Universidad de las Ciencias Informáticas

Facultad 4 y Laboratorio de Soluciones e Investigaciones Avanzadas en Gestión de Proyectos

Propuesta de solución de la Vista de Despliegue de la Arquitectura del Sistema GESPRO 12.05

Trabajo de D iploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Leannet Reyes Ruiz

Tutor: Ing. Lizardo Ramírez Tabuada

Co-tutor: Ing. Ernesto A. Mederos Franqueiro

La Habana, Junio 2012

"Año 54 de la Revolución"

Pensamiento

Pero la juventud tiene que crear. Una juventud que no crea es una anomalía realmente."

Ernesto Ché Guevara

Declaración de autoría

Por este medio declaro que soy la única autora de este trabajo y autorizo a la Universidad de las Ciencias Informáticas (UCI) para que hagan el uso que estimen pertinente con él. Para que así conste firmo la presente a los __ días del mes de __ del año __.

Leannet Reyes Ruiz

_____

Firma del autor

Ing. Lizardo Ramírez Tabuada

_____

Firma del tutor

Agradecimientos

La realización de una tesis es una aventura vital de carácter personal, pero al igual que la propia vida, no puede llegar a buen puerto, sin la ayuda, el aliento y el afecto de quienes nos rodean. Sirva esta página para agradecer a todos los que, de un modo u otro, han contribuido a la culminación de esta aventura.

A Ernesto y Félix: que sin su ayuda hubiera sido imposible la culminación de esta tesis, gracias por su paciencia, y por tantas horas de dedicación.

A mi tutor: por la ayuda brindada, y por defenderme siempre que hizo falta.

A mi grupo 8102: quien fue partícipe de mis primeros pasos universitarios, de noches enteras de desvelo, lo mismo en fiestas que en pruebas. A ellos: gracias por todo lo que compartimos.

A mis amigas de siempre: Lumey, Leydita, Maire, Macu y especialmente a Maru.

A Melvin: sobran las palabras**,** gracias por todo lo vivido.

A mis amigos de aquí: Dailen, May, Alejandro, Dayana y mi pareja favorita: Darle y Danir.

A mi novio Edelso: por todo lo que me ha ayudado este curso, por estar a mi lado aún cuando parecía que el mundo se me vendría encima, gracias por no desfallecer.

A mis tíos y primos Yaya, Suse, Tomy, Mery, Angelito, Daril, Papa y Armando: por todo el cariño y apoyo recibido durante estos 23 años.

A mis abuelos Teresita, Irma, Angel Enrique y Omar: gracias!!!, presentes o ausentes, siempre estarán en mi corazón.

A mi Marbe: gracias por ser mi segunda mamá, por estar ahí siempre, soportando mis malacrianzas con la paciencia que sólo las madres saben tener. Te quiero mucho!!!

A mi hermanita: que sé que mejorará mis pasos, que tan sólo su existencia ya es regocijo para mí, que le sirva de ejemplo y fuente de inspiración. Te amo Chrysti.

A mi papá: gracias por darme la vida, por todo el amor que has depositado en mí para ser quien soy. Hoy quiero decirte**:** "que no hay adorno más grande para un hijo que la gloria de un padre, ni para un padre que la conducta honrosa de un hijo." Espero estés orgulloso de mí. Te amo papi.

A mi mamá: la luz de mi vida, gracias por traerme a este mundo, por guiarme durante tantos años y creer en mí cuando nadie más lo hizo. Por sobreponerte a tantas vicisitudes por las que hemos atravesado juntas, para sacarme adelante. Por tu preocupación, sacrificios; por ser ejemplo de superación incasable, de comprensión y confianza. Porque has sido madre, maestra, amiga. Porque sin ti este sueño hoy no sería posible. Te amo.

Dedicatoria

A mis padres que son mi fuente de inspiración.

Resumen

La clave para lograr la eficiencia y calidad del sistema radica en la correcta definición de la arquitectura de software. Como resultado de las insuficiencias en el Modelo de Actualización y Despliegue de las herramientas de la Arquitectura del Sistema GESPRO se han visto afectados los tiempos de despliegue, configuración y actualización de las soluciones basadas en esta arquitectura. Con el desarrollo de la presente investigación, se defendió como premisa definir e implantar la Vista de Despliegue de la Arquitectura del Sistema GESPRO 12.05, de forma tal que disminuyeran los tiempos afectados. Como resultado de la investigación se lograron establecer los elementos y estrategias que forman parte de la solución para la Vista de Despliegue. La validación de la propuesta permitió demostrar el aporte práctico esperado, de optimizar el proceso de despliegue, configuración y actualización del Sistema para la gestión de proyectos GESPRO, de forma tal que se facilitara el trabajo, haciendo más factible el rendimiento y la eficacia de dicha herramienta.

Palabras Claves:

Arquitectura de software, gestión de proyectos, vista de despliegue.

Abstract

The key to achieve the efficiency and quality of the system lies in the correct definition of the software architecture. As a result of weaknesses in the Model Updating and Deployment Tools System Architecture GESPRO have been affected the time of deployment, configuration and updating of solutions based on this architecture. With the development of this research, defended the premise defining and implementing the Deployment View System Architecture GESPRO 12.05, in a way that would reduce the time involved. As a result of the investigation was able to establish the elements and strategies that are part of the solution for Deployment View. The validation of the proposal allow for demonstrating the practical contribution expected to optimize the process of deployment, configuration and updating of the System for GESPRO project management, so as to facilitate the work, making it possible performance and the effectiveness of this tool .

Key Words:

Software Arquitecture, deployment view, project management.

