Beneficio (CBAM)

"El método de análisis de costos y beneficios (*Cost-Benefit Analysis Method*, CBAM) es un marco de referencia que no toma decisiones por los involucrados en el desarrollo del sistema. Por el contrario, ayuda en la elicitación y documentación de los costos, beneficios e incertidumbre, y provee un proceso de toma de decisiones racional. Uno de los elementos que introduce el método son las llamadas estrategias arquitectónicas, que consisten en posibles opciones para la resolución de conflictos entre atributos de calidad presentes en una arquitectura." **(In, 2001)**

Este método cuenta con dos fases: la primera es solo necesaria cuando existe un amplio conjunto de propuestas arquitectónicas que se debatirán, y se necesita reducir el número de arquitecturas candidatas. No obstante, el proceso de evaluación se desarrolla como tal en la segunda fase, conocida con examen detallado. Para ambas fases se establecen nueve pasos en la evaluación.

El método CBAM, al igual que SAAM y ATAM, se basa en la evaluación de la arquitectura, enfatizando en los beneficios económicos de las decisiones arquitectónicas.

1.5.4 Revisión Activa para el Diseño Intermedio (ARID)

"El método Revisión Activa para el Diseño Intermedio (*Active Reviews for Intermediate Designs,* ARID) es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura." **(Kazman, 2001)** Según el criterio de varios autores, el ARID es un híbrido entre los métodos *Active Design Review* (ADR), utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos, y el ATAM, explicado anteriormente.

Según lo propuesto por Kazman, “tanto el ADR como el ATAM proveen características útiles para el problema de la evaluación de diseños preliminares, dado que ninguno por sí solo es conveniente.” **(Kazman, 2001)** En el método de evaluación ARD, los involucrados reciben una documentación detallada y completan cuestionarios, cada uno de ellos por separado; sin embargo el ATAM está orientado a la evaluación de toda una arquitectura.

“Ante esta situación, y la necesidad de evaluación en las fases tempranas del diseño, se propone la utilización de las características que proveen tanto ADR como ATAM por separado. De ADR, resulta conveniente la fidelidad de las respuestas que se obtiene de los involucrados en el desarrollo. Así mismo, la idea del uso de escenarios generados por los involucrados con el sistema es tomada del ATAM.” **(Kazman, 2001)** Al combinar las filosofías de ambos métodos surge ARID, con el propósito de realizar una evaluación temprana de los diseños de una arquitectura de software.

En general, el método de evaluación ARID está compuesto por fuertes cualidades de los métodos ADR y ATAM. El mismo surge prácticamente de la unión de estos dos métodos, con el objetivo de evaluar el diseño de la arquitectura en fases tempranas de su definición. Por tal razón, permite reunir las partes interesadas y los diseñadores en las primeras fases de desarrollo del software. De esta forma, los involucrados generan los escenarios con el fin de realizar un análisis de capacidad a la arquitectura, mientras que el uso de escenarios promueve una lluvia de ideas por parte de los diseñadores, lo que permite la valoración nuevos diseños y evaluaciones. Al terminar el proceso de evaluación se realizan los ajustes necesarios en el diseño arquitectónico.

Las principales desventajas de este método yacen en la manera en que realiza la evaluación del diseño arquitectónico, basándose únicamente en el análisis de los componentes de la arquitectura, sin tener en cuenta las conexiones que se establecen entre ellos. Aparejado a esto, es de señalar que no contempla ningún atributo de calidad, o sea, solo evalúa la conveniencia del diseño arquitectónico.

1.5.5 Método de Análisis del Nivel de Modificación de la Arquitectura (ALMA)

“El Método de análisis del nivel de modificación de la arquitectura (*Architecture Level Modifiability Analysis*, ALMA) tiene como objetivo la realización de un análisis correcto basado en la hipótesis de la evaluación de la arquitectura de software a través de la modificabilidad de un conjunto de indicadores: la

predicción de los costes de mantenimiento, la evaluación de los riesgos, entre otros. En caso de evaluar y comparar diferentes sistemas, el análisis que realiza el método ALMA apoya el logro de la calidad arquitectónica y la selección de la alternativa correcta. Es por esto que este método utiliza como técnica de evaluación, los escenarios de cambio, proporcionados por las partes interesadas." **(Bengtsson, 2008)**

"ALMA es un método de evaluación orientado a metas, que se apoya en el uso de escenarios de cambio, los cuales escriben los eventos posibles que provocarían cambios al sistema, y como se llevarían a cabo estos."**(Salvador, 2007)**

Entre las consideraciones más importantes que aporta este método está la utilización de las vistas arquitectónicas propuestas por Kruchten, además del uso de la notación UML con el fin de representar la arquitectura de software, lo que proporciona el entendimiento de los participantes en las sesiones de evaluación propuestas por dicho método.

Este método solo ofrece atención al tratamiento del atributo de calidad modificabilidad, por tanto desecha otros atributos relevantes para la calidad del producto final, tales como la confiabilidad y la portabilidad del sistema, lo que es considerado su principal desventaja.

1.5.6 Método de Análisis de Familias de Arquitectura (FAAM)

"El Método de análisis de familias de arquitectura (*Family-Architecture Assessment Method,* FAAM), tiene como objetivo establecer un proceso (con el apoyo de directrices, indicadores, recomendaciones y procesos) para la evaluación de la información de la familia de arquitecturas de sistemas." **(Ionita, 2006)**

Diferente de otros métodos, FAAM contribuye en **(Ionita, 2006)**:

- Implicar activamente a los productos de la familia interesados en la el proceso de creación del producto,
- Centrándose en los atributos de calidad interoperabilidad y extensibilidad de la información de las familias de sistema,
- Haciendo hincapié en la práctica de los conocimientos y mecanismos de técnicas que permitan a los equipos de desarrollo aplicar el método dentro de las organizaciones.

La principal fortaleza de este método radica en que centra su atención en atributos de calidad que no han sido tratados en métodos de evaluación anteriores, como son: la interoperabilidad y la extensibilidad. Con el surgimiento de las redes de comunicación, el número de sistemas distribuidos ha crecido considerablemente dentro de los desarrollos de software, por tal razón la existencia de un método que permita una evaluación de estas arquitecturas integradoras es prácticamente obligatoria.

Una de las desventajas de este método consiste en que solo evalúa los atributos de interoperabilidad y extensibilidad, sin reparar en el tratamiento de atributos de calidad como la eficiencia, muy necesario para la evaluación de arquitecturas distribuidas. Otro problema que presenta es que abarca contextos demasiado específicos dentro del grupo de productos de software, por lo que solo se utiliza para un grupo pequeño de sistemas.

1.5.7 Consideraciones sobre los Métodos de Evaluación

Luego de analizar algunos de los métodos de evaluación, se puede asegurar que los mismos logran una coherencia en el orden de los pasos que se ejecutan a la hora de llevar a cabo la evaluación de una arquitectura, realizando un análisis detallado del diseño arquitectónico del sistema que se evalúa. A pesar de esto, se considera que la principal desventaja de dichos métodos es que, en su mayoría, se enfocan en uno, o muy pocos atributos de calidad, y alejan su atención de la comprobación de los otros, o lo hacen de forma superficial, lo que trae consigo que muy pocos logren una medición precisa del comportamiento de los mismos.

Otro factor a destacar, es que dichos métodos no verifican el uso de patrones y estilos en la arquitectura evaluada, elementos relevantes a la hora de medir el comportamiento de los atributos, pues el uso de los mismos asegura en cierta medida el cumplimiento de un conjunto de atributos de calidad que deben estar presentes.

1.6 Evaluación de la Arquitectura de Software en la UCI

En vista de convertirse en una industria de software reconocida tanto a nivel nacional como internacional, la UCI centra sus fuerzas en desarrollar un producto de alta calidad, que logre satisfacer las expectativas y requerimientos de sus clientes. Pero como en toda entidad joven, se evidencian aún un conjunto de

dificultades, que surgen producto de la poca atención que se centra sobre la evaluación de los diseños arquitectónicos, y que trae consigo que decaiga la calidad del producto desarrollado, aparejado a pérdidas de tiempo y esfuerzo.

A continuación se muestra mediante un diagrama Causa-Efecto las deficiencias existentes en el proceso de producción de la UCI:

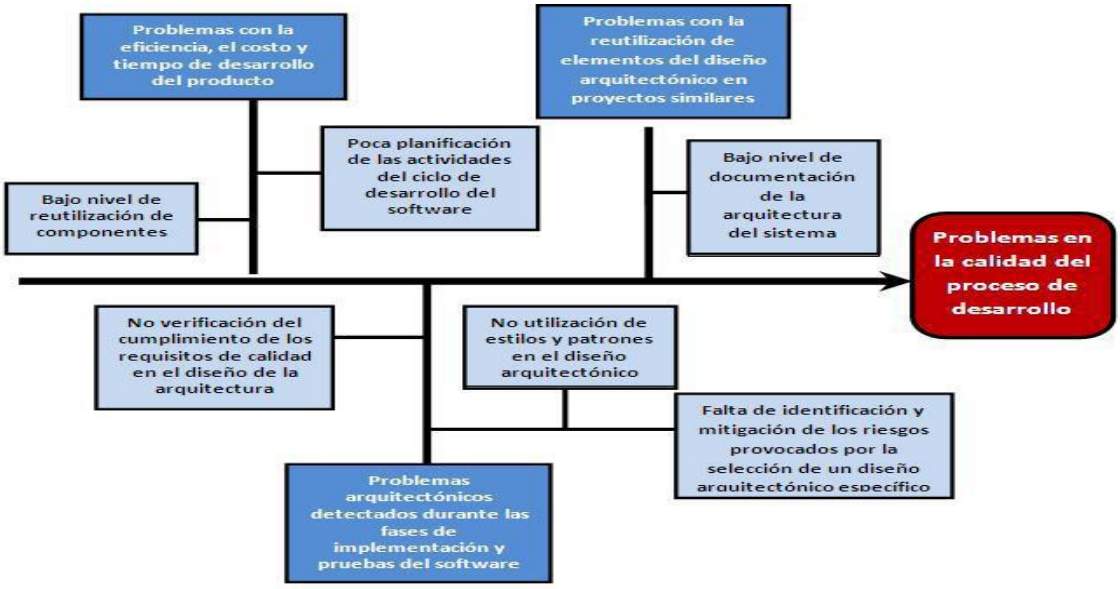

Figura 1. Diagrama Causa-Efecto **(Vigil, 2009)**

De acuerdo con encuesta realizas el pasado año en varios de los proyectos productivos, se pudo evidenciar que “no es una práctica de los equipos de desarrollo el evaluar la Arquitectura; y en los casos aislados donde se logra una evaluación, la documentación referente a este proceso es deficiente o simplemente no existe.”**(Rondón, 2009)**

Con el desarrollo de estas encuestas se detectó además que en su mayoría, los proyectos encuestados coinciden en el hecho de priorizar la documentación del proceso de evaluación, con el objetivo de utilizar toda esa experiencia en el desarrollo de proyectos con cortes similares que se realizarán posteriormente.

Los encuestados encaminaron sus inquietudes hacia la necesidad de que se documentaran **(Rondón, 2009)**:

- La definición de los involucrados en el proceso de evaluación.
- La descripción de la arquitectura propuesta.
- Una valoración de la arquitectura propuesta respecto a otras candidatas.
- La definición de los atributos de calidad a evaluar en la arquitectura propuesta.
- Una valoración del comportamiento de los atributos de calidad evaluados.
- Los puntos críticos de la arquitectura evaluada.
- La identificación de posibles cambios.

"Por estos motivos, surge la decisión del diseño de una metodología[2] de evaluación de arquitecturas de software, que constituya una guía del proceso evaluativo, conformada por actividades, involucrados, artefactos, herramientas de apoyo y un conjunto de buenas prácticas; que aseguren el logro de la calidad de los procesos de desarrollo que lleva a cabo la universidad." **(Vigil, 2009)**

1.7 Propuesta de Metodología de Evaluación de Arquitectura de Software

La necesidad de diseñar una metodología de evaluación de arquitecturas de software en la UCI, surge de las dificultades que existen hoy en proyectos productivos de esta entidad, ocasionados en su mayoría por la poca práctica de evaluación del diseño arquitectónico, lo que ha provocado falta de eficiencia y calidad en el proceso de desarrollo.

"La metodología de evaluación tiene como objetivo principal la valoración de la arquitectura del sistema a desarrollar, midiendo el comportamiento de los atributos de calidad e identificando los riesgos que provoca el diseño arquitectónico establecido." **(Vigil, 2009)**

La metodología está compuesta por un conjunto de componentes de los que se nutre el proceso evaluativo. A continuación se enumeran cada uno de ellos y se explica brevemente de qué forma están presentes durante la evaluación:

[2] Una metodología hace referencia a una guía de actividades, procedimientos, productos, técnicas y participantes que interactúan para alcanzar un fin común. **(Cueva, 2007)**

- **Requisitos del software:** es el componente encargado de definir las funcionalidades y características que debe cumplir el producto que se desarrolla. Los requisitos de software están estrechamente relacionados con los requerimientos de calidad con los que debe cumplir el software, y es necesario que el diseño arquitectónico garantice el cumplimiento de los mismos ya que constituyen la base del desarrollo del producto.

- **Representación de la arquitectura de software y vistas arquitectónicas:** consiste una representación gráfica y documentada del diseño arquitectónico que argumente de forma clara las distintas decisiones. Luego de los requisitos del software, este constituye el componente de entrada más importante.

- **Elementos acertados de los métodos y técnicas de evaluación:** se realiza un análisis de las dificultades y beneficios de un conjunto de métodos y técnicas de evaluación de arquitectura de software; teniendo en cuenta para el desarrollo de la metodología: la sencillez y simplicidad de la técnica basada en escenarios y la manera de seleccionar los puntos de riesgo, sensibilidad e intercambio del método ATAM. En general, la metodología logra un nivel de flexibilidad que permite la utilización de varios métodos de evaluación de arquitectura en la determinación de los riesgos, pero sugiere principalmente el uso de ATAM como método de evaluación.