Índice de contenido

Introducción 10

Capítulo 1: Fundamentación Teórica 14

Introducción....14

1.1 Definiciones de la Arquitectura de Software 14

1.2 Tendencias de la Arquitectura 16

1.2.1 Arquitectura como etapa de la ingeniería de software orientada a objeto....16

1.2.2 Arquitectura estructural, basada en un modelo estático de estilos, ADLs y vistas 16

1.2.3 Estructuralismo arquitectónico radical 17

1.2.4 Arquitectura basada en patrones 17

1.2.5 Arquitectura procesual 17

1.2.6 Arquitectura basada en escenarios 18

1.3 Vistas de la Arquitectura de Software 18

1.3.1 Definiciones de Vista Arquitectónicas....18

1.3.2 Clasificaciones de las Vistas Arquitectónicas 19

1.4 Elementos a tener en cuenta en la Vista de Despliegue 23

1.4.1 Estrategias de Desarrollo 23

1.4.2 Estrategias de empaquetamiento de productos 25

1.4.3 Licencias de software....26

1.4.4 Tipos de Licenciamientos vistos desde la arquitectura....28

1.4.5 Mercadotecnia y estrategias de ingreso 28

1.5 Importancia de documentar la Arquitectura de software 29

1.6 Conclusiones....30

Capítulo 2: Licencias y estrategias de GESPRO....32

Introducción....32

2.1 Vista despliegue 32
2.2 Pasos de la vista de despliegue 32
2.3 Caracterización de la solución vista desde las licencias de los componentes que la forman. 33
2.3.1 Herramientas que componen el producto 35
2.4 Caracterización de la solución vista desde las estrategias de desarrollo empleadas para su elaboración 37
2.5 Definición del tipo de licenciamiento de la solución vista desde la arquitectura 38
2.6 Definición de las estrategias de ingreso vista desde la arquitectura 38
2.7 Estrategia de empaquetamiento para implantación y actualización 40
2.7.1 Activos reutilizables requeridos para el entorno de desarrollo 40
2.7.2 Activos reutilizables en forma de plugins 42
2.7.3 Activos compuestos que forman parte del producto final 45
2.8 Conclusiones 47
Capítulo 3: Despliegue y soporte de GESPRO. Validación de resultados 48
Introducción 48
3.1 Requerimientos mínimos de instalación 48
3.2 Estrategia para el soporte 50
3.2.1 Detalles de soporte nivel 3 50
3.3 Estrategia para el despliegue y actualización de la solución 52
3.4 Validación de la propuesta 56
3.4.1 Validación por preexperimento. 58
3.4.2 Gráfico de los tiempos de despliegue y configuración de la Plataforma GESPRO antes y después de la implantación de la propuesta 60
3.5 Conclusiones 61
Conclusiones Generales 62
Recomendaciones 63
Referencias Bibliográficas 64

Glosario de Términos 67

Índice de figuras

Fig. 1 Guía base de análisis arquitectónico. 22

Fig. 2 Proceso de actualización de GESPRO. 54

Fig. 3 Estructura de carpetas en el directorio raíz de la aplicación. 55

Fig. 4 Contenido del directorio *shared.* 55

Fig. 5 Contenido del directorio *releases*. 56

Fig. 6 Gráfico de los tiempos de despliegue y configuración. 61

Índice de tablas

Tabla 1 Tabla de Herramientas que componen el producto. 35

Tabla 2 Tabla de Gemas. 40

Tabla 3 Tabla de Plugins 42

Tabla 4 Tabla de activos compuestos. 45

Tabla 5 Tabla de requerimientos mínimos de Hardware 48

Tabla 6 Tabla de requerimientos mínimos de Software 49

Tabla 7 Tabla de niveles de soporte 50

Tabla 8 Horarios de servicio 51

Tabla 9 Medios de acceso al servicio 51

Tabla 10 Selección de estrategia para garantizar el despliegue. 53

Tabla 11 Selección de estrategia para garantizar la actualización. 53

Tabla 12 Validación de la Propuesta de Despliegue de la Arquitectura de GESPRO 58

Introducción

El mundo actual se encuentra inmerso en un constante desarrollo que depende en gran medida de la vertiginosa evolución alcanzada por las Tecnologías de la Información y las Comunicaciones (TIC). Con el renacer de esta nueva era va perdiendo importancia el trabajo físico que requirió la revolución industrial, al mismo tiempo que adquiere mayor importancia el trabajo mental que necesita la nueva revolución informática.

Cuba a pesar del constante bloqueo impuesto por el gobierno de los Estados Unidos se ha trazado la tarea de informatizar la sociedad cubana, mejorando así el proceso de automatización de las empresas e instituciones importantes para el estado. Juega un papel fundamental en esta inmensa labor la Universidad de las Ciencias Informáticas (UCI), creada en el ámbito de la Batalla de Ideas por nuestro Comandante en Jefe para promover el desarrollo de la industria del software en el país.

En la UCI se llevan a cabo diferentes proyectos de software para dar solución a problemas dentro de la universidad, el país y a esfera internacional. En el año 2009 coexistían en la red de centros de la UCI ocho herramientas para la Gestión de Proyectos. Una entrevista realizada a especialistas de CALISOFT (30) arrojó que del total de proyectos de la universidad, un 2 % empleaba la herramienta *Team Foundation Server*, el 21 % *GForge*, mientras que un 16 % *Redmine* y otro 15 % el *Trac*. Para *Bugzilla, DotProject, Jira* y *Microsoft Team System* se obtenía un 1 % de uso respectivamente. Se reveló además que el 42 % restante de los proyectos no utilizaban ninguna herramienta. Esta situación provocaba dificultades relacionadas con la toma de decisiones, la integración de la información, así como la aparición de una estrategia única para el tratamiento de las licencias de los productos utilizados en la universidad, y las facilidades para el soporte y la actualización de los mismos.

Con el objetivo de unificar la información, y lograr un mejor control y seguimiento de los proyectos se desarrolla en el año 2010, por parte de la Dirección Técnica de la Universidad, el Paquete de Gestión de Proyectos GESPRO.

Ante las dificultades existentes, se realiza un análisis de la situación integral de la Plataforma y se identifican los siguientes problemas:

- Potencialidades no explotadas para el desarrollo de soluciones de gestión de proyectos y la comercialización nacional e internacional de las mismas, garantizando el montaje de modelos de investigación-desarrollo sostenibles y las estrategias para la generación de ingresos y la comercialización.

- Dificultades para la estimación de los requerimientos mínimos de software y hardware para el despliegue de nuevas plataformas para la gestión de proyectos.

- Dificultades con la gestión de las licencias y tipos de licenciamiento asociados.

- Dificultades para la gestión del soporte de las plataformas de gestión de proyectos y la actualización de las soluciones.

- Se identificó además que los períodos de despliegue, configuración y actualización de las plataformas eran largos y costosos, pudiéndose constatar además, en los escenarios externos de la universidad, dificultades para la logística que obligaba la necesaria reducción para estos tiempos de despliegue, configuración y actualización (32).

Por tanto el **problema a resolver** queda definido como: Las insuficiencias en el Modelo de Actualización y Despliegue de las herramientas de la **Arquitectura del Sistema GESPRO v1.0** afecta los tiempos de despliegue, configuración y actualización de las soluciones basadas en esta arquitectura. Es por ello que se define como **objeto de estudio** la Arquitectura del Sistema GESPRO.

Se persigue como **objetivo general:** Definir e implantar la Vista de Despliegue de la Arquitectura del Sistema GESPRO 12.05 de forma tal que disminuyan los tiempos de despliegue, configuración y actualización de las soluciones basadas en esta arquitectura. El **campo de acción** queda enmarcado en**:** La Vista de Despliegue de la Arquitectura del Sistema GESPRO.

Para darle cumplimiento al **objetivo general** se definen los siguientes objetivos específicos:

- ✓ Elaborar el marco teórico de la investigación.
- ✓ Definir la Vista de Despliegue de la Arquitectura del Sistema GESPRO.
- ✓ Implantar la Vista de Despliegue de la Arquitectura del Sistema GESPRO.
- ✓ Validar los resultados esperados.

El **tipo de investigación** para poder resolver el problema que se muestra será la **explicativa**.

Se plantea como **hipótesis** lo siguiente**:** Si se define e implanta la Vista de Despliegue de la Arquitectura del Sistema GESPRO se disminuirán los tiempos de despliegue, configuración y actualización para la solución desplegada en la UCI.

Se tiene la **variable dependiente:** Tiempo de despliegue, configuración y actualización de las soluciones basadas en la Arquitectura del Sistema GESPRO 12.05

La **población** a estudiar será: los centros de la red de producción de la UCI, tomando como **muestra**: la red de centros de la sede central.

Para realizar la investigación se utilizan los siguientes **métodos**:

Los métodos teóricos:

1. El método **Histórico-Lógico:** Se utiliza en el análisis de las tendencias actuales de los modelos arquitectónicos.
2. El método **Análisis y Síntesis:** Se utiliza en el trabajo con la información y el arribo a las conclusiones de la investigación.

Los métodos empíricos:

1. El método de la **Entrevista**: Se utiliza este método en la investigación ya que gran parte de la información está en manos de los especialistas.

Con la definición e implantación de la Vista de Despliegue de la Arquitectura del Sistema GESPRO 12.05 se espera como **aporte práctico** optimizar el proceso de despliegue, configuración y actualización del mismo, a partir de la disminución de los tiempos para estos procesos, haciendo más factible el rendimiento y la eficacia de dicha herramienta.

El **diseño de investigación** es el **preexperimento** de Pre y Post prueba con un solo grupo, en la que hay al menos un punto de referencia inicial. La validez interna puede ser afectada fácilmente por la historia, la maduración, la elección de un grupo atípico, la regresión y las interacciones.

Este trabajo se divide en tres capítulos para darle solución a la investigación:

Capítulo 1: Fundamentación Teórica: En este capítulo se trata lo referente a la parte teórica de la investigación, donde se muestran los principales conceptos de Arquitectura de Software y de sus vistas arquitectónicas. Se analizan además algunos de los modelos arquitectónicos más utilizados en el mundo, en nuestro país y aquí en la Universidad, concluyendo sobre las ventajas de algunos con respecto a otros, para finalmente saber cuál es más factible aplicar al problema. Con esta minuciosa investigación se espera un mejor desarrollo del trabajo.

Capítulo 2: Diseño arquitectónico: En este capítulo se propone parte de la solución al problema planteado en el diseño teórico de la investigación.

Capítulo 3: Evaluación de la Arquitectura Propuesta: En este capítulo se culmina la solución al problema planteado en el diseño teórico de la investigación y se evalúa la propuesta completa para analizar los resultados obtenidos.

Capítulo 1: Fundamentación Teórica

Introducción

En el presente capítulo se realiza la fundamentación teórica de la investigación a través del estudio del arte de la Arquitectura de Software. Se exponen brevemente los principales conceptos asociados a esta disciplina, haciendo énfasis en el término Vista Arquitectónica. Se realiza un estudio de las principales escuelas de la arquitectura y de los modelos arquitectónicos asociados a las mismas, arribando a conclusiones de las ventajas que brinda el modelo de las 9+1 vistas para la elaboración de la Vista de Despliegue. Se realiza un breve estudio de los principales aspectos a tener en cuenta para el desarrollo de esta vista tales como: las diferentes estrategias a tomar, los tipos de licenciamientos, así como los temas de implantación y soporte que deberán tomarse en cuenta para el éxito de la presente investigación.

1.1 Definiciones de la Arquitectura de Software

Actualmente es posible encontrar numerosas definiciones del término Arquitectura de Software, cada una con planteamientos diversos y ninguna adoptada completamente por la totalidad de los arquitectos del mundo. Se hace evidente que su conceptualización sigue todavía en discusión, debido a que no existe un estándar que pueda ser tomado como marco de referencia.

La definición oficial de Arquitectura de Software se ha acordado que sea la que brinda el documento de IEEE Std 1471-2000, adoptada también por Microsoft, que expresa lo siguiente: “ 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” (2) .

Mary Shaw y David Garlan, sugieren que la arquitectura del software sea un nivel del diseño referido a las ediciones “...más allá de las estructuras de los algoritmos y de datos del cómputo; diseñar y especificar la estructura del sistema total emerge como nueva clase de problema. Las ediciones estructurales incluyen la organización gruesa y la estructura global del control; los protocolos para la comunicación, la sincronización, y el acceso a los datos; asignación de la

funcionalidad para diseñar elementos; distribución física; composición de los elementos del diseño; escalamiento y funcionamiento; y selección entre alternativas del diseño" (3).

Esta definición no es capaz de definir el momento en el ciclo de desarrollo del software donde se debe llevar a cabo la arquitectura, según esta, el desarrollo de la misma comienza en un momento entre la definición de los requisitos y la modelación del sistema o entre la modelación del sistema, el análisis y el diseño (4).

Rational Unified Process, 1999, propone que:

"Una arquitectura es el sistema de decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y de sus interfaces por los cuales el sistema es compuesto, junto con su comportamiento según lo especificado en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y del comportamiento en subsistemas progresivamente más grandes, y el estilo arquitectónico que dirigen esta organización, los elementos y sus interfaces, sus colaboraciones y su composición" (5).