- **Modelo de calidad:** teniendo en cuenta que una de las dificultades observada en los métodos de evaluación analizados fue la superficialidad con la que se realiza la medición de los atributos de calidad, se propone utilizar un modelo de calidad en las distintas fases de la metodología con el objetivo de lograr la selección y refinamiento de dichos atributos, así como una medición cuantitativa de los mismos.

- **Estilos y patrones arquitectónicos:** ya que otra de las dificultades presentes en los métodos de evaluación es que nunca toman como criterio de medida la verificación de estilos y patrones de arquitectura, y teniendo en cuenta que uno de los problemas detectados en los proyectos de la universidad es la no utilización de estos estilos y patrones en la definición de la arquitectura, se decide incluir la valoración de los mismos en el proceso que guía la metodología, llevándola a cabo durante su fase de desarrollo.

- **Expediente de evaluación arquitectónica:** producto a la poca documentación que se genera acerca de la definición y seguimiento de la arquitectura del sistema durante el proceso de desarrollo en los proyectos de la UCI, se diseñaron un conjunto de planillas con el objetivo de estandarizar toda la información que surge del proceso evaluativo, y de esta forma darle solución al problema existente.

1.7.1 Descripción de la evaluación

La metodología de evaluación de arquitectura a la que se hace mención cuenta con dos áreas de procesos fundamentales, el proceso de evaluación y el proceso de documentación. Dicho proceso de evaluación está compuesto por cuatro fases que pueden verse como subprocesos dentro del mismo: la fase de concepción, la fase de presentación, la fase desarrollo donde se llevan a cabo cada una de las valoraciones arquitectónicas y por último la fase exposición de resultados, en las mismas se desarrollan diferentes actividades. En cada una de estas fases se generan artefactos que constituyen la base del proceso de documentación y que se desarrolla a la par del proceso de evaluación.

Otro importante factor que incide en el proceso evaluativo es el grupo de involucrados, compuesto por el líder de proyecto, arquitectos, analistas, desarrolladores, asesores de calidad y clientes, teniendo en cuenta que no participan en su totalidad todas los personas relacionadas con los roles mencionados, sino una representación de los mismos.

1.8 Conclusiones

A lo largo de este capítulo se ha hecho referencia a la importancia que encierra una evaluación arquitectónica y al impacto que esta provoca durante el proceso de desarrollo de un software. Se han tratado además conceptos básicos muy relacionados con el tema, así como una serie de técnicas y métodos de evaluación que resultan muy útiles a la hora de evaluar un diseño arquitectónico.

Se han mencionados un conjunto de dificultades existentes hoy en los proyectos productivos de la UCI, que atentan contra el buen funcionamiento del proceso de desarrollo, inducidos en su mayoría por la poca atención que se le presta a la evaluación del diseño de las arquitecturas, y que han incidido de forma directa en la calidad del producto.

CAPÍTULO 2: Tendencias y tecnologías actuales

2.1 Introducción

Con el desarrollo de la Informática y las Telecomunicaciones (TICs) han surgido un sinnúmero de aplicaciones informáticas y han evolucionado otras ya existentes, que brindan un conjunto de herramientas prácticamente indispensables a la hora de enfrentarse al desarrollo de un producto de software.

Con el propósito de dar cumplimiento al objetivo general propuesto, se hace necesario analizar las tendencias y tecnologías que se utilizan actualmente, para así decidir cuáles serán convenientes a utilizar en el desarrollo de una aplicación para la automatización de los documentos generados en la evaluación de la Arquitectura de Software en los Proyectos Productivos de la UCI. Con este fin se hará una valoración en este capítulo acerca de dichas tecnologías, donde se definirán conceptos, características y utilidades de las mismas, y se decidirá cuáles son de mayor beneficio a la hora de desarrollar el sistema propuesto.

2.2 Metodologías de desarrollo de software

Las metodologías surgen como una alternativa que facilita el buen desarrollo de un software, debido a que es esta una tarea bastante complicada. Las mismas definen:"Quién" está realizando "Qué", "Cuándo" lo lleva a cabo y "Cómo" lo está haciendo, teniendo como objetivo principal imponer disciplina sobre el proceso de desarrollo y hacerlo más predecible y eficiente.

Actualmente existen un gran número de propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Un ejemplo de las mismas son las llamadas tradicionales o robustas, que se centran específicamente en controlar el proceso, y que han demostrado ser bastante factibles a la hora de desarrollar grandes sistemas, pero no ofrecen una buena solución para proyectos donde el entorno es volátil y donde no se conocen los requisitos con precisión.

Como respuesta a los problemas que surgen al utilizar metodologías tradicionales surgieron otras metodologías encaminadas a adaptarse a la realidad del desarrollo de software, llamadas metodologías

ágiles. Estas nuevas metodologías buscan un justo medio entre ningún proceso y demasiado proceso, proporcionando simplemente suficiente proceso para que el esfuerzo valga la pena.

2.2.1 Metodología robusta

2.2.1.1 Proceso Unificado de Desarrollo (RUP)

"El Proceso Unificado de Desarrollo (*Rational Unified Process*, RUP) es una metodología para la ingeniería de software que va más allá del mero análisis y diseño orientado a objetos para proporcionar una familia de técnicas que soportan el ciclo completo de desarrollo de software. El resultado es un proceso basado en componentes, dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental." **(Jacobson, et al., 2000.)**

Sus principales elementos son:

- Trabajadores (Quién)
- Actividades (Cómo)
- Artefactos (Qué)
- Flujo de actividades (Cuándo)

En RUP se agruparon las actividades en grupos lógicos definiéndose nueve flujos de trabajo principales. Los seis primeros se conocen como flujos de ingeniería y los tres restantes como flujos de apoyo. Cuenta además con cuatro faces que se desarrollan en iteraciones, involucrando actividades de todos los flujos de trabajo.

2.2.2 Metodologías ágiles

2.2.2.1 Programación Extrema (XP)

La Programación Extrema (*Extreme Programming*, XP) fue ideada por Kent Beck en 1996, durante un proyecto que se desarrolló en la empresa Chrysler para gestionar sus nóminas. Es con la publicación del libro "*Extreme Programming Explained - Embrace Change*", en octubre de 1999, que se constituye el nacimiento formal de esta metodología. XP es una metodología ligera, eficiente y flexible, que posee bajo

riesgo. La misma persigue principalmente satisfacer en su totalidad las expectativas del cliente, por lo que debe responder a sus necesidades de forma rápida, sin importar que se realicen cambios al final del ciclo de la programación. XP se centra además en suscitar el trabajo en equipo, y potenciar las relaciones interpersonales como elemento clave para alcanzar el éxito.

Dentro de sus características principales se encuentran **(Calero, 2003)**:

- Desarrollo iterativo e incremental: se llevan a cabo pequeñas mejoras, unas tras otras.
- Integración del equipo de desarrollo con el cliente: es recomendable que un representante del cliente trabaje junto al equipo de desarrollo.
- Propiedad del código compartida: no busca la división de las responsabilidades durante el desarrollo de los módulos en diferentes grupos de trabajo, sino que promueve que todo el personal pueda corregir y extender cualquier parte del proyecto.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas.

2.2.2.2 Proceso Unificado Ágil (AUP)

El Proceso Unificado Ágil (*Agile Unified Process*, AUP) se conoce como una versión simplificada de RUP, en el que se describe de forma simple el enfoque del desarrollo de software utilizando técnicas y conceptos ágiles, pero aún así manteniendo fidelidad a RUP. Es una mezcla entre XP y RUP tradicional, un proceso ágil que incluye actividades y artefactos utilizados comúnmente.

Las disciplinas se llevan a cabo de manera sistemática, y se describen de la siguiente forma **(De Rogatis, y otros, 2004)**:

- **Modelado:** su objetivo principal es entender el negocio de la organización y tener dominio del problema que aborda el proyecto, para así poder definir una solución viable que haga frente a dicho problema.
- **Implementación:** transforma el modelo en código ejecutable para posteriormente llevar a cabo un nivel básico de las pruebas.
- **Prueba:** realiza una evaluación objetiva con el fin de garantizar la calidad del producto.
- **Despliegue:** se enfoca en entregar el sistema a los usuarios finales.

- **Gestión de la Configuración:** gestiona el acceso a los artefactos producidos por el proyecto, controlando y gestionando los cambios que ocurran en ellos.
- **Gestión de Proyectos:** su principal propósito es dirigir las actividades que se llevan a cabo en el proyecto.
- **Entorno:** se centra en apoyar el buen funcionamiento del proceso de desarrollo, poniendo a disposición del equipo las herramientas necesarias en el momento indicado.

2.2.2.3 Selección de la metodología de desarrollo

Teniendo en cuenta que se persigue desarrollar una aplicación que no cuenta con gran complejidad, con un pequeño equipo de desarrollo y en un período de tiempo relativamente corto, se decidió que sería acertado utilizar una metodología ágil para su desarrollo, seleccionando entre estas a AUP, debido a que consiste en una mezcla acertada entre la metodología ágil XP, para la que la administración guarda recelos debido a que no muestra explícitamente cómo crear algunos de los artefactos que se desean ver, y la metodología robusta RUP, que cuenta con una gran cantidad de artefactos al punto de resultar tedioso a los desarrolladores.

Con el propósito de lograr una documentación que no sea demasiado extensa pero tampoco demasiado pobre, se escogió AUP, una metodología que adopta muchas técnicas ágiles de XP y otros procesos que conservan aún la formalidad de RUP.

2.3 El Lenguaje Unificado de Modelado (UML)

El Lenguaje de Modelado Unificado (*Unified Modeling Language*, UML) es la sucesión de un conjunto de métodos de análisis y diseño orientados a objetos que aparecen a finales de la década 1980 y principios de los años 90. “UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software.” **(Jacobson, y otros, 2000)**

La definición anterior proporciona las principales funciones del UML:

- Visualizar: Describe gráficamente un sistema, de manera que resulte de fácil comprensión.
- Especificar: Especifica las características del sistema que se desarrolla antes de pasar a la construcción del mismo.

- Construir: Basándose en los modelos especificados se realiza la construcción de los sistemas diseñados.
- Documentar: Los elementos gráficos constituyen la documentación del sistema, y pueden servir para su revisión futura.

2.4 Herramientas de Ingeniería de Software Asistidas por Computadora (CASE)

Las herramientas de Ingeniería de Software Asistidas por Computadoras (*Computer Aided Software Engineering*, CASE) no son más que aplicaciones informáticas que persiguen incrementar la productividad y calidad durante el desarrollo de un software mediante la reducción de tiempo, esfuerzo y dinero. Estas automatizan el dibujo de diagramas y ayudan en la creación de relaciones en la BD, provocando que el trabajo de diseño del software sea más fácil y agradable. El principal objetivo de estas herramientas es proporcionar un lenguaje para describir el sistema general, que sea lo suficientemente explícito para generar todos los programas necesarios.

2.4.1 Rational Rose Enterprise Edition

Rational Rose Enterprise Edition es el producto más completo de la familia Rational Rose, incluye, al igual que el resto de los productos de esta familia, soporte UML. Es una herramienta propietaria de diseño de software, que se encarga de llevar a cabo la automatización de los sistemas para la posterior generación de código, o sea, realiza la modelación visual y construcción de componentes de dichos sistemas.

Incluye las siguientes características **(González, y otros, 2007)**:

- Permite el modelado en UML para diseñar BD, que integra los requisitos de datos y aplicaciones mediante diseños lógicos y analíticos.
- Proporciona al equipo de desarrollo un lenguaje común de modelado, que facilita la rápida creación de un software de calidad.
- Incluye funciones de visualización, modelado y herramientas para desarrollar aplicaciones Web.
- Posibilita la integración con otras herramientas de desarrollo de IBM Rational.
- Consiente la generación de código a partir de modelos Ada, ANSI C++, C++, CORBA, Java/J2EE, Visual C++ y Visual Basic.

- Capacidad de análisis de calidad de código.

2.4.2 Visual Paradigm para UML

Visual Paradigm es una herranienta UML profesional, que contituye una aternativa factible para trabajar con el lenguaje de modelado UML, y que soporta todo el ciclo de vida del desarrollo de un software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Coopera con la rápida construcción de aplicaciones de calidad a un menor coste. Admite el dibujo de todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Entre sus características principales se destacan:

- Es una herramienta multiplataforma
- Posibilita la integración en los principales IDEs de desarrollo.
- Soporta varios lenguajes en la Generación de Código e Ingeniería Inversa (C++, PHP, Python, entre otros)

2.4.3 Selección de la herramienta CASE

Entre las herramientas CASE analizadas se seleccionó para la construcción de la aplicación a Visual Paradigm, ya que constituye una herramienta factible para trabajar con el lenguaje de modelado UML. Resulta de gran ayuda a la hora de desarrollar aplicaciones de calidad en menor tiempo y costo. Es una herramienta de fácil instalación y uso, lo que apoya la comunicación del equipo de desarrollo y contribuye a alcanzar un producto de excelencia. Entre sus características se destacan que es una herramienta multiplataforma y colaborativa, además de proporcionar la integración con los principales entornos de desarrollo integrados, como por ejemplo Eclipse y NetBeans.

2.5 Arquitectura Cliente-Servidor

La arquitectura Cliente-Servidor es básicamente un modelo para el desarrollo de sistemas de información, donde existe un cliente que realiza peticiones a otro programa, al que se denomina servidor, y este se encarga de dar respuesta a dichas peticiones. Este tipo de transacciones se encuentran divididas en procesos independientes, que cooperan entre sí con el fin de intercambiar información, recursos o

servicios. Se nombra cliente al proceso que inicia el diálogo o solicita los recursos, para quien la ubicación de los datos es totalmente invisible y servidor al proceso encargado de responder a las solicitudes realizadas por el cliente.