La misma define la arquitectura como una etapa más de la Ingeniería de Software enfocada a la orientación a objetos y a UML, como lo plantea la metodología de desarrollo RUP, representa la arquitectura como algo más que la simple composición de herramientas o tecnologías a usar en el desarrollo del sistema (4).

Por otra parte Bass define de la siguiente forma: "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" (6).

De manera similar aparece otra de las definiciones más reconocidas de la academia, la de Paul Clements: "La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones" (2).

A pesar de la cantidad de definiciones que existen de Arquitectura de Software, la gran mayoría coincide en que esta disciplina se refiere a la estructura a grandes rasgos del sistema (alto nivel de abstracción), estructura consistente en componentes y relaciones entre ellos.

Se puede concluir que la Arquitectura de Software es: una disciplina independiente de cualquier metodología de desarrollo, representa un alto nivel de abstracción del sistema y responde a los requerimientos no funcionales, debido a que estos se satisfacen mediante el modelado y diseño de la aplicación.

1.2 Tendencias de la Arquitectura

Se pueden distinguir a grandes rasgos seis corrientes representativas o tendencias: (7), (8).

- Arquitectura como etapa de ingeniería y diseño orientada a objeto.
- Arquitectura estructural, basada en un modelo estático de estilos, ADLs y vistas.
- Estructuralismo arquitectónico radical.
- Arquitectura basada en patrones.
- Arquitectura procesual.
- Arquitectura basada en escenarios.

1.2.1 Arquitectura como etapa de la ingeniería de software orientada a objeto

Esta corriente arquitectónica está basada en el modelo de James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman, combinado estrechamente el mundo de UML y Rational (9). En esta referencia la arquitectura se restringe a las fases preliminares del proceso, enfocándose en los niveles más elevados de abstracción. Importa más la abundancia, el detalle de diagramas y técnicas disponibles que la visión estructural del conjunto. La definición de arquitectura que se promueve en esta corriente tiene que ver con aspectos formales a la hora del desarrollo. Las definiciones revelan que la Arquitectura de Software, en esta perspectiva, concierne a decisiones sobre organización, selección de elementos estructurales, comportamiento, composición y estilo arquitectónico susceptibles de ser descritas a través de las cinco vistas clásicas del modelo 4+1 de Kruchten (10).

1.2.2 Arquitectura estructural, basada en un modelo estático de estilos, ADLs y vistas

La Arquitectura estructural constituye la corriente fundacional y clásica de la disciplina. Los representantes de esta corriente son todos académicos, mayormente de la Universidad

Carnegie Mellon en Pittsburgh: Mary Shaw, Paul Clements, David Garlan, Robert Allen, Gregory Abowd, John Ockerbloom. En toda la corriente, el diseño arquitectónico no sólo es el de más alto nivel de abstracción, sino que además no tiene por qué coincidir con la configuración explícita de las aplicaciones; rara vez se encontrarán referencias a lenguajes de programación o piezas de código (12), (13). La potente conceptualización de las estructuras y marcos analíticos de la arquitectura, soportados en los cánones teóricos de la disciplina, hacen de esta escuela el punto de referencia a seguir en la continuidad de esta investigación.

1.2.3 Estructuralismo arquitectónico radical

Se trata de un desprendimiento de la corriente anterior, mayoritariamente europeo, que asume una actitud más radical con el mundo UML (7), (14). En el seno de este movimiento hay al menos dos tendencias, una que excluye de plano la relevancia del modelado orientado a objetos para la arquitectura de software y otra que procura definir nuevos metamodelos y estereotipos de UML como correctivos de la situación (15).

1.2.4 Arquitectura basada en patrones

Esta corriente se basa principalmente en la redefinición de los estilos como patrones arquitectónicos o de diseño (16), (17). El diseño consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura, con características de solución a problemas arquitectónicos comunes y clasificados. La alta potencialidad que ofrece su enfoque y el volumen de soluciones tipo presentes, resultan puntos de referencias incorporados en la propuesta de solución, con el objetivo de garantizar la homogenización de soluciones y en la viabilidad de la comunicación entre arquitectos y el propio equipo de desarrollo (11).

1.2.5 Arquitectura procesual

Desde comienzos del siglo XXI, con centro en el SEI (Software Engineering Institute),, los arquitectos de Carnegie Mellon de la primera generación y muchos nombres nuevos de la segunda: Rick Kazman, Len Bass, Paul Clements, Felix Bachmann, Fabio Peruzzi, Jeromy Carrière, Mario Barbacci, Charles Weinstock. La Arquitectura procesual intenta establecer modelos de ciclo de vida y técnicas de diseño de requerimientos, diseño, análisis, selección de alternativas, validación, comparación, estimación de calidad y justificación económica específicas para la arquitectura de software. Otras variantes dentro de la corriente procesual se caracterizan por cambios en la etapa del proceso de extracción de la arquitectura, generalización y reutilización (8).

El enfoque procedimental del modelo de gestión y creación de la arquitectura que esta corriente brinda, resultaron significativos para que el presente trabajo, incorporara conceptos procedimentales de la creación de diseño abstracto de alto nivel, que se formaliza en vistas arquitectónicas, con sus respectivos nexos metodológicos, para gestionar, integrar o colaborar, en el proceso creativo de la solución de software.

1.2.6 Arquitectura basada en escenarios

La Arquitectura basada en escenarios es la corriente más nueva. Se trata de un movimiento predominantemente europeo con centro en Holanda. Recupera el nexo de la arquitectura de software con los requerimientos y la funcionalidad del sistema, ocasionalmente borroso en la arquitectura estructural clásica. En esta corriente suele utilizarse diagramas de casos de uso UML como herramienta ocasional, dado que los casos de uso son uno de los escenarios posibles y que no están orientados a objeto. Los autores vinculados con esta modalidad han sido, aparte de los codificadores de ATAM, CBAM, QASAR y demás métodos del SEI, los arquitectos holandeses de la Universidad Técnica de Eindhoven, de la Universidad Brije, de la Universidad de Groningen y de Philips Research: Mugurel Ionita, Dieter Hammer, Henk Obbink, Hans de Bruin, Hans Van Vliet, Eelke Folmer, Jilles Van Gurp y Jan Bosch. La presencia de holandeses es significativa; la Universidad de Eindhoven es, incidentalmente, el lugar en el que surgió lo que P. I. Sharp proponía llamar la escuela arquitectónica de Dijkstra (11).

En la literatura consultada se pueden distinguir seis corrientes representativas, pero en ninguna de ellas se integran de manera sistémica los elementos teóricos de las restantes, sin embargo se alcanza a ver que predomina como factor común de elementos conceptuales conductivos en el diseño y desarrollo de la arquitectura de software, las vistas arquitectónicas, que describen características de la arquitectura en un plano especifico de la solución, según el interés de involucrados (11). También la mayoría de los framework y estrategias consultadas reconocen las vistas como la estructura central para la formalización y descripción de las arquitecturas (25).

1.3 Vistas de la Arquitectura de Software

1.3.1 Definiciones de Vista Arquitectónicas

Los marcos arquitectónicos, al igual que las metodologías de modelado de los organismos, acostumbran a ordenar las diferentes perspectivas de una arquitectura en términos de vistas.

Desde los inicios de la arquitectura de software se planteó la necesidad de contar con varias vistas para separar los diferentes elementos del software, como se describe en E. Perry, Dewayne y L. Wolf, Alexander, 1992:

'…el arquitecto del software necesita un numero de vistas diferentes de la arquitectura de software para varios usos y usuarios…son requeridas para enfatizar y entender diferentes aspectos de la arquitectura…'

Existen diferentes definiciones para el término Vista: "un subconjunto resultante de practicar una selección o abstracción sobre una realidad, desde un punto de vista determinado" (13).

De acuerdo a la definición del estándar 1471 de la IEEE, es lo siguiente:

"Una vista es una representación de un sistema completo desde la perspectiva de un conjunto de concerns relacionados" (24).

En conclusión una vista no es más que una temática desde la que se enfoca el proyecto, ya sea desde el punto de vista de los datos, integración, etc. y que hace posible enfrentar el diseño de la arquitectura desde diversas perspectivas reduciendo la complejidad presente en su desarrollo.

Las vistas arquitectónicas constituyen las estructuras fundamentales de la arquitectura. El diseño de las vistas arquitectónicas implica establecer una sincronización entre sus elementos para conformar una arquitectura integrada, y de poder ser capaz de mantener la consistencia entre sus elementos. Cada vista agrupa a elementos que pertenecen a diferentes intereses, además de esto se llegan a establecer relaciones de correspondencias entre los elementos de distintas vistas puesto que al formar parte de un mismo sistema de alguna manera se llegan a relacionar. Estas relaciones de correspondencia constituyen el adhesivo que mantiene enlace a las vistas que integran a la arquitectura del software.

Para separar las vistas se debe tener en cuenta todo el trabajo que se lleva a cabo a la hora de estructurar una arquitectura, tratando de no dejar ningún contenido fuera de alguna de ellas. Esto como es lógico no es una labor nada sencilla, y mientras más se crece en los conocimientos arquitectónicos, más complejo y engorroso se vuelve, generando muchas dudas sobre dónde va un contenido, si es suficiente con integrarlo a alguna vista ya propuesta o si es indispensable crear una nueva vista solo para él.

1.3.2 Clasificaciones de las Vistas Arquitectónicas

David Parnas (´74) las estructura en tres grupos**:**

- Estructura de módulos: es parte de, o comparte el mismo secreto que la asignación de trabajo.
- Estructura de uso: depende de la corrección de programas.
- Estructura de procesos: brinda trabajo computacional a procesos (17).

Perry y Wolf (´94): Reconocen la necesidad de vistas que enfatizan ciertos aspectos arquitectónicos útiles para distintas audiencias o para diferentes propósitos.

Siemens Corporate Research (´95) propone:

- Vista conceptual: principales elementos de diseño y su interrelación.
- Vista de módulos: estructura funcional y de capas.
- Vista de ejecución: estructura dinámica.
- Vista de código: organización de código fuente, binarios y bibliotecas en el ambiente de desarrollo (17).

Libro: "Software Systems Architecture" de Nick Rozanski y Eóin Woods.

- Vista funcional.
- Vista de información.
- Vista de concurrencia.
- Vista de desarrollo.
- Vista de despliegue.
- Vista operacional (17).

Propuestas de Microsoft

- Vista Conceptual: Es usada para definir los requerimientos funcionales y la visión que los usuarios del negocio tienen de la aplicación y describir el modelo de negocio que la arquitectura debe cubrir. Esta vista muestra los subsistemas y módulos en los que se divide la aplicación y la funcionalidad que brinda dentro de cada uno de ellos. Casos de Uso, Diagramas de Actividad, Procesos de Negocio Entidades del Negocio, etc.

- Vista Lógica: Muestra los componentes principales de diseño y sus relaciones de forma independiente de los detalles técnicos y de cómo la funcionalidad será implementada en la plataforma de ejecución. Los arquitectos crean modelos de diseño de la aplicación, los cuáles son vistas lógicas del modelo funcional y que describen la solución. Realización de los Casos de Uso, subsistemas, paquetes y clases de los casos de uso más significativos arquitectónicamente.

- Vista Física: Ilustra la distribución del procesamiento entre los distintos equipos que conforman la solución, incluyendo los servicios y procesos de base. Los elementos definidos en la vista física se "mapean" a componentes de software (servicios, procesos, etc.) o de hardware que definen más precisamente cómo se ejecutará la solución.

- Vista Implementación: Describe cómo se implementan los componentes físicos mostrados en vista de distribución agrupándolos en subsistemas organizados en capas y jerarquías, ilustra, además las dependencias entre éstos. Básicamente, se describe el mapeo desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos (17).

Propuestas de Arquitectura 4+1 Vistas de Kruchten (RUP)

- Vista lógica: requerimientos de comportamiento, mecanismos comunes de diseño (basada en diagramas de objetos y clases).
- Vista de procesos: distribución, integridad, tolerancia a fallas (basada en describir una red lógica de programas que se comunican).
- Vista de desarrollo o implementación: rehúso, portabilidad, asignación de requerimientos y trabajo de equipos. Organización del software en el ambiente de desarrollo.
- Vista física: disponibilidad, confiabilidad, performance, escalabilidad. Mapea elementos de las otras vistas a nodos de procesamiento.
- Vista de Casos de Uso o Escenarios: definición de procesos, agrupamiento en paquetes (17).