Es reconocida como una arquitectura de procesamiento cooperativo que relaciona procesos que corren en máquinas separadas, donde uno se encarga de proveer los servicios que el otro necesita. Se reconoce como una de sus principales ventajas la organización y centralización de la gestión de la información, además de la separación de responsabilidades, que permite realizar de manera más fácil y clara el diseño del sistema.

Según IBM[3], "es la tecnología que proporciona al usuario final el acceso transparente a las aplicaciones, datos, servicios de cómputo o cualquier otro recurso del grupo de trabajo y/o, a través de la organización, en múltiples plataformas. El modelo soporta un medio ambiente distribuido en el cual los requerimientos de servicio hechos por estaciones de trabajo inteligentes o "clientes", resultan en un trabajo realizado por otros computadores llamados servidores".

2.6 Lenguajes de programación

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

A continuación se mencionarán algunos de los lenguajes que se deben tener en cuenta a la hora de desarrollar aplicaciones Web, dividiendo los siguientes epígrafes basados en la arquitectura cliente-servidor para lograr así un mejor entendimiento, o sea, se explicarán los lenguajes del lado del cliente y los del lado del servidor por separados.

2.6.1 Lenguajes del lado del cliente

[3] *International Business Machines* o IBM es una empresa multinacional que fabrica y comercializa herramientas, programas y servicios informáticos.

Los lenguajes del lado del cliente son aquellos que son interpretados directamente por el navegador y no requieren de un tratamiento previo. Los mismos, se encargan de garantizar el control sobre los formularios y sus principales eventos. Seguidamente se introducirán algunos de ellos, los cuales serán tomados en cuenta a la hora de desarrollar la aplicación propuesta.

2.6.1.1 Lenguaje de Marcas de Hipertexto (HTML)

El lenguaje de marcas de hipertexto (HyperText Markup Language, HTML) es, como su nombre lo indica, un lenguaje de marcado, o sea, es un lenguaje que incorpora junto con el texto, etiquetas o marcas que contienen información adicional acerca de la estructura de dicho texto. Es muy utilizado en la construcción de páginas Web. HTML se escribe en forma de etiquetas, cada una con un significado específico, y rodeadas por corchetes angulares, que permiten describir la apariencia o estructura de un documento, es decir, se encargan de indicar al navegador donde debe colocar cada texto, imagen o video y la forma que tendrán estos al ser colocados en la página.

2.6.1.2 JavaScript

JavaScript es un lenguaje de programación usado principalmente para construir páginas Web dinámicas. Es un lenguaje sencillo y pensado para trabajar con rapidez, incluso para aquellas personas con poca experiencia en la programación, aprenderlo y practicarlo resulta bastante fácil. JavaScript proporciona dinamismo a la página Web, ya que permite incorporar efectos especiales a la misma, así como interactividad con los usuarios. Teniendo en cuenta que es un lenguaje de programación interpretado, no existe la necesidad de compilar los programas para ejecutarlos, es decir, que aquellos programas escritos en este lenguaje pueden ser probados directamente desde el navegador sin precisar procesos intermedios.

2.6.2 Lenguajes del lado del servidor

Los lenguajes de programación del lado del servidor son aquellos que son reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato compresible para él. Son en su mayoría lenguajes script que el servidor se encarga de interpretar y que permiten que las páginas no se sobrecarguen de código en el lado del cliente. Son lenguajes de programación mucho más potentes que

los del lado del cliente. A continuación se hará referencia a algunos de ellos, los cuales serán tomados en cuenta a la hora de desarrollar la aplicación propuesta.

2.6.2.1 PHP

PHP es el acrónimo de *Hipertext Pre-processor*, es una tecnología de código abierto, gratuita e independiente de plataforma, diseñada específicamente con el fin de permitir a los programadores la generación dinámica de páginas Web de forma fácil y rápida. Es un leguaje de programación usado principalmente para la interpretación del lado del servidor y contiene una gran cantidad de librerías de funciones que facilitan muchísimo el desarrollo de las aplicaciones. Es un lenguaje sencillo, de sintaxis cómoda, que en su mayoría ha sido tomada de otros lenguajes de programación como C, Java y Perl, aunque por supuesto, posee características específicas de sí mismo.

Puesto que es un producto código abierto, goza de la ayuda de un gran grupo de programadores, lo que posibilita que los fallos de funcionamiento sean encontrados y reparados rápidamente. Su código está expuesto continuamente a mejoras y extensiones del lenguaje con el fin de ampliar sus capacidades.

2.6.2.2 Java

Java es un lenguaje de programación orientado a objetos, desarrollado a principios de la década de 1990 por la compañía Sun Microsystems con el objetivo de crear un lenguaje que funcionara en redes computacionales heterogéneas, o lo que es lo mismo, redes formadas por más de un tipo de computadora. Muchas de sus sintaxis son tomadas de los lenguajes C y C++, pero posee un modelo de objetos mucho más sencillo y elimina herramientas de bajo nivel, que pueden provocar varios errores, como por ejemplo la manipulación directa de punteros.

Cuenta con varias características, de las que se destacan:

- Es independiente de plataforma.
- Es orientado a objetos, por lo que permite el encapsulamiento, la herencia, y el polimorfismo.
- Tiene altas prestaciones.
- Es multitarea: incorpora el uso de diferentes hilos de ejecución.

- Es un lenguaje interpretado, lo que provoca que sea independiente de la arquitectura en la que se vaya a ejecutar.

2.6.3 Selección del lenguaje del lado del servidor

Teniendo como propósito desarrollar una aplicación eficiente, se escoge PHP, teniendo en cuenta que este lenguaje de programación contiene características que lo convierten en una herramienta que posee una mezcla entre rendimiento y flexibilidad. Se tuvo en cuenta además que es una tecnología libre, por lo tanto no hay que pagar licencias, ni su distribución se encuentra limitada. Es un lenguaje que goza de una amplia documentación, lo que posibilita su fácil aprendizaje, agregando a esto que es sencillo, de sintaxis rápida, que posee una biblioteca que crece a medida que sus nuevas versiones van surgiendo. Además, fue diseñado principalmente para suscitar la rápida generación de páginas Web. Las tecnologías que se analicen posteriormente estarán basadas en el uso de este lenguaje de programación.

2.7 Frameworks

Un framework, en términos de desarrollo de software, constituye una estructura conceptual y tecnológica de soporte definida, que incluye artefactos de software concretos, y permite el desarrollo organizado de un proyecto de software. No obstante, fuera de las aplicaciones informáticas, pueden considerarse como un conjunto de procesos que facilitan la resolución de problemas complejos.

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

2.7.1 Frameworks PHP

Los frameworks PHP proponen estructuras básicas mediante las cuales se pueden desarrollar aplicaciones Web escritas en PHP de una forma más dinámica. Es decir, los frameworks promueven el rápido desarrollo de aplicaciones, a través de la reutilización de código y la disminución de cantidad de código repetitivo para los desarrolladores.

Los frameworks juegan también un importante papel a la hora de desarrollar aplicaciones estables, asegurando la interacción apropiada con la base de datos y la codificación en la presentación del diseño. Los mismos resultan muy útiles cuando se lleva a cabo el mantenimiento del sitio, debido a la correcta organización con la que se desarrollan este tipo de aplicaciones. Estos frameworks se basan en el patrón arquitectónico Modelo Vista Controlador (*Model View Controller*, MVC), el cual separa el proceso de desarrollo de una aplicación, permitiendo con ello trabajar sobre elementos individuales sin afectar los otros.

2.7.1.1 Zend Frameworks

Zend Framework es un framework para el desarrollo de aplicaciones y servicios Web con PHP, resulta muy útil a la hora de construir sitios Web modernos, robustos y seguros. Es un framework Open Source[4] desarrollado por Zend, la empresa que respalda comercialmente a PHP.

Sus principales características son **(Leopoldo, 2007)**:

- Trabaja con MVC.
- Cuenta con módulos para manejar archivos PDF y canales RSS.
- El Marco de Zend incluye objetos de las diferentes bases de datos, lo que provoca que las consultas a la base de datos sean bastante sencillas, al punto de no tener que escribir ninguna consulta SQL.

2.7.1.2 Symfony

Symfony es uno de los frameworks de PHP más populares. Está compuesto por un grupo de características que tributan a la optimización del desarrollo de aplicaciones Web. Symfony es un framework estable, productivo, elegante y bien documentado, que separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación Web. Está compuesto por un conjunto de herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación Web compleja. Además, se encarga de automatizar las tareas más comunes, lo que permite al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación.

[4] En español código abierto, es el término con el que se conoce al software distribuido y desarrollado libremente.

Symfony está compuesto por las siguientes características **(Potencier, y otros, 2008)**:

- Permite una fácil instalación y configuración en la mayoría de plataformas.
- Es independiente del sistema gestor de bases de datos.
- Trabaja con MVC.
- Aunque es fácil de usar en la mayoría de casos, resulta lo suficientemente flexible como para adaptarse a casos complejos.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la Web.
- Código fácil de leer y que permite un mantenimiento muy sencillo.
- Utiliza programación orientada a objetos, de ahí la necesidad del uso de PHP5.

2.7.2 Selección del Framework

Como framework a utilizar para el desarrollo de la aplicación propuesta se seleccionó Symfony, ya que está compuesto por un enorme conjunto de herramientas y utilidades que simplifican el desarrollo de las aplicaciones Web que utilicen el lenguaje PHP. Está desarrollado en PHP 5 y se puede utilizar en múltiples plataformas, así como es compatible con la mayoría de las herramientas gestoras de Base de Datos. Symfony está bien documentado, es un proyecto de software libre exitoso que cuenta con el apoyo de una gran comunidad de usuarios, razones por las cuales constituye un framework adecuado para desarrollar aplicaciones Web usando PHP.

2.8 Sistemas de Gestión de Bases de Datos (SGBD)

Un Sistema Gestor de Bases de Datos (*Database Management System*, SGBD) es un tipo de software muy específico, que permite almacenar información y posteriormente acceder de forma rápida y estructurada a la misma, asegurando su integridad, confidencialidad y seguridad. Está encaminado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

Un SGBD debe permitir:

- Definir una base de datos: especificar tipos, estructuras y restricciones de datos.
- Construir la base de datos: guardar los datos en algún medio controlado por el mismo SGBD.
- Manipular la base de datos: realizar consultas, actualizaciones, generar informes.

2.8.1 MySQL

MySQL es un SGBD capaz de almacenar una enorme cantidad de información de gran variedad. Es relacional, por lo que almacena los datos en tablas separadas en vez de recopilarlos todos en un mismo sitio, lo que le adhiere mayor velocidad y flexibilidad. Proporciona un servidor BD SQL bastante rápido, multihilo, lo que le permite realizar varias tareas concurrentemente, y multiusuario, que hace posible satisfacer de forma simultánea las necesidades de varios usuarios que comparten los mismos recursos. Existen infinidad de librerías y varias herramientas que hacen posible su uso a través de un gran número de lenguajes de programación, tiene fácil instalación y configuración por lo que actualmente goza de gran aceptación.

Algunas de sus características son **(Paez, y otros, 2008)**:

- Utiliza el lenguaje de consultas SQL.
- Posibilita el acceso a la Base de Datos (BD) de forma simultánea por varios usuarios y/o aplicaciones.
- Permite manipular grandes BD.
- Carece de un panel de control gráfico y detallado.
- Facilita conexiones entre máquinas con distintos sistemas operativos.
- No es intuitivo.

2.8.2 PostgreSQL

PostgreSQL es un SGBD relacional orientado a objetos y libre, basado en el proyecto POSTGRES, de la universidad de Berkeley. Es una derivación libre de este proyecto, que utiliza el lenguaje SQL. Al igual que otro grupo de proyectos de código abierto, su desarrollo no es manejado por una sola empresa, sino por una comunidad de desarrolladores y organizaciones comerciales que trabajan en su desarrollo. PostgresSQL utiliza Control de Concurrencia Multi-Versión con el fin de evitar bloqueos innecesarios, lo que le permite obtener una mejor respuesta en ambientes de grandes volúmenes. Cuenta además con una Interfaz de Programación de Aplicaciones (API) bastante flexible, que le permite facilitar soporte para su desarrollo en varios lenguajes de programación, tales como: Object Pascal, Python, Perl, PHP, Java/JDBC, C/C++, entre otros.

Algunas de las características de PostgreSQL son **(Quiñones, 2005)**:

- Es altamente adaptable a las necesidades del cliente.
- Corre en casi todos los principales sistemas operativos: Linux, Unix, BSDs, Mac OS, Beos, Windows.
- Documentación muy bien organizada, pública y libre, con comentarios de los propios usuarios.
- Añade una estructura de datos array.
- Incorpora diversas funciones, tales como: manejo de fechas, geométricas, orientadas a operaciones con redes, entre otras.
- Posibilita la declaración de funciones propias, así como la definición de disparadores.
- Soporta el uso de índices, reglas y vistas.
- Contiene herencia entre tablas (no entre objetos, debido a que no existen), por tanto se considera un gestor de base de datos objeto-relacional.

2.8.3 Selección del Sistema Gestor de Base de Datos

Entre los SGBD se determinó que PostgreSQL era el más adecuado para el desarrollo de la aplicación propuesta, debido a que no está sujeto a las restricciones que pudiera estar MySQL, el cual es en parte privatizado. El mismo posee características de código abierto ya que está bajo licencia BSD[5] y constituye actualmente uno de los SGBD libres más avanzados. Es una herramienta multiplataforma disponible para varios sistemas operativos y está orientado a objetos, lo que le permite el uso de herencia y polimorfismo.

2.9 Entorno de Desarrollo Integrado (IDE)

Un Entorno de Desarrollo Integrado (*Integrated Development Environment*, IDE) es una aplicación informática compuesta por un conjunto de herramientas de programación que se encargan de facilitar el trabajo de los desarrolladores de software a la hora de llevar a cabo una aplicación. Un IDE permite dedicarse a un único lenguaje de programación o hacer uso de varios de ellos.