Otro de los modelos estudiados es el de las 9+1 vistas, propuesta desarrollada en la UCI, en el cual se propone un modelo arquitectónico de referencia donde se integran las corrientes más representativas, centrándose en el papel de este proceso en la gestión de los proyectos informáticos, la definición de los productos y su evolución.

Este modelo presenta nueve vistas y además una guía base de análisis arquitectónico (Figura 1), desde la perspectiva de procesos, que facilita el ordenamiento y priorización de estas vistas arquitectónicas, es además un elemento normativo, formativo y de auto evaluación del arquitecto (25).

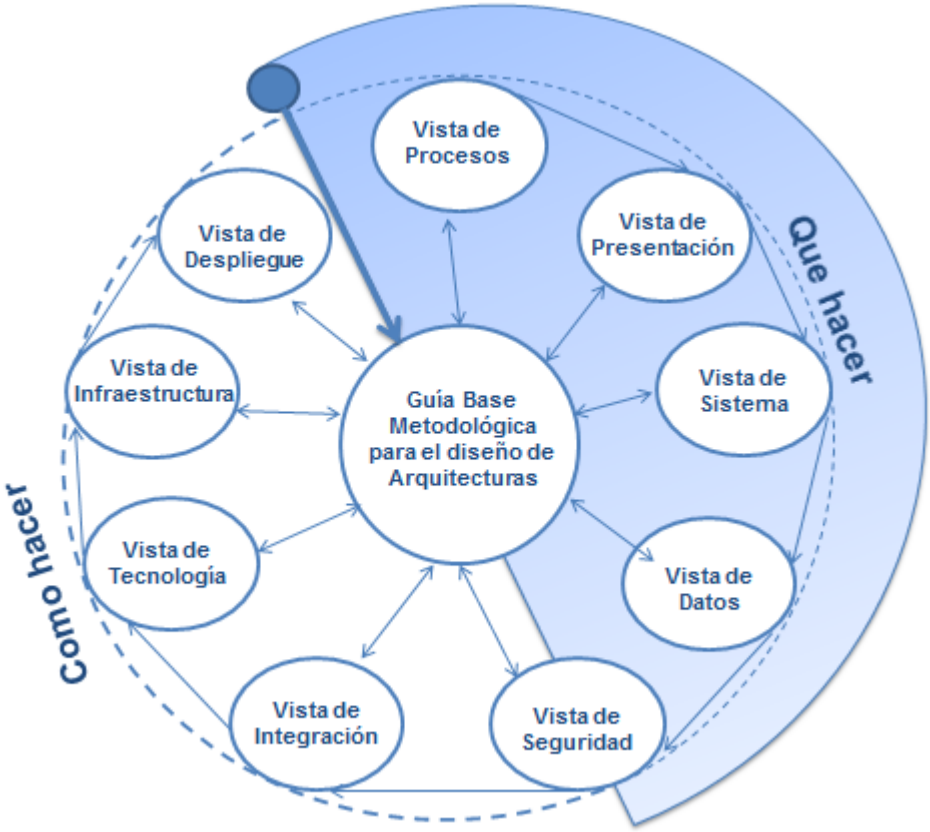

Fig. 1 Guía base de análisis arquitectónico (25).

Las vistas usadas para una arquitectura dependen de las necesidades y el uso que se le dará a las mismas, por ello para cada arquitectura independientemente de cualquier metodología existente, se puede contar con tantas vistas como sean necesarias, mientras su estructura y diseño lo exijan.

Estas son algunos modelos arquitectónicos existentes y no todos resultan óptimos para la solución de este trabajo, por ejemplo, RUP en sus 4+1 vistas, descuida los temas referidos a Datos, Seguridad, Presentación, Sistema y el Despliegue, siendo este último de gran

importancia para el desarrollo de la presente investigación. David Parnas, trata elementos a los que Siemens Research hace adiciones y viceversa, y ambos también descuidan el despliegue. Por su parte Perry y Wolf proponen vistas que enfatizan ciertos aspectos arquitectónicos útiles para distintas audiencias o para diferentes propósitos, propuesta que se queda demasiado general. En el libro Software Systems Architecture de Nick Rozanski y Eóin Woods, se abordan 5 vistas que abarcan un amplio contenido, pero no tratan las disciplinas de seguridad e integración. Siendo así se llega a la conclusión que el modelo de las 9+1 vistas propuesto en la Universidad es el más adecuado para desarrollar el presente trabajo, pues se incluye el despliegue como una vista más, donde en cada una de ellas se caracteriza el escenario (condiciones específicas de funcionamiento, respuesta del sistema y características relevantes de su entorno) elementos que justifican ampliamente la decisión tomada evitando subjetividades (25).

1.4 Elementos a tener en cuenta en la Vista de Despliegue

La Vista de Despliegue cuenta con un grupo de elementos de suma importancia. Entre ellos están los diferentes tipos de estrategias, tanto de desarrollo como de empaquetamiento del producto que se van a emplear en la solución. También es importante conocer de las licencias de los componentes que va a tener la misma, así como los elementos fundamentales concernientes a la Instalación, Implantación y Soporte del producto. Para la mejor toma de decisiones se hace necesario un estudio de estos elementos anteriormente mencionados.

1.4.1 Estrategias de Desarrollo

Dado los exigentes requerimientos en el cumplimiento de tiempos y presupuestos que las empresas desarrolladoras de software viven constantemente, el rendimiento en el proceso de desarrollo ha constituido una necesidad creciente. Algunas organizaciones han establecido diversas estrategias de trabajo, e incluso han agrupado las mejores prácticas en esta materia con el fin de industrializar el desarrollo de software.

A raíz de esta evolución en el desarrollo de los sistemas, varias organizaciones encontraron como una adecuada práctica (la cual le proporcionaría mayor calidad, menor tiempo de desarrollo y mayores ingresos), el construir activos de software de sistemas comunes que compartieran un mismo dominio de solución. La aplicación de estas prácticas requirió de un mayor esfuerzo en las organizaciones, más coordinación, control y disciplina en el trabajo (26).