Generalmente están compuestos por:

[5] BSD es una licencia de distribución de Software que permite el uso del código fuente en software no libre.

- Un editor de código.
- Un compilador.
- Un intérprete.
- Un depurador.
- Y un constructor de Interfaces Gráficas de Usuario (GUI).

2.9.1 Zend Studio para Eclipse

Zend Studio para Eclipse es un IDE orientado a desarrollar aplicaciones Web, que utiliza PHP como lenguaje de programación. Con este, los desarrolladores combinaron las tecnologías de Zend y las de Eclipse en busca de materializar un potente IDE totalmente Open Source. Contiene una lista de ayudas que van desde la creación y gestión de proyectos hasta la depuración de código, teniendo en cuenta que esta última constituye una herramienta de gran interés debido a que permite ejecutar páginas y conocer el contenido de las variables de la aplicación y del entorno en todo momento. Este IDE divide sus funcionalidades en dos partes, la parte del cliente y la del servidor, las cuales se instalan por separado.

Entre sus principales características se desatacan **(Alameda, 2008)**:

- Provee soporte para PHP 4 y PHP 5.
- Contiene resaltado de sintaxis, autocompletado de código, ayuda de código, así como una lista de parámetros de funciones y métodos de clase.
- Permite la inserción automática de paréntesis, corchetes de cierre y sangrado automático entre otras ayudas.
- Detecta errores de sintaxis en tiempo real.
- Proporciona soporte para la navegación en bases de datos y ejecución de consultas SQL.

2.9.2 Eclipse

Eclipse es un IDE de código abierto multiplataforma, desarrollado originalmente por IBM, aunque pasó luego a manos de la Fundación Eclipse, una organización independiente que fomenta la comunidad del código abierto. Actualmente existen otras herramientas similares, pero esta es reconocida por la calidad y facilidad que brinda a sus usuarios. Una de sus principales ventajas radica en la gran comunidad de

usuarios que extiende constantemente su código fuente, creando así funcionalidades nuevas que hacen posible el desarrollo en diversos lenguajes de programación, no solo Web, sino también de escritorio.

Entre sus características principales se encuentran **(Bermejo, y otros, 2007)**:

- Dispone de un editor de texto con resaltado de sintaxis.
- La compilación es en tiempo real.
- Realiza pruebas unitarias.
- Mediante de su arquitectura de plug-ins permite añadir soporte para lenguajes adicionales, así como otras aplicaciones, tal es el caso de herramientas UML.

2.9.3 NetBeans

NetBeans es un proyecto de código abierto escrito en Java, diseñado para el desarrollo de aplicaciones portables entre distintas plataformas. Es utilizado por los programadores para escribir, compilar, depurar y ejecutar programas. Su aprendizaje se vuelve cada vez más rápido, sobre todo para aquellos que gustan de desarrollar aplicaciones multiplataforma. NetBeans permite desarrollar aplicaciones mediante módulos formados por un archivo de Java que les permite interactuar con las APIs del NetBeans. Las aplicaciones que hayan sido creadas a partir de estos módulos pueden ser extendidas añadiendo módulos nuevos.

Dentro de sus características se destacan **(Cerdas, 2009)**:

- Dispone de soporte para crear interfaces gráficas de forma visual, sin importar el sistema operativo donde se instale.
- Permite crear aplicaciones Web con PHP5.
- Presenta soporte para el framework Symfony
- Aunque está escrito en Java, sirve para otros lenguajes de programación.

2.9.4 Selección del Entorno de Desarrollo Integrado

El entorno integrado de desarrollo seleccionado es el Zend Studio para Eclipse ya que es un programa orientado a desarrollar páginas Web que cuenta con un editor de texto para páginas PHP. Este IDE contiene una ayuda contextual con todas las librerías de funciones del lenguaje que asiste y ofrece en

todo momento los nombres de las funciones y parámetros que se deben recibir. Además, esta ayuda contextual no solo comprende funciones definidas en el lenguaje, sino que también reporta ayudas con las funciones que el programador vaya creando. Además, Zend Studio para Eclipse cuenta la vibrante comunidad de código abierto de Eclipse.

2.10 Servidores Web

Un servidor Web es un programa que implementa el protocolo HTTP[6] con el propósito de transferir hipertextos, páginas Web o páginas HTML, figuras, formularios, entre otro conjunto de objetos y animaciones. Este se mantiene atento a las peticiones realizadas por el cliente, conocido como navegador Web y las responde adecuadamente. En dependencia del tipo de petición, el servidor Web busca una página Web o bien ejecuta un programa en el servidor. De cualquier manera, siempre devuelve algún tipo de resultado HTML al cliente o navegador que realizó la petición.

2.10.1 Servidor Apache

El Apache HTTP Server, es uno de los servidores más utilizados y populares actualmente, no solo por su rapidez y robustez, sino también por ser multiplataforma y además una tecnología libre de código abierto.

Su desarrollo se inició en el 1995, basándose únicamente en código de NCSA HTTPd[7], aunque más tarde fue reescrito completamente. Es un software estructurado en módulos, que gracias a este diseño modular altamente configurable, permite ampliar sus capacidades incorporándosele módulos nuevos, lo que le ha permitido añadir nuevas extensiones entre las que se destaca PHP. A pesar de ser criticado por falta de interfaz gráfica que ayude con su configuración, se reconoce como uno de los servidores más flexibles, estables y eficientes.

2.10.2 Microsoft Internet Information Services (IIS)

[6] HTTP o *HyperText Transfer Protocol*, es un protocolo de transferencia de hipertexto usado en cada transacción de la Web.

[7] El NCSA HTTPd fue un Servidor Web desarrollado inicialmente por Robert McCool y una lista de colaboradores. Su desarrollo fue suspendido en 1998, pero su código sobrevivió por un tiempo en manos del Proyecto Apache. Prácticamente todo el código de NCSA se ha ido reescribiendo progresivamente en versiones Apache.

Internet Information Services es un potente servidor Web, y aunque ofrece una estructura de gran fiabilidad, capacidad de manejo y escalabilidad para las aplicaciones Web, es un software propietario y sus servicios se limitan únicamente a los ordenadores que trabajan con Windows. Estos servicios proporcionan las funciones y herramientas que se necesitan a la hora de administrar de manera sencilla un servidor Web seguro. IIS se basa en varios módulos, los que le permiten procesar distintos tipos de páginas. Actualmente se conoce como el segundo servidor Web más utilizado en el mundo, detrás del anteriormente mencionado servidor Apache.

2.10.3 Selección del servidor Web

Como servidor Web para el desarrollo de la aplicación propuesta se seleccionó el Apache, producto a que el mismo es reconocido por su escalabilidad, seguridad, y rendimiento, y se considera el servidor más utilizado en la actualidad a nivel mundial. Apache, a diferencia de IIS, es multiplataforma, por lo que está disponible para varios sistemas operativos. Es un servidor Web HTTP de código abierto que se adecua al lenguaje de programación que fue seleccionado para el desarrollo de la aplicación.

2.11 Conclusiones

Después hacer un breve análisis de las tendencias y tecnologías actuales se puede decir al respecto que en su mayoría todas podrían ser consideradas como herramientas factibles para el desarrollo de un producto de software, siempre y cuando se adecuen a las necesidades del proyecto que se desarrolla. Teniendo en cuenta que la universidad, y el país en general persigue fomentar el uso de herramientas libres, la investigación siempre estuvo orientada a la selección de las mismas, teniendo en cuenta que se ajustaran a la aplicación que se persigue implementar.

Por tales razones se estipuló como metodología de desarrollo la AUP, con el uso de Visual Paradigm como herramienta CASE. El lenguaje de programación seleccionado fue PHP, incluyendo a Symfony como el framework más apropiado para dicho lenguaje. El SGBD escogido fue el PostgreSQL, y como IDE a utilizar se designó el Zend Studio para Eclipse. Por último se eligió como servidor Web para el desarrollo de la aplicación al Apache. Con la elección de cada una de las herramientas anteriormente mencionadas, se persigue desarrollar un producto de alta calidad, que logre satisfacer las necesidades y expectativas de sus clientes.

CAPITULO 3: Presentación de la solución propuesta

3.1 Introducción

A continuación se hará una descripción de la solución que se propone para lograr un sistema que permita automatizar la gestión de documentos que se generan de las evaluaciones a las arquitecturas de software que se llevan a cabo en los distintos proyectos de la UCI. Para ello se hará un modelado del proceso, que abarcará la especificación de actores y trabajadores involucrados, casos de uso, requisitos funcionales y no funcionales que debe cumplir el sistema, así como los diagramas de casos de uso tanto del negocio como del sistema. Se tratarán además aspectos importantes del diseño como son: los patrones utilizados, diagramas de clases del diseño, de clases persistente y el modelo de datos. En fin, se mostrarán un conjunto de artefactos que posibiliten el entendimiento de la solución que se propone.

3.2 Descripción del proceso a automatizar

En los proyectos productivos de la UCI se pretende implementar una nueva forma de evaluación de Arquitectura de Software, guiada por un proceso evaluativo que se basa en una metodología de evaluación bastante joven. La misma cuenta con dos áreas de procesos fundamentales, el de evaluación y la de documentación. Su desarrollo está dado a través de fases, en las que se desarrollan un conjunto de actividades de las que se generan planillas donde se describe detalladamente aspectos importantes en el desarrollo de la misma.

Durante este proceso evaluativo intervienen un grupo de personas que se encargan de desarrollar exitosamente el mismo. El equipo de desarrollo está compuesto por el líder de proyecto, arquitectos, analista, equipo de evaluación y clientes. Se tiene en cuenta que no participan en su totalidad todas los personas relacionadas con los roles mencionados, sino una representación de los mismos.

El proceso evaluativo se inicia cuando el jefe de arquitectura de un departamento le indica al grupo de arquitectos de un proyecto determinado que es conveniente llevar a cabo una evaluación a la arquitectura de dicho proyecto y muestra las razones de esta decisión. Es entonces que los arquitectos del proyecto se encargan de generar un documento donde se detalle la fase de desarrollo del proyecto, así como los motivos por los que se lleva a cabo la misma. A esto se le denomina Fase de Concepción, y es donde el

líder de proyecto, analista, arquitectos y clientes se encargan de llevar a cabo todas las actividades de organización, planificación y preparación que aseguran el éxito de las restantes fases. De cada una de las actividades que se desarrollen se generarán documentos que dan fe del cumplimiento de las mismas.

El segundo paso es la Fase de Presentación, donde se expone toda la información necesaria en el proceso evaluativo. Los evaluadores que se seleccionaron en la fase anterior a esta se encargarán de describir todo el proceso de evaluación, así como los clientes expondrán las metas del negocio. La propuesta de arquitectura será mostrada de manos de los arquitectos.

La tercera fase de desarrollo del proceso evaluativo se denomina Fase de Desarrollo, y constituye el momento más importante en todo el proceso ya que es aquí donde se evalúa propiamente la arquitectura presentada. Los arquitectos, evaluadores, analista y clientes trabajan en la identificación, descripción y prioridad de los atributos de calidad y escenarios de la arquitectura evaluada, dejando plasmados los mismos en las planillas que se generan de cada actividad.

El momento final de la evaluación se conoce por el nombre de Fase de Exposición de los Resultados. En la misma el grupo de arquitectura se encarga de identificar los cambios arquitectónicos necesarios según el resultado de la evaluación y paralelamente a esto los evaluadores emiten un documento final del proceso realizado, donde incluyen una evaluación de Mal (M), Regular (R), Bien (B) o Excelente (E) en dependencia del nivel de cumplimiento de los atributos de calidad y la cantidad de riesgos arquitectónicos encontrados.

Con el objetivo de automatizar este proceso evaluativo se desarrolla un Sistema para la Gestión de la Evaluación de la Arquitectura (SGEVARQ), con el propósito de facilitar el acceso a la información que se genera durante el desarrollo del mismo.

3.3 Modelado

Durante el desarrollo de esta disciplina se persigue comprender el negocio de la organización, así como el dominio del problema que aborda el proyecto con el objetivo de encontrar una solución factible al mismo. Resulta muy favorable para el desarrollo de esta disciplina la participación activa de los clientes

conjuntamente con el equipo de desarrollo. En AUP, el modelado comprende lo que serían en RUP el modelo de negocio, los requerimientos y el análisis y diseño.

3.3.1 Actores del negocio

Un actor del negocio es cualquier individuo, grupo, organización o máquina que interactúa con el negocio. El término actor significa el rol que algo o alguien juega cuando interactúa con el negocio. Este permanece siempre fuera de las fronteras del negocio y debe dársele un nombre que describa su rol dentro del mismo.

Actor	Descripción
J` Arquitectura del Dpto.	Es la persona encargada de indicarle a un proyecto determinado que se hace necesario realizar una evaluación a la arquitectura del mismo.

Tabla 1. Actores del Negocio.

3.3.2 Trabajadores del negocio

Un trabajador del negocio representa una persona (o grupo de personas), una máquina o un sistema automatizado dentro del negocio, los mismos son los que realizan las actividades que comprende un caso de uso, interactuando con otros trabajadores del negocio y manipulando entidades del mismo. Un trabajador está dentro de la frontera del negocio y es quien se convertirá en un futuro en usuario del sistema que debe construir. Cada trabajador se identifica dentro del negocio a través de un rol determinado.

Luego de realizar el análisis correspondiente se identificaron los siguientes trabajadores del negocio:

Trabajador	Descripción
Grupo arquitectura	Es el encargado de emitir el documento con que se inicia el proceso evaluativo, donde se recoge la fase de desarrollo del proyecto y las razones de la evaluación.

Líder de proyecto	Le corresponde elaborar el cronograma del proceso evaluativo para lograr organización y coherencia en las evaluaciones y permitir el control del cumplimiento de las mismas.

Tabla 2. Trabajadores del Negocio.

3.3.3 Diagramas de casos de uso del negocio

Un diagrama de casos de uso del negocio se encarga de describir la interacción que existe entre el actor del negocio y el caso de uso asociado a este. El diagrama que se muestra a continuación está compuesto por el actor del negocio (J` Arquitectura del Dpto.) que es quien inicia el proceso del negocio. Este tipo de diagramas posibilita tener una visión general de los diferentes procesos del negocio expresados en casos de uso, permiten además mostrar las fronteras y el entorno de la organización.

Figura 2. Diagrama de Casos de Uso del Negocio.

3.3.4 Descripción textual de los Casos de Uso del Negocio

El proceso del negocio define un conjunto de tareas que se relacionan de forma lógica y que siguen una secuencia de pasos coherentes empleando recursos de la organización para alcanzar los objetivos propuestos. Un caso de uso del negocio representa un determinado proceso del negocio, por lo que define un conjunto de acciones que producen un resultado observable para ciertos actores del negocio. Desde el punto de vista de un actor individual, puede verse como un flujo de trabajo completo que produce resultados deseables.

El objetivo principal de describir los casos de uso del negocio es representar su flujo de sucesos detalladamente, incluyendo como comienza, termina e interactúa con los actores y trabajadores del negocio.

Caso de Uso:	Indicar Evaluación
Actores:	J` Arquitectura del Dpto.
Trabajadores:	Grupo arquitectura, Líder de proyecto.
Resumen:	El actor indica al proyecto que debe realizar una evaluación a la arquitectura del software. El grupo de arquitectura se encarga de elaborar el documento que define la evaluación. El líder del proyecto revisa el documento hecho por el grupo de arquitectura y elabora el cronograma de evaluación.
Precondiciones:	---

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Negocio**
1. El J` de arquitectura del Dpto. le comunica al proyecto que es necesario realizar una evaluación a la arquitectura establecida.	2. Solicita motivo de la evaluación.
3. Describe los motivos de la evaluación.	4. Elabora el documento donde se define la Evaluación de la Arquitectura. 5. Revisa el documento. 6. Elabora el cronograma de la evaluación.

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	5.1 Si el documento no ha sido redactado correctamente ir al punto 4.

Pos condiciones	---

Tabla 3. Descripción del CUN "Indicar Evaluación."

3.3.5 Descripción de los requisitos del software

La gestión de requisitos constituye actualmente una buena práctica que favorece en gran medida el entendimiento común entre clientes y desarrolladores acerca de los requerimientos que el producto final debe cumplir. La misma contribuye considerablemente a la comprensión del problema que se necesita resolver y a las posibles maneras de solucionarlo, factores que resultan fundamentales para alcanzar el éxito de los proyectos de software. Describir los requerimientos de un software implica establecer una especificación preliminar de los requisitos funcionales y no funcionales del producto que se desarrolla.

3.3.5.1 Requisitos Funcionales

Los requisitos funcionales se definen como capacidades o condiciones que el sistema debe cumplir, o sea, son las características que el cliente solicita del sistema y espera que cumpla el mismo, en vista de solucionar su problema o conseguir su objetivo.

La aplicación que se pretende construir debe cumplir los siguientes requisitos funcionales:

RF 1. Autenticar usuario.

RF 2. Gestionar Usuario.

2.1. Adicionar Usuario.
2.2. Modificar Usuario.
2.3. Eliminar Usuario.
2.4. Listar Usuarios.

RF 3. Gestionar Evaluación.

3.1. Adicionar Evaluación.
3.2. Modificar Evaluación.
3.3. Mostrar Evaluación.
3.4. Listar Evaluación.

RF 4. Gestionar Definición de la Evaluación de Arquitectura.

4.1. Adicionar Documento de la Evaluación de la Arquitectura.

4.2. Modificar Documento de la Evaluación de la Arquitectura.

4.3. Mostrar Documento de la Evaluación de la Arquitectura.

4.4. Listar Documento de la Evaluación de la Arquitectura.

RF 5. Gestionar Cronograma de la Evaluación.

5.1. Adicionar Cronograma de la Evaluación.

5.2. Modificar Cronograma de la Evaluación.

5.3. Mostrar Cronograma de la Evaluación.

5.4. Listar Cronograma de la Evaluación.

RF 6. Gestionar Involucrados.

6.1. Adicionar Involucrados.

6.2. Modificar Involucrados.

6.3. Mostrar Involucrados.

6.4. Listar Involucrados.

RF 7. Gestionar Metas del Negocio.

7.1. Adicionar Metas del Negocio.

7.2. Modificar Metas del Negocio.

7.3. Mostrar Metas del Negocio.

7.4. Listar Metas del Negocio.

RF 8. Gestionar Descripción de la Arquitectura.

8.1. Adicionar Documento de la Arquitectura.

8.2. Modificar Documento de la Arquitectura.

8.3. Mostrar Documento de la Arquitectura.

8.4. Listar Documento de la Arquitectura.

RF 9. Gestionar Componentes de la Arquitectura.

9.1. Adicionar Componentes de la Arquitectura.

9.2. Modificar Componentes de la Arquitectura.

9.3. Listar Componentes de la Arquitectura.

RF 10. Gestionar Cronograma de Sesiones de la Evaluación.

10.1. Adicionar Cronograma de la Evaluación.

10.2. Modificar Cronograma de la Evaluación.

10.3. Mostrar Cronograma de la Evaluación.

10.4. Listar Cronograma de la Evaluación.

RF 11. Gestionar Participantes del Cronograma de Sesiones.

11.1. Adicionar Participantes del Cronograma de Sesiones.

11.2. Modificar Participantes del Cronograma de Sesiones.

11.3. Listar Participantes del Cronograma de Sesiones.

RF 12. Gestionar Valoraciones de los Objetivos del Negocio.

12.1. Adicionar Valoraciones de los Objetivos del Negocio.

12.2. Modificar Valoraciones de los Objetivos del Negocio.

12.3. Mostrar Valoraciones de los Objetivos del Negocio.

12.4. Listar Valoraciones de los Objetivos del Negocio.

RF 13. Gestionar Valoraciones de la Arquitectura Presentada.

13.1. Adicionar Valoraciones de la Arquitectura Presentada.

13.2. Modificar Valoraciones de la Arquitectura Presentada.

13.3. Mostrar Valoraciones de la Arquitectura Presentada.

13.4. Listar Valoraciones de la Arquitectura Presentada.

RF 14. Gestionar Atributos de Calidad.

14.1. Adicionar Atributos de Calidad.

14.2. Modificar Atributos de Calidad.

14.3. Mostrar Atributos de Calidad.

14.4. Listar Atributos de Calidad.

RF 15. Gestionar Comportamiento de los Atributos de Calidad.

15.1. Adicionar Comportamiento de los Atributos de Calidad.

15.2. Modificar Comportamiento de los Atributos de Calidad.

15.3. Mostrar Comportamiento de los Atributos de Calidad.

15.4. Listar Comportamiento de los Atributos de Calidad.

RF 16. Gestionar Escenarios.

16.1. Adicionar Escenarios.

16.2. Modificar Escenarios.

16.3. Mostrar Modificar Escenarios.

16.4. Listar Modificar Escenarios.

RF 17. Gestionar Comportamiento del Atributo de Calidad por Escenarios.

17.1. Adicionar Comportamiento del Atributo de Calidad por Escenarios.

17.2. Modificar Comportamiento del Atributo de Calidad por Escenarios.

17.3. Listar Comportamiento del Atributo de Calidad por Escenarios.

RF 18. Gestionar Interacción entre componentes del Escenario.

18.1. Adicionar Interacción entre componentes del Escenario**.**

18.2. Modificar Interacción entre componentes del Escenario

18.3. Listar Interacción entre componentes del Escenario

RF 19. Gestionar Puntos Críticos de la Arquitectura.

19.1. Adicionar Puntos Críticos de la Arquitectura.

19.2. Modificar Puntos Críticos de la Arquitectura.

19.3. Mostrar Puntos Críticos de la Arquitectura.

19.4. Listar Puntos Críticos de la Arquitectura.

RF 20. Gestionar Atributos Involucrados en Puntos Críticos.

20.1. Adicionar Atributos Involucrados en Puntos Críticos.
20.2. Modificar Atributos Involucrados en Puntos Críticos.
20.3. Listar Atributos Involucrados en Puntos Críticos.

RF 21. Gestionar Documento de Cambios de la Arquitectura.

21.1. Adicionar Cambios de la Arquitectura.
21.2. Modificar Cambios de la Arquitectura.
21.3. Mostrar Cambios de la Arquitectura.
21.4. Listar Cambios de la Arquitectura.

RF 22. Gestionar Atributos que Impactan en los Cambios de la Arquitectura.

22.1. Adicionar Atributos que Impactan en los Cambios de la Arquitectura.
22.2. Modificar Atributos que Impactan en los Cambios de la Arquitectura.
22.3. Listar Atributos que Impactan en los Cambios de la Arquitectura.

RF 23. Mostrar Ayuda

3.3.5.2 Requisitos no Funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe cumplir. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Después de determinar lo que el software debe hacer debe establecerse cómo el mismo ha de comportarse y cuáles serán sus cualidades.

Hardware:

Las computadoras clientes deben contar con:

- Un microprocesador con una velocidad de 3 GHz, una memoria RAM de 128 MB o superior y una tarjeta de red.

El nodo (PC) que alojará la aplicación y la Base de Datos deberá contar con:

- Un microprocesador con una velocidad de 3 GHz, una memoria RAM de 1 GB o superior, más de 100 GB de disco duro y una tarjeta de red.

Software:

Las computadoras clientes deben tener instalado:

- Windows o GNU/Linux en alguna de sus distribuciones.
- Navegador Web Mozilla Firefox con versiones superiores a la 1.5.

El nodo (PC) que alojará la aplicación y la Base de Datos deberá tener instalado:

- Windows o GNU/Linux en alguna de sus distribuciones.
- Apache 2 como servidor Web y el servidor de script PHP 5 o superior.
- PostgreSQL 8.2 y el framework Symfony en su versión 1.2.8.

Apariencia o interfaz externa:

- El sistema debe contar con una interfaz sencilla y agradable, así como mostrar el contenido de manera comprensible, con el objetivo de ser usado fácilmente por el usuario sin necesidad de mucho entrenamiento.

Disponibilidad:

- A los usuarios se les garantizará el acceso a la aplicación en cualquier horario de trabajo así como en tiempo extra, garantizándoles obtener la información, lo que posibilitará a los mismos disponer de los datos deseados en el momento que lo precisen.

Usabilidad:

- El sistema debe diferenciar las opciones y permisos para los usuarios que accedan al sistema con diferentes roles.
- La información deberá estar disponible en todo momento, limitada solamente por las restricciones de acuerdo a las políticas de seguridad definidas.

Rendimiento:

- El sistema debe responder en un tiempo relativamente corto a las peticiones hechas por el usuario.

Seguridad:

- Al sistema podrán acceder únicamente los usuarios que estén registrados en la Base de Datos.
- Los usuarios que accedan al sistema podrán trabajar solamente sobre las actividades y recursos que les permita su rol, así como los permisos que este tenga establecido.

Portabilidad:

- Debido a que el sistema propuesto será una aplicación Web implementada con PHP 5 que es un lenguaje multiplataforma, podrá ser usado bajo varios sistemas operativos, tanto libres como propietarios.

Requerimiento de Ayuda:

- El sistema debe proporcionar documentación de ayuda al usuario según el rol que va a desempeñar en el sistema, explicando cuales serán las interfaces a las que tendrá acceso y cómo interactuar con ellas.

3.3.6 Actores del Sistema

"Un actor es una idealización de una persona externa, de un proceso, o de una cosa que interactúa con un sistema, un subsistema, o una clase. Un actor caracteriza las interacciones que los usuarios exteriores pueden tener con el sistema. En tiempo de ejecución, un usuario físico puede estar limitado a los actores múltiples dentro del sistema. Diferentes usuarios pueden estar ligados al mismo actor y por lo tanto pueden representar casos múltiples de la misma definición de actor." **(Pressman, 2002)** Cada actor participa en uno o más casos de uso.

Los actores que intervienen en el sistema son:

- Líder de proyecto.
- Grupo arquitectura.

- Cliente del proyecto.
- Grupo evaluación.
- Usuario.
- Administrador.