Existen varios modelos de desarrollo, entre ellos está el propuesto por Fernandes en el 2004 que se encuentra encaminado a clasificar las fábricas en dependencia de su alcance o dominio de solución en el cual se desarrolla. Existe además el Modelo propuesto por Basili que divide una LPS (Línea de Productos de Software) en dos grandes entidades basadas en proyectos y línea de componentes. También se encuentra el denominado Hildebrando, el cual enfoca su desarrollo en el tiempo, recursos y alcance del sistema haciendo énfasis en la pro actividad de las acciones y riesgos para el éxito del proyecto. El SEI (Software Engineering Institute), capturó las actividades esenciales de una línea de producción de software según lo planteado por los autores (Clements, y otros) y desarrolló un framework, basado en tecnologías de la Web denominado “Framework para las áreas prácticas de las líneas de productos de Software” el cual recoge lo esencial para la “ingeniería de sistemas, la administración técnica y la administración de la organización”. Existe además el Modelo basado en la Norma ISO - 9001 y CMM, este modelo basa su principal organización y estructura industrial en las normas CMM e ISO – 9001 respectivamente. Divide el trabajo del modelo en cinco entidades, y basa sus principales procesos en normas y estándares certificados (26).

Otros de los modelos estudiados es el Modelo UCI – GESPRO, este modelo propone un grupo de entidades encargadas del desarrollo de productos informáticos en un modelo de líneas de producción de software. Basa sus principios en las líneas de productos de software, la arquitectura de empresas, la mejora continua y las corrientes más actuales de reutilización y desarrollo basado en componentes (26).

Define seis entidades: Ingeniería de Dominio, Ingeniería de Aplicación, Gestión de Recursos Humanos, Gestión del Conocimiento y la Configuración, Gestión de la Calidad y Gestión de la Línea. Adopta las mejores prácticas y tendencias mundiales sobre esta rama (26).

Otro de los modelos es el propuesto por Ing. Henry Cruz Mulet, de la Universidad de las Ciencias Informáticas, en el año 2011 en su tesis de maestría, donde se definen elementos importantes que forman parte de la familia de productos como son: la mercadotecnia, los tipos de licencia de los productos y sus clasificaciones, las estrategias a emplear, los activos de software entre otros aspectos de interés que serán de gran utilidad en la toma de decisiones para la solución de este trabajo. Entre las estrategias de desarrollo de la propuesta de Henry aparecen:

Desarrollo de código abierto único proveedor: El software es distribuido usando una licencia de software libre y todas las contribuciones y extensiones son publicadas pero el desarrollo es dominado por una única organización suministradora.

Desarrollo comunitario de código abierto: El software es distribuido usando licencias de código abierto y es desarrollado públicamente por una comunidad de individuos u organizaciones suministradoras.

Desarrollo mixto de código abierto: El producto o servicio es suministrado por un único proveedor y está basado en la combinación de proyectos desarrollados en software libre desarrollados públicamente por comunidades u otras organizaciones.

Desarrollo híbrido: Se refiere a modelos de desarrollo donde el software es distribuido usando licencias de código abierto pero algunas funcionalidades son desarrolladas a puertas cerradas, se destacan en este sentido las siguientes tres categorías:

- Proveedores híbridos: El desarrollo es dominado por los desarrolladores de una única organización y algunas funcionalidades son desarrolladas a puertas cerradas.
- Comunidad híbrida: La mayor parte del software es desarrollado de forma pública por una comunidad de individuos o suministradores, y algunas funcionalidades son desarrolladas a puertas cerradas por proveedores del producto o servicio.
- Híbrido mixto: El producto o servicio está basado en la combinación de proyectos, que usan desarrollos de código abierto desarrollado públicamente por comunidades o múltiples proveedores y algunas funcionalidades son desarrolladas a puertas cerradas.

Desarrollo completamente cerrado: Todos los componentes del software y funcionalidades son cerrados por la entidad productora y no se basan en software libre.

1.4.2 Estrategias de empaquetamiento de productos

Durante el despliegue del software se elabora un plan que incluye el empaquetamiento del producto, en este se hace entrega de la versión liberada del software, los manuales de usuario y guías para la instalación, al concluir, el producto estará listo para la entrega al usuario final.

Para el empaquetamiento se utilizan los activos de software, este término abarca todos los artefactos que se generan de un software y que puedan ser reutilizados. Montilva (28) lo clasifica como “un producto de software diseñado expresamente para ser utilizado múltiples veces en el desarrollo de diferentes sistemas o aplicaciones”. Un activo de software puede ser:

- Un componente de software
- Una especificación de requisitos
- Un modelo de negocios
- Una especificación de diseño
- Un algoritmo
- Un patrón de diseño
- Una arquitectura de dominio
- Un esquema de base de datos
- Una especificación de prueba
- La documentación de un sistema
- Un plan (28)

Activos Reutilizables: Lo forman, las clases, funcionalidades, componentes. Estilos arquitectónicos bases para el desarrollo de los productos.

1.4.3 Licencias de software

La licencia de software es una especie de contrato, en donde se especifican todas las normas y cláusulas que rigen el uso de un determinado programa, principalmente se estipulan los alcances de uso, instalación, reproducción y copia de estos productos (29).

El tema de las licencias de software puede ser muy complejo. El negocio del software se basa en licencias binarias. La propiedad intelectual de los distribuidores de software comercial nace del código fuente. Las licencias de software se crean con diversos fines empresariales y para afrontar diversos tipos de relaciones (como distribuidor/cliente y partner/partner). Los desarrolladores de software tanto comercial como no comercial utilizan decenas de licencias que abarcan una gran variedad de términos y condiciones (29).

Las licencias de uso de software generalmente se clasifican en alguna de estos tipos:

- (Permisiva) Licencias de software de código abierto permisivas: Se puede crear una obra derivada sin que ésta tenga obligación de protección alguna. Muchas licencias pertenecen a esta clase, entre otras.

- (No Permisiva) Licencias de software de código abierto robustas fuertes o con copyleft fuerte: Contienen una cláusula que obliga a que las obras derivadas o modificaciones

que se realicen al software original se deban licenciar bajo los mismos términos y condiciones de la licencia original.

- (No Permisiva) Licencias de software de códigos abiertos robustos débiles, con copyleft débil/suave o híbridas: Contienen una cláusula que obliga a que las modificaciones que se realicen al software original se deban licenciar bajo los mismos términos y condiciones de la licencia original, pero que las obras derivadas que se puedan realizar de él puedan ser licenciadas bajo otros términos y condiciones distintas.
- Licencias propietarias: Son licencias de productos propietarios que debió comprar para poder comercializar su producto o licencias propietarias propia que usted cree.