3.3.7 Diagrama de Casos de Uso del Sistema

"Un diagrama de Casos de Uso del Sistema describe parte del modelo de casos de uso y muestra un conjunto de casos de uso y actores con una asociación entre cada par actor/caso de uso que interactúan." **(Jacobson, y otros, 2000)**

A continuación se muestra la interacción de los actores con el sistema a través del Diagrama de Caso de Uso del Sistema:

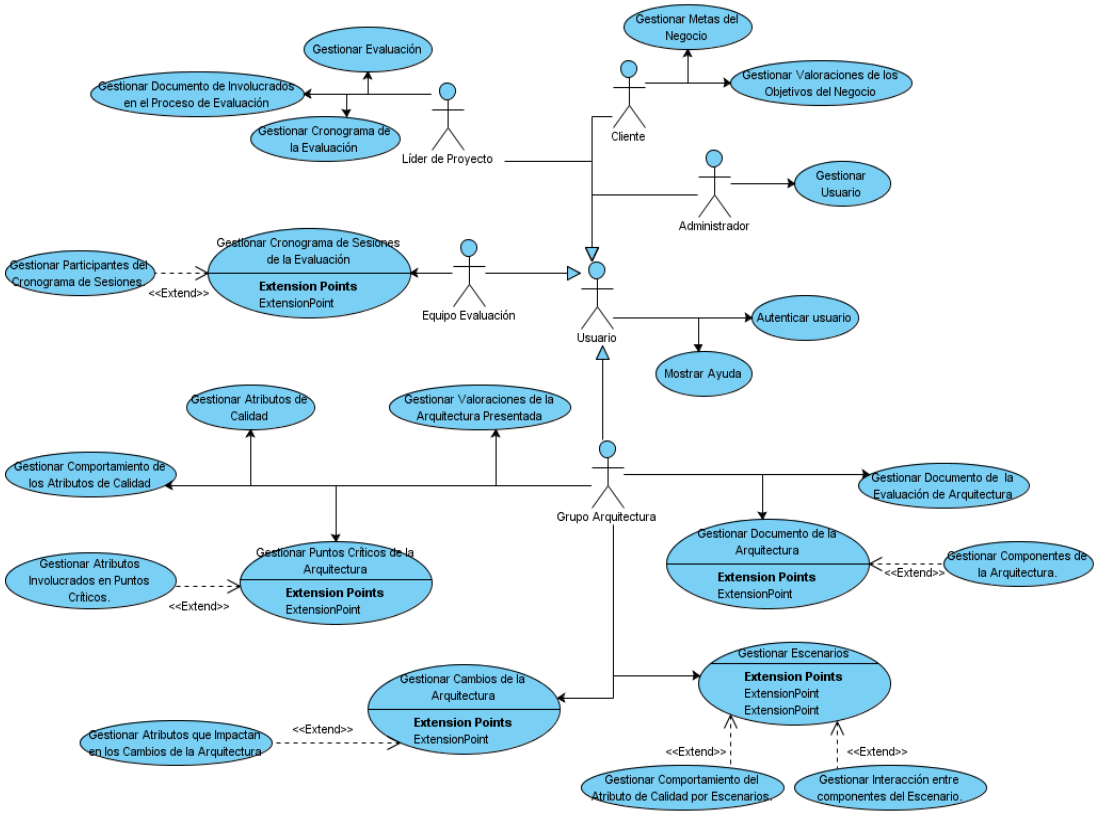

Figura 3. Diagrama de Casos se Uso del Sistema.

3.3.8 Descripción de los Casos de Uso del Sistema

Un Diagrama de Casos de Uso del Sistema representa gráficamente a los procesos y su interacción con los actores. Los mismos se definen como un fragmento de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores, cada uno de ellos debe comunicarse con al menos un actor del sistema.

Para ver la descripción detallada de cada uno de los casos de uso que componen el sistema dirigirse al **Anexo 1**.

3.3.9 Análisis del Sistema

Aún reconociendo que el modelo del análisis logra un refinamiento de los requisitos del software, ya que refleja una vista interna del sistema a través de una mejor especificación de los casos de uso y la determinación de las clases necesarias para llevar a cabo las funcionalidades del sistema, se ha decidido que no se realizará el desarrollo del mismo. Se tuvo en cuenta a la hora de tomar esta decisión que la metodología de desarrollo que guía el proceso es AUP, la cual por ser una metodología ágil busca reducir el coste en tiempo y esfuerzo, permitiendo generar los artefactos indispensables durante el proceso de desarrollo de un software.

Se destaca además que el análisis es una traza directa al diseño por lo que si se ha realizado una buena definición de los procesos, debido a que los requisitos exigidos por la aplicación resultan bastante simples, se puede pasar fácilmente al diseño del sistema sin necesidad de llevar a cabo el análisis del mismo.

3.3.10 Diseño del Sistema

La etapa de diseño de un sistema es de vital importancia ya que es aquí donde se modela dicho sistema y se encuentra su forma (la arquitectura), lo que permite dar soporte a los requisitos del mismo, tanto funcionales como no funcionales, así como a las restricciones que se le suponen. Es en esta etapa donde se crea una entrada apropiada y un punto de partida para la implementación de la aplicación.

3.3.10.1 Modelo Vista Controlador (MVC)

Como se ha mencionado anteriormente el framework seleccionado para desarrollar la aplicación fue Symfony, el mismo ofrece grandes ventajas para el diseño e implementación de la misma puesto que está basado en un patrón clásico del diseño Web conocido como arquitectura MVC, formado por tres niveles:

A continuación de describe detalladamente cómo está compuesta la aplicación que se desarrolla de acuerdo a este patrón arquitectónico.

Modelo

El modelo se divide en dos capas, la capa de acceso a datos que muestra la lógica del negocio y la capa de abstracción a la BD, la cual posibilita no tener que generar sentencias SQL y con esto facilita el trabajo a los programadores.

En las aplicaciones llevadas a cabo con el framework Symfony (como es el caso), la gestión de los datos almacenados en la BD resulta bastante sencilla ya que se realiza a través de objetos y nunca se accede a los mismos de forma directa. Por cada tabla se generan cuatro clases que se encargan en conjunto de llevar a cabo la gestión de los datos y la lógica de la aplicación.

- **NombreTabla y NombreTablaPeer:** son las responsables de la lógica del negocio.
- **BaseNombreTabla**: es la contenedora de los atributos que se definen en la tabla, así como los métodos ya implementados que posibilitan el acceso a los datos.
- **BaseNombreTablaPeer:** cuenta con un grupo de métodos estáticos que contribuyen al acceso y la lógica de los datos.

Vista

La vista está compuesta en su mayoría por plantillas PHP. En ella es donde el modelo se convierte en una página Web con la que interactúa el usuario. Sus tres partes principales son:

- **Layout (plantilla global):** contiene el código HTML que se utiliza en todas las páginas, logrando así no tener código repetido en ellas.
- **Complemento de las acciones (plantillas):** recogen los resultados de la acción y se incluyen en el cuerpo de la plantilla global generando la página Web que resulta de la petición hecha por el usuario.

- **Páginas clientes y formularios:** son las páginas finales con las que interactúa el usuario.

Controlador

Se encarga del manejo de las peticiones hechas por el usuario y realiza los cambios correspondientes en la vista y el modelo. El mismo se encuentra dividido en:

- **Acciones:** es el que obtiene los resultados del modelo y especifica las variables para la vista. Representa una actividad que llevará a cabo el sistema como respuesta a una determinada petición del usuario.
- **Controlador frontal:** consiste en el punto de entrada único de la aplicación, se encarga de la configuración y las acciones a ejecutarse.

3.3.10.2 Diagramas de Clases del Diseño

Un diagrama de clases del diseño describe gráficamente las especificaciones de las clases del software y de las interfaces en una aplicación. Normalmente contiene la siguiente información **(Larman, 2004)**:

- Clases, asociaciones y atributos.
- Interfaces, con sus operaciones y constantes.
- Métodos.
- Información sobre los tipos de los atributos.
- Navegabilidad.
- Dependencias.

Los diagramas de clases del diseño han sido construidos agrupando los casos de uso por funcionalidad. Con el desarrollo de los mismos se busca proveer una abstracción de la implementación del sistema. El paquete “Modelo” que está presente en los diagramas, contiene cada una de las tablas de la BD que están relacionadas con el caso de uso correspondiente. El mismo quedaría como se muestra a continuación:

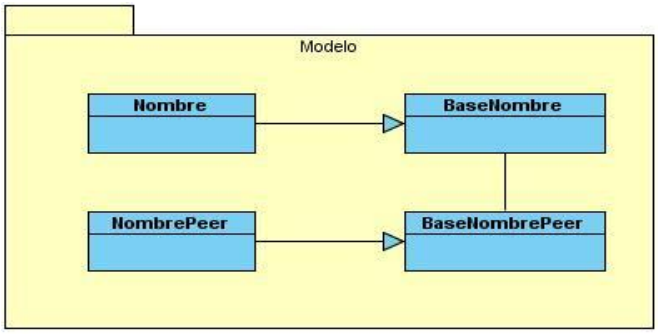

Figura 4. Modelo Genérico.

Para ver los diagramas de clases del diseño correspondientes a la aplicación dirigirse al **Anexo 2**.

3.3.10.3 Diseño de la Base de Datos

Durante esta actividad se identifican las clases del diseño que formarán parte de la BD del sistema, así como la estructura de la misma. Diseñar una BD tiene como propósito fundamental asegurarse del almacenamiento de los datos persistentes, además de definir el comportamiento que debe ser implementado en la BD.

Se reconoce que el primer paso a la hora de diseñar una BD es definir las clases persistentes, que a diferencia de las clases temporales que son manejadas y almacenadas por el sistema únicamente durante la ejecución del programa, representan el almacenamiento de los datos que persistirán más allá de la ejecución del software.

A continuación se muestra el diagrama de clases persistentes que responde a las necesidades del sistema:

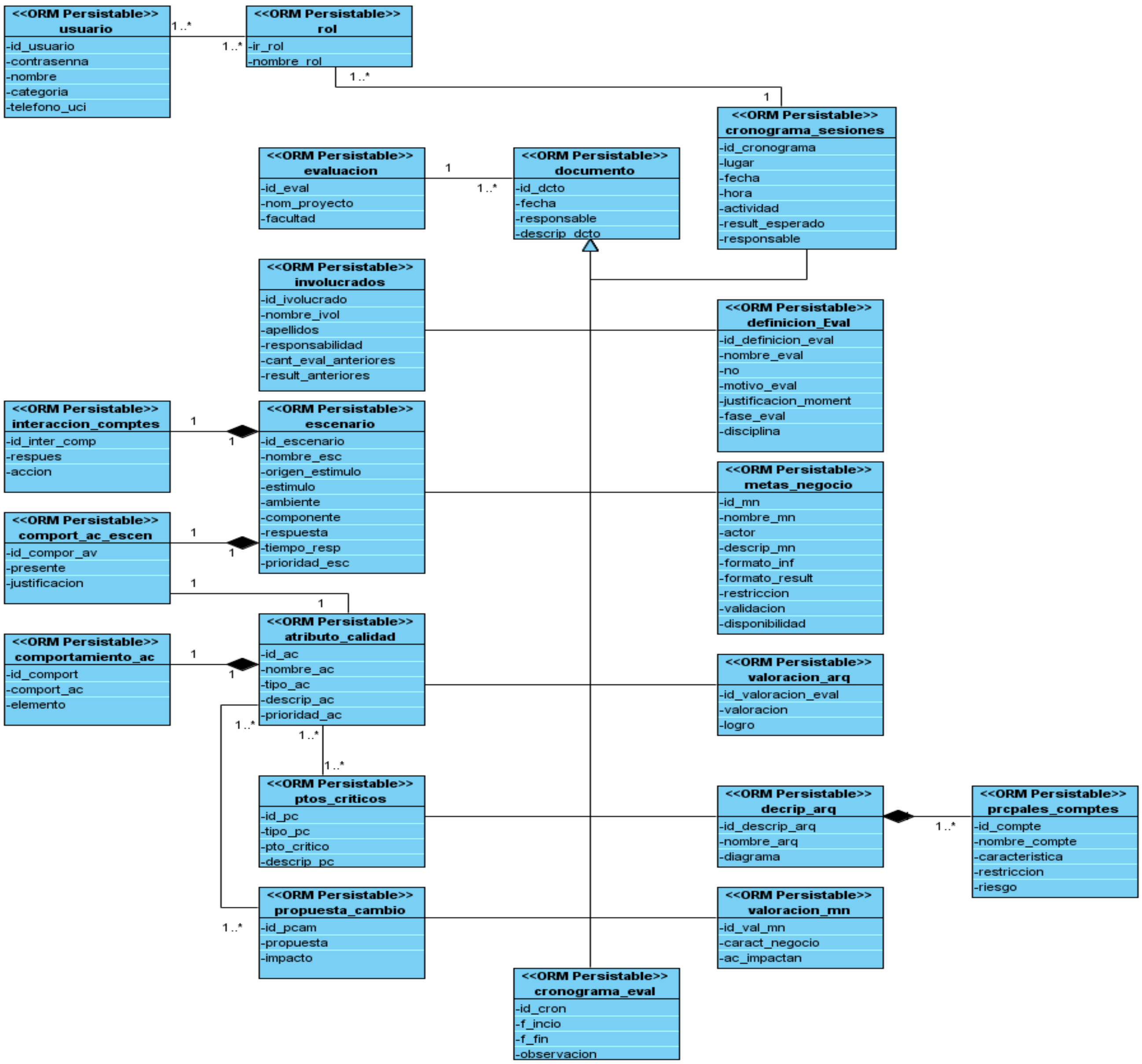

Figura 5. Diagrama de Clases Persistentes.

3.3.10.4 Patrones de diseño

"Un patrón es un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos sobre cómo aplicarlo en nuevas situaciones, o sea, un patrón es una descripción de un problema bien conocido que suele incluir: descripción, escenario de uso, solución concreta, consecuencias de utilizar el patrón, ejemplos de implementación y lista de patrones relacionados." **(Larman, 2004)**

"Dentro de los patrones de diseño existen los patrones GOF (*Gang of Four* o Pandilla de los cuatro) los cuales juegan un papel muy importante dentro de los patrones de diseño de forma general y son utilizados en innumerables ocasiones cuando se desarrollan aplicaciones informáticas. También se encuentran los patrones GRASP (Patrones de Asignación de Responsabilidades) que comunican los principios fundamentales de asignación de responsabilidades en el diseño orientado a objetos." **(Larman, 1999)**

Estos patrones se han convertido en una metodología por excelencia del diseño a la hora de desarrollar una solución, describen clases y objetos, así como la comunicación entre ellos en vista de resolver un problema general del diseño en un contexto específico, por tales razones resultaron de gran ayuda durante el diseño del sistema. A continuación se muestran cuáles han sido utilizados:

Patrones GOF utilizados:

- **Instancia única** (*Singleton*): Este patrón asegura la existencia de una instancia única para una clase y proporciona un mecanismo de acceso global a dicha instancia.

- **Observador** (*Observer*): Se emplea cuando se tiene un objeto observado y varios observadores, encargándose de notificar cualquier cambio ocurrido en el objeto observado. Es decir, define una dependencia entre el objeto observado y el conjunto de observadores, de forma tal que los cambios en el primero se vean reflejados en los otros.

- **Envoltorio** (*Decorator*): Añade nuevas responsabilidades a un objeto dinámicamente, por lo que representa una alternativa mucho más factible que la de crear subclases para extender las funcionalidades de una clase.

Se utilizaron además los muy conocidos patrones GRASP: creador, experto, controlador, alta cohesión y bajo acoplamiento.

3.4 Conclusiones

Durante el desarrollo de este capítulo se llevaron a cabo correctamente las etapas de negocio y requerimiento, ya que a través de las misma se alcanza la comprensión adecuada sobre el problema que se pretende resolver, y es aquí donde se muestran las características del negocio a raíz de las cuales se dictan los requisitos funcionales y no funcionales de la aplicación que se desarrolla, seguidos de las descripciones textuales correspondientes a cada caso de uso del sistema, lo que facilita una visión general de las funcionalidades del mismo.

Luego de haber realizado el modelado del sistema estaban creadas las condiciones para pasar al diseño del mismo, por lo que basados en el patrón MVC se desarrolló el modelo del diseño aprovechando las ventajas que brinda este patrón arquitectónico. Se determinaron además las clases persistentes a través de las cuales se obtuvo el modelo de datos a partir del cual se confeccionó la BD. Este acápite se concluye habiendo creado una base sólida que servirá de apoyo a la implementación de la aplicación informática dándole así continuidad a la solución de los problemas planteados.

CAPITULO 4: Implementación y Prueba

4.1 Introducción

En este capítulo quedarán expuestos todos los detalles referentes a la implementación del sistema, así como las pruebas que se le realicen al mismo. Se presentará el diagrama de despliegue donde se muestra la distribución física del sistema en los diferentes elementos de hardware que le darán soporte y se hará referencia además a los diagramas de componentes proporcionando una vista específica que detalle las organizaciones y dependencias lógicas entre los componentes de la aplicación que se desarrolla.

4.2 Modelo de Despliegue

"El modelo de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre nodos de cómputo. Se utiliza como entrada fundamental en las actividades de diseño e implementación debido a que la distribución del sistema tiene una influencia principal en su diseño." **(Jacobson, y otros, 2000)**

A continuación se muestra la distribución física del sistema a través del diagrama de despliegue:

Figura 6. Modelo de Despliegue.

4.3 Diagrama de Componentes

Un diagrama de componentes describe la forma en que se organizan los componentes de software, así como la dependencia lógica entre los mismos, siendo estos componentes de código fuente, binarios o

ejecutables. Los componentes representan todos los tipos de elementos software que incluye la fabricación de una aplicación informática.

Para consultar los diagramas de componentes pertenecientes cada uno de los casos de uso del sistema ver **Anexo 3.**

4.4 Pruebas de Software

Probar un sistema resulta prácticamente imprescindible si se quiere verificar la calidad del mismo. Una prueba no es más que un proceso de ejecución de un programa con el propósito de encontrar errores. Se reconoce como una buena prueba a aquella que tiene altas probabilidades de encontrar faltas que no han sido detectadas hasta el momento, ya que su objetivo fundamental es demostrar la existencia de errores, nunca la ausencia de estos. Para probar el sistema en cuestión se utilizó el método de prueba basado en caja negra.

4.4.1 Pruebas de Caja Negra

Las pruebas de caja negra o funcionales son aquellas que se desarrollan sobre la interfaz del software. Estas se enfocan únicamente en los requerimientos establecidos y la funcionalidad del sistema, resultando innecesario conocer la lógica del mismo. Se basan en verifican que las entradas se acepten adecuadamente y se produzcan los resultados esperados, demostrando así la operatividad de las funciones de dicho sistema.

Los Diseños de Casos de Prueba (DCP) se le realizaron al 50% de los casos de uso críticos de la aplicación, algunos de estos diseños se muestran en el **Anexo 4**. Luego de realizar las pruebas se detectaron en total siete no conformidades lo que indujo a que se desarrollara una segunda iteración de las mismas.

De las no conformidades detectadas se solucionaron satisfactoriamente seis de ellas, tomando la NC.7 como una recomendación ya que no afecta el buen funcionamiento del sistema. Luego de corregir los errores detectados se desarrolló una segunda iteración, donde se tuvieron en cuenta los mismos casos de uso evaluados en la primera, arrojando como resultado dos no conformidades, las cuales fueron resueltas satisfactoriamente.

El siguiente gráfico muestra los resultados obtenidos en las dos iteraciones, mostrando la diferencia entre ambas.

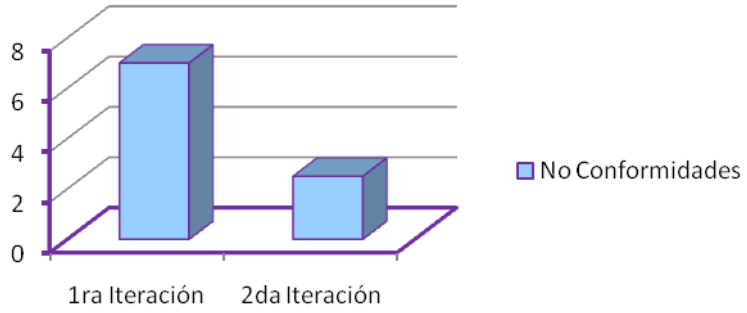

Figura 7. Comparación entre las pruebas realizadas.

4.5 Conclusiones

Con la realización de este capítulo se desarrollaron los flujos de trabajo Implementación y Prueba. Durante la etapa de implementación se crearon los artefactos correspondientes a la misma: se detalló la relación hardware y software del sistema a través del diagrama de despliegue, se describió la manera en que se organizan los componentes de software, así como la dependencia lógica entre los mismos mediante el diagrama de componentes y se logró implementar la lógica del negocio.

Se llevaron a cabo pruebas de caja negra con el objetivo de evaluar el cumplimiento de los requisitos funcionales definidos en los inicios del proceso de desarrollo, obteniendo como resultados de estas varias no conformidades que fueron resueltas satisfactoriamente, por lo que se puede afirmar que el sistema satisface las necesidades para las que fue creado.

CONCLUSIONES GENERALES

Al concluir la investigación realizada para desarrollar un sistema que automatice la gestión de los documentos que se generan durante la evaluación a la arquitectura de software se le dio cumplimiento al objetivo general planteado, así como a cada una de las tareas propuestas. Con el desempeño de dichas tareas se obtuvieron los siguientes resultados:

- Se hizo un breve estudio acerca de las distintas metodologías y métodos que se utilizan para evaluar la arquitectura del software, así como de la metodología que se pretende utilizar en los proyectos de la UCI, llegando a la conclusión que esta última reúne un grupo de características que la hacen superior a las anteriores, ya que toma los elementos acertados de los distintos métodos y técnicas de evaluación e incluye el uso de un modelo de calidad en las distintas fases de la metodología, con el objetivo de lograr la selección y refinamiento de los atributos de calidad.

- Luego de analizar varias de las tecnologías actuales se determinó que en su mayoría todas podrían considerarse herramientas factibles para el desarrollo de un software, siempre que las mismas se adecuen a las necesidades del sistema que se desarrolla, pero atendiendo que la universidad busca fomentar el uso de tecnologías libres, la selección se orientó en ese sentido, obteniendo como resultado un conjunto de herramientas que facilitaron el desarrollo de la aplicación.

- Al concluir la implementación de las distintas funcionalidades requeridas por la aplicación, se obtuvo un producto que logra dar solución al problema científico propuesto, ya que promueve la organización y centralización de los documentos que se generan durante el desarrollo de la evaluación de la arquitectura en los proyectos de la UCI y facilita el acceso a los mismos, lo que contribuye a agilizar dicho proceso.

- Luego de realizarle varias pruebas al sistema y solucionar las no conformidades detectadas, se puede concluir afirmando que la aplicación logra dar cumplimiento a los requisitos exigidos, por lo que se deduce que podrá contribuir a desarrollar evaluaciones arquitectónicas con un alto grado de eficiencia y simplicidad.

RECOMENDACIONES

Luego de concluida la presente investigación y tomando como punto de partida los resultados obtenidos, se reconoce que el objetivo general fue cumplido. Aún así se recomiendan algunos aspectos que garantizan el éxito de la utilización del sistema, tales como:

- Continuar el desarrollo de la aplicación implementando nuevas funcionalidades que optimicen el funcionamiento de la misma.
- A la hora de registrar un nuevo usuario, que los datos del mismo sean comprobados haciendo uso de los servicios habilitados en el servidor de directorio activo (LDAP) de la universidad.
- Que se difunda la existencia del sistema desarrollado para que el mismo sea utilizado por los proyectos productivos de la universidad.

BIBLIOGRAFÍA

- Pressman, Roger S. ***La Ingeniería de Software. Un enfoque práctico.*** **Madrid : Concepción Fernández Madrid, 2002.**
- Camacho, Erick, Cardeso, Fabio y Nuñez, Gabriel. ***Arquitecturas de Software.*** **Ciudad de la Habana : s.n., 2004.**
- Alvarez de Sayaz, Carlos. ***Metodología de la Investigación Científica.*** **Santiago de Cuba : Universidad de Oriente, 1995.***.*
- Hernández Leon, Rolando Alfredo y Coello Hernández, Sayda. ***El paradigma cuantitativo de la investigación científica.*** **Ciudad de La Habana : Editorial Universitaria (Eduniv), 2002.**
- Shaw, Mary. ***Larger Scale Systems Require Higher-Level Abstractions.*** **s.l. : Computer Society, 1989.**
- Bass, Len, Clements, P y Kazman, R. ***Software Architecture in practice.*** **s.l. : Adison Wesley, 2005.**
- Clements, Paul. ***A Survey of Architecture Description Languages.*** **Alemania : Proceedings of the International Workshop on Software Specification and Design, 1996.**
- Hofmeister, Christine, Nord, Robert y Soni, Dilip. ***Applied Software Architecture.*** **s.l. : Addison Wesley, 2000.**
- Salvador Gómez, Omar. ***Evaluando Arquitecturas de Software. Parte 1. Panorama General.*** **México : Brainworx S.A, 2007. 1870-0888.**
- Kazman, R, Clements, P y Klein, M. ***Evaluating Software Architectures. Methods and case studies.*** **s.l. : Adison Wesley, 2007.**
- Brey, Gustavo Andrés, y otros. ***Arquitectura de Proyectos de IT. Evaluación de Arquitecturas.*** **Buenos Aires : Universidad Tecnológica Nacional. Departamento de Sistemas, 2005.**
- Bengtsson, P. ***Design and Evaluation of Software Architecture.*** **s.l. : University of Karlskrona., 1999.**
- Kazman, R, Clements, P y Klein, M. ***Evaluating Software Architectures.*** **s.l. : Addison Wesley, 2001.**
- Bosch, J. ***Design & Use of Software Architectures.*** **s.l. : Addison-Wesley , 2000.**
- Clements, Paul, Kazman, Rick y Klein, Mark. ***Paul Clements, Rick Kazman & Mark Klein. "Evaluating Software Architectures". SEI Series in Software Engineering.*** **s.l. : Addison Wesley, 2002.**

Bibliografía

- Bosch, J. ***Design & Use of Software Architectures.*** **s.l. : Addison-Wesley, 2006.**
- Lane, T. ***Studying Software Architecture Through Design Spaces and Rules.*** **s.l. : The Computer Science Department, Carnegie Mellon University, 2006.**
- Kazman, R, Clements, Paul y Klain, M. ***Evaluating Software Architectures. Methods and case studies.*** **s.l. : 1ra Edición. Addison-Wesley Professional, 2005. 978-0201704822.**
- Kazman, R, Clements, Paul y Klein, M. ***Evaluating Software Architectures. Methods and case studies.*** **s.l. : 2da Edición. Addison Wesley, 2007.**
- Ionita Mugurel, T, Hammer Dieret, K y Obbink, Henk. ***Scenario-Based Software Architecture Evaluation Methods: An Overview.*** **Eindhoven, The Netherlands : SARA, 2006.**
- In, H, R, Kazman y Olson, D. ***From Requirements Negotiation to Software Architectural Decisions.*** **s.l. : Software Engineering Institute, Carnegie Mellon University, 2001.**
- Bengtsson, PerOlof. ***Architecture-Level Modifiability Analysis.*** **s.l. : Department of Software Engineering and Computer Science. Blekinge Institute of Technology, 2008.**
- Vigil Regalado, Yamila. ***Metodología de evaluación de arquitectura de software.*** **Ciudad de La Habana : Universidad de las Ciencias Informáticas, 2009.**
- Rondón Bolúa, Yoangel. ***Definición de los Elementos Necesarios para lograr un Amplio Nivel de Documentación y Especificación del Proceso de Evaluación de Arquitectura de Software en los Proyectos Productivos de la Universidad de las Ciencias Informáticas.*** **Ciudad de La Habana : Universidad de las Ciencias Informáticas, 2009.**
- Cueva Lovelle, Juan Manuel. ***Calidad del Software.*** **España : Universidad de Oviedo, 2007.**
- H Canós, Jose, Letelier, Patricio y Penades, María Carmen. ***Metodologías Ágiles en el Desarrollo de Software.*** **Valencia : DSIC -Universidad Politécnica de Valencia. 46022, 2004.**
- Jacobson, Ivar, Booch, Grady y Rumbaugh, James. ***El Proceso Unificado de Desarrollo de Software.*** **s.l. : Addison Wesley, 2000.**
- Billy Reynoso, Carlos. 2004. ***Introducción a la Arquitectura del Software.*** **Buenos Aires : Universidad de Buenos Aires, 2004.**
- Carrasco Puebla, Yoan, Chaviano Gómez, Enrique y Cespedes Vega, Anisleydi. 2009. **Tecnología e Internet.** ***Procedimiento para la evaluación de arquitecturas de software basadas en componentes.*** **[En línea] 07 de 05 de 2009. [Citado el: 25 de 01 de 2010.] GestioPolis.com.**