Entre las licencias más reconocidas están:

- La Licencia Pública General o GPL que otorga al usuario la libertad de compartir el software licenciado bajo ella, así como realizar cambios en él. Es decir, el usuario tiene derecho a usar un programa licenciado bajo GPL, modificarlo y distribuir las versiones modificadas de éste (29).

- La Licencia Pública General Menor o LGPL que es una modificación de la licencia GPL. La LGPL reconoce que muchos desarrolladores de software no utilizarán el código fuente que se distribuya bajo la licencia GPL, debido a su principal desventaja que determina que todos los derivados tendrán que seguir los dictámenes de esa licencia. La LGPL permite que los desarrolladores utilicen programas bajo la GPL o LGPL sin estar obligados a someter el programa final bajo dichas licencias (29).

- La Licencia de Distribución de Software de Berkeley o BSD que no impone ninguna restricción a los desarrolladores de software en lo referente a la utilización posterior del código en derivados y licencias de estos programas. Este tipo de licencia permite a los programadores utilizar, modificar y distribuir a terceros el código fuente y el código binario del programa de software original con o sin modificaciones. Los trabajos derivados pueden optar a licencias de código abierto o comercial (29).

- La Licencia Apache que es una descendiente de la licencia BSD. Permite al desarrollador hacer lo que desee con el código fuente, incluso productos propietarios,

sin entregar el código fuente. La única restricción es que se reconozca el trabajo del desarrollador (29).

- La licencia MIT (*Massachusetts Institute of Technology*) que permite reutilizar el software así licenciado tanto para ser software libre como para ser software no libre, permitiendo no liberar los cambios realizados al programa original. También permite licenciar dichos cambios con licencia BSD, GPL, u otra cualquiera que sea compatible (es decir, que cumpla las cláusulas de distribución) (29).

1.4.4 Tipos de Licenciamientos vistos desde la arquitectura

Licenciamiento dual: El código base es licenciado a diferentes usuarios como código abierto o como licenciamiento comercial.

Licenciamiento Núcleo-Abierto: El núcleo de la solución es abierto bajo licencia de código abierto. Pero las versiones profesionales de los productos incluyen desarrollos de código abierto y cerrado, y son licenciadas comercialmente.

Licenciamiento Abierto y Cerrado: Los proyectos de código abierto son desarrollados a partir de la unión de partes independientes que se pueden comercializar bajo licenciamiento comercial desarrolladas como código cerrado.

Código abierto: La solución es comercializada bajo una licencia de código abierto solamente.

Código abierto ensamblado: El producto o servicio incluye código de algunos proyectos que utilizan múltiples licencias de código abierto.

Cerrado: El producto está basado en código abierto pero no está disponible bajo licenciamiento de código abierto.

1.4.5 Mercadotecnia y estrategias de ingreso

La mercadotecnia, es una función de la línea, que tiene la finalidad de identificar a los mercados meta para satisfacer sus necesidades o deseos, mediante una adecuada implementación de sus funciones. Por otra parte, comprende una serie de actividades (identificación de oportunidades, investigación de mercados, formulación de estrategias y tácticas, etc.) con objetivos propios, pero que están estrechamente interrelacionados con otros departamentos, para de esta manera servir a los objetivos globales de la empresa. Su correcto funcionamiento, va encaminado a la capacidad que posea de evaluar la fiabilidad del producto,

que necesita ser desarrollado para el mercado, así como la capacidad que posea esta entidad para insertar estos en el mercado. Esto conlleva a que trabaje estrechamente vinculada, con el resto de las entidades, debido a que debe conocer la calidad, precio, rendimiento de la producción de la línea para asumir nuevos retos (26). Entre los principales procesos que en ella se realizan está la definición de las estrategias de ingresos que forma parte de la solución del presente trabajo.

Una de las principales tareas de la mercadotecnia es la definición acerca de cuáles vías efectuarán las formas de ingreso de la organización, a través de los productos y servicios que ella brinda, estas formas de ingreso pueden ser:

- **Licenciamiento comercial:** Es un licenciamiento bajo una licencia comercial en la forma tradicional.
- **Subscripciones**: Es donde se contratan servicios de soporte, actualizaciones, por un periodo de tiempo que generalmente es anual.
- **Servicios y soporte:** Es generalmente soportado por llamadas telefónicas, entrenamiento y contratos de consultoría.
- **Hardware empotrado:** El código abierto es empotrado en hardware que son generalmente distribuidos bajo licencias comerciales.
- **Software empotrado:** El código abierto es empotrado en paquetes de soluciones integrales comerciales (soluciones tecnológicas integrales).
- **Software como servicio (SaaS):** Los usuarios pagan por el acceso y el uso del software vía internet.
- **Publicidad:** El software es libre para ser usado bajo condiciones de publicidad.
- **Desarrollo a la medida:** Los usuarios pagan por el desarrollo de software personalizado para sus requerimientos específicos.
- **Otros productos y servicios:** El código abierto no es utilizado directamente para generar ingresos sino que otros productos y servicios son los que generan los mismos.

1.5 Importancia de documentar la Arquitectura de software

Documentar la arquitectura de software es cuestión de documentar las vistas que a su consideración son relevantes. Clements plantea en el libro “Documentación de la Arquitectura de Software”:

“El principio básico de documentar una arquitectura como un conjunto de vistas aporta una ventaja de divide y vencerás a la tarea de documentación, pero si las vistas son irrevocablemente diferentes, sin relación la una con la otra, nadie podría entender el sistema como un todo”.

Un sistema pobremente documentado carece de valor aunque haya funcionado bien en alguna ocasión. En el caso de programas pequeños y poco importantes que sólo se utilizan durante un corto período de tiempo, unos cuantos comentarios en el código podrían ser suficientes. No obstante, la mayoría de los programas cuya única documentación es el código, se quedan obsoletos rápidamente y es imposible mantenerlos. A menos que se sea infalible y se viva en un mundo en el que nada cambia, se tendrá que volver a consultar el código que ya estaba escrito y se pondrán en duda decisiones que se tomaron durante el desarrollo del mismo. Si no se documentan las decisiones se estará siempre cometiendo los mismos errores y tratando de comprender lo que se pudo haber descrito fácilmente en una ocasión. La falta de documentación