- García Arenas, María Isabel. 2006. **Turotial básico. [En línea] 16 de Mayo de 2006. [Citado el: 2 de Marzo de 2010.] http://geneura.ugr.es/~maribel/php/ .**
- Lockhart, Thomas. 2000. ***Tutorial de PostgreSQL.*** **2000.**
- W. Ambler, Scott. 2009. **Agile Unified Process (AUP) . [En línea] 2009. [Citado el: 2010 de Marzo de 5.] http://www.ambysoft.com/unifiedprocess/agileUP.html.**
- Alameda, Federico. 2008. **Teachtear.com.** ***Zend Studio for Eclipse, desarrollo profesional en PHP.*** **[En línea] 22 de Enero de 2008. [Citado el: 22 de Febrero de 2010.] http://www.techtear.com/2008/01/22/zend-studio-for-eclipse-desarrollo-profesional-en-php/.**
- Bermejo, Laura y Gómez, Enrique. 2007. ***Eclipse como IDE.Características principales, funcionalidad, utilización y caso práctico.*** **2007.**
- Calero, Manuel. 2003. ***Una explicación de la programación extrema (XP).*** **Madrid : s.n., 2003.**
- Cerdas, Felipe. 2009. ***NetBeans. El único IDE que necesitan.*** **2009.**
- De Rogatis, Natalia, y otros. 2004. ***Procesos Ágiles.*** **2004.**
- González, Rubén y Pérez, Sergio. 2007. ***Introducción al Rational Rose. Funcionalidad general.*** **Barcelona : Universidad Politécnica de Catalunya, 2007.**
- Leopoldo, Carlos. 2007. **TechTastico. Teconologia y mas que eso.** ***Zend Framework, una introduccion.*** **[En línea] 27 de Noviembre de 2007. [Citado el: 20 de Febrero de 2010.] http://techtastico.com/post/zend-framework-una-introduccion/.**
- —. 2007. **TechTastico. Teconologia y más que eso.** ***Zend Framework, una introducción.*** **[En línea] 27 de Noviembre de 2007. [Citado el: 22 de Febrero de 2010.] http://techtastico.com/post/zend-framework-una-introduccion/.**
- Lozano, Angel L. 2009. **Experiencias Vitales y tecnologias. [En línea] 6 de Enero de 2009. [Citado el: 16 de Febrero de 2010.] http://rdeheras.wordpress.com/2009/01/06/metodologias-de-desarrollo-software-por-angel-luis-lozano/.**
- Paez, Tatiana y Gómez, Pilar. 2008. **MySQL. [En línea] 4 de Septiembre de 2008. [Citado el: 22 de Febrero de 2010.] http://sistemaspyt.blogspot.com/2008/09/caractersticas-distintivas.html.**
- Potencier, Favier y Zaninotto, François. 2008. ***Symfony, la guía definitiva.*** **2008.**
- Quiñones, Ernesto. 2005. ***Introducción a posgreSQL.*** **2005.**
- Larman, Craig. 1999. ***UML y Patrones. Introducción al análisis y diseño orientado a objetos.*** **La Habana : Felix Varela, 1999.**

ANEXOS

Anexo No. 1. Descripción de los Casos de Uso del Sistema.

Caso de Uso:	Autenticar Usuario
Actores:	Usuario.
Resumen:	El caso de uso se inicia cuando el actor introduce su usuario y contraseña para acceder al sistema, los mismos se verifican contra la base de datos. Finaliza cuando se habilita la entrada al usuario con los permisos asignados a este o se deniega su acceso.
Precondiciones:	---
Referencias	RF 1
Prioridad	Crítico
Flujo Normal de Eventos	
Sección "Autenticar Usuario"	
Acción del Actor	**Respuesta del Sistema**
1. El actor inserta su nombre de usuario y contraseña en los campos correspondientes para acceder al sistema.	2. El sistema busca en la base de datos el nombre del usuario. 3. Compara la contraseña correspondiente a ese usuario. 4. Se buscan los permisos a los que tiene acceso el actor. 5. Se le permite el acceso al sistema con los permisos que tiene asignado el rol al que pertenezca.
Flujos Alternos	
Nombre o contraseña incorrectos	
Acción del Actor	**Respuesta del Sistema**

	2.1. Si el nombre del usuario introducido no existe en la base de datos o la contraseña proporcionada no coincide se muestra el mensaje “Verifique usuario o contraseña”. 2.2. Se ofrece la oportunidad de volver a introducir sus datos.
	
Poscondiciones:	Al terminar el caso de uso el usuario quedará autenticado.

Tabla 4. Descripción del CU Autenticar Usuario.

Anexo No.2. Diagramas de Clases del Diseño.

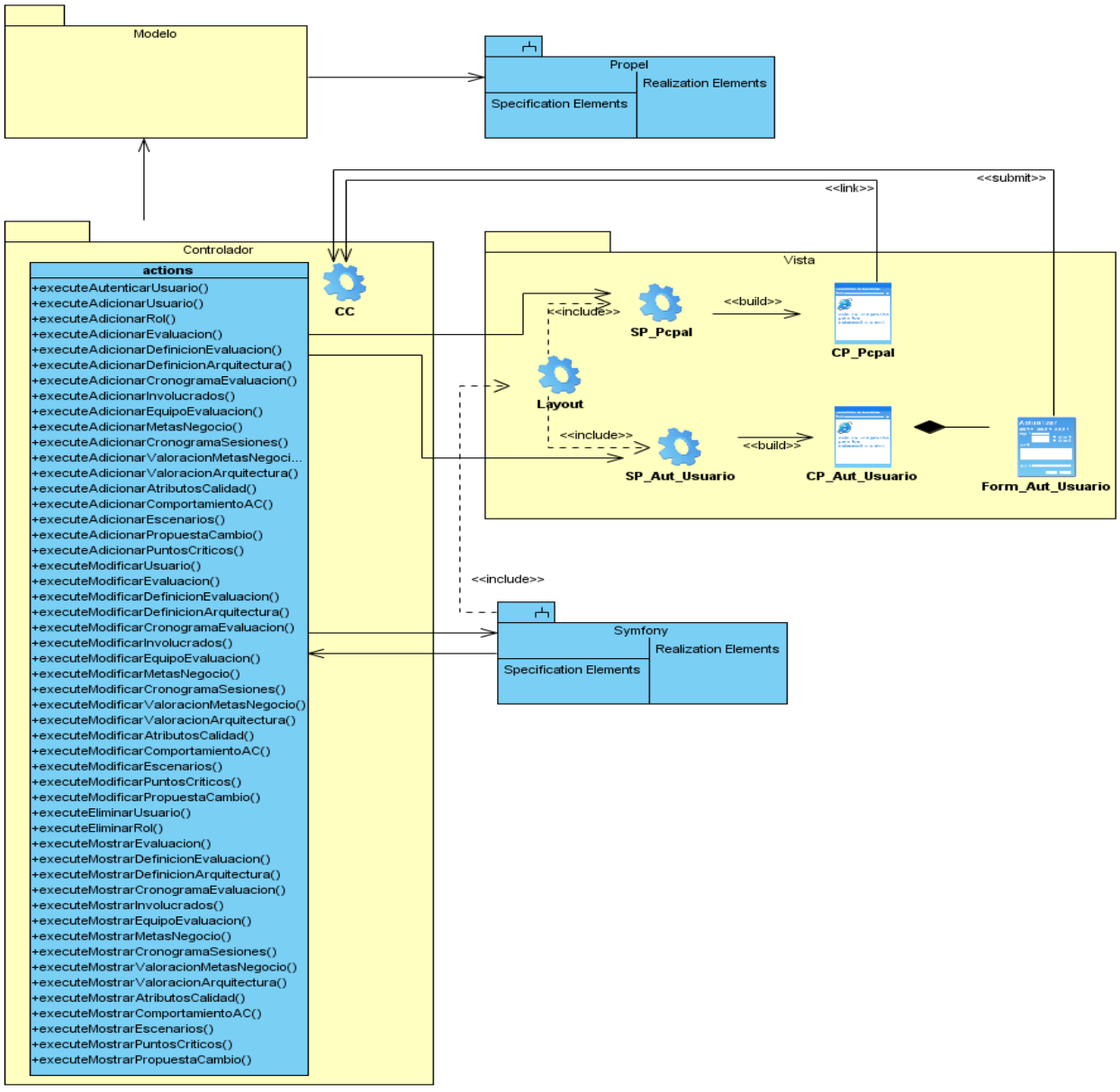

Figura 8. DCD[8] del CU Autenticar usuario.

[8] Diagrama de Clases del Diseño.

Anexo No.3. Diagramas de Componentes.

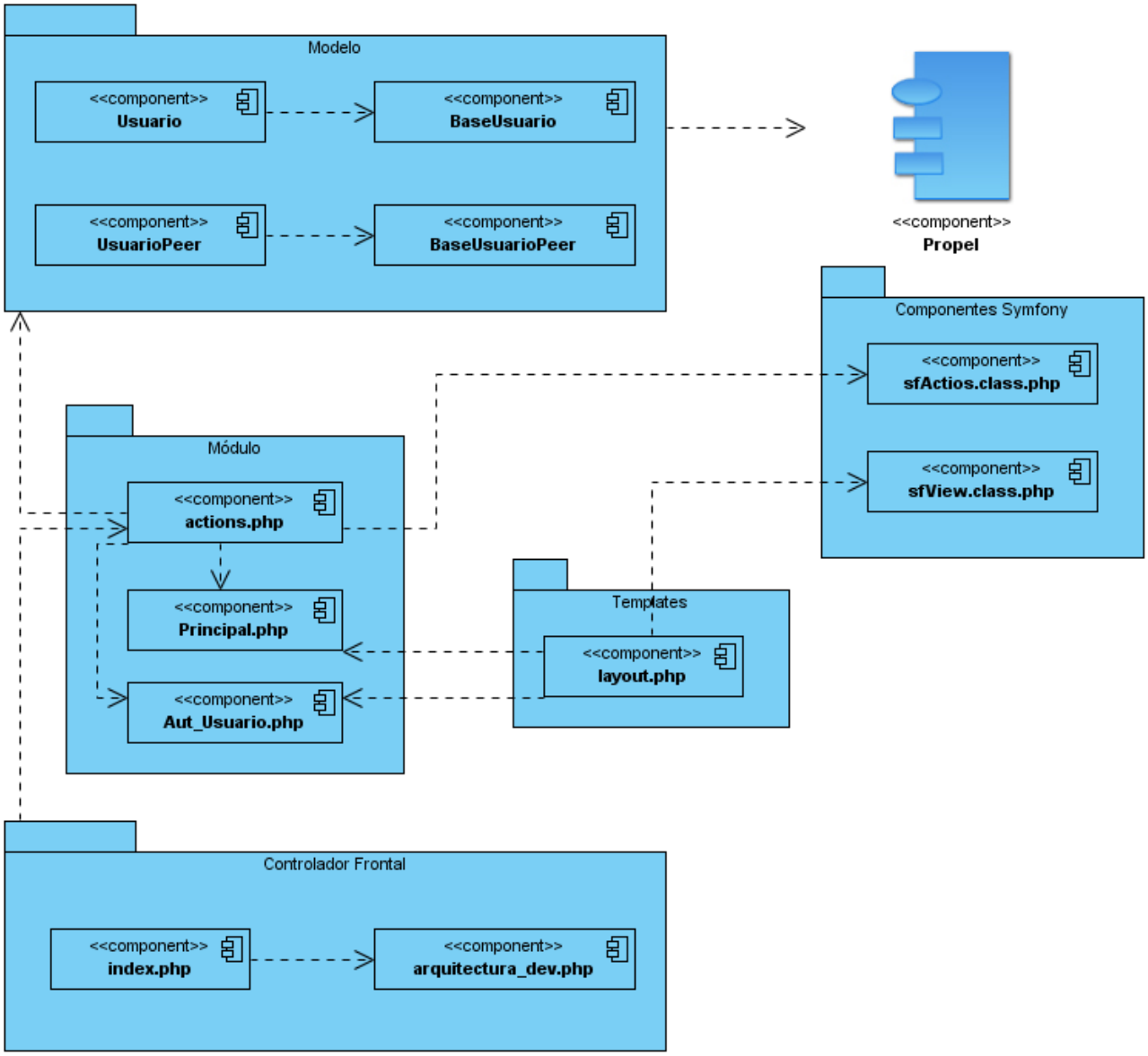

Figura 9. DC[9] del CU Autenticar Usuario.

[9] Diagrama de Componentes.

Anexo No.4. Diseños de Casos de Pruebas.

Anexo 4.1. Caso de Uso Autenticar Usuario

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: "Autenticar usuario"	EC 1.1: Autenticar usuario satisfactoriamente.	Autentica al usuario en el sistema.	El caso de uso se inicia cuando el actor introduce su usuario y contraseña para acceder al sistema, los mismos se verifican contra la base de datos y la entrada del usuario es habilitada con los permisos asignados al mismo.
	EC 1.2: Nombre o contraseña incorrectos.	El sistema muestra un mensaje indicando que los datos no son correctos.	El caso de uso se inicia cuando el actor introduce su usuario y contraseña para acceder al sistema, los mismos se verifican contra la base de datos. En caso que el nombre del usuario introducido no existe en la base de datos o la contraseña proporcionada no coincide se muestra el mensaje "Usuario o contraseña incorrecto". Se ofrece la oportunidad de volver a introducir sus datos.

Tabla 5. DCP CU Autenticar Usuario.

Id del escenario	Escenario	Usuario	Contraseña	Respuesta del Sistema	Resultado de la Prueba
EC 1.1	Autenticar usuario satisfactoriamente.	(V)Juan	(V)miclave1	La entrada del usuario es habilitada con los permisos asignados al mismo.	Satisfactorio

EC 1.2	Nombre o contraseña incorrectos	**Usuario**	**Contraseña**	**Respuesta del Sistema**	**Resultado de la Prueba**
		(V)Juan	-----	El sistema muestra un mensaje indicando que los datos no son correctos	Satisfactorio
		(I)5H4f15f	(V)miclave1	El sistema muestra un mensaje indicando que los datos no son correctos.	Satisfactorio

Tabla 6. SC 1: “Autenticar Usuario.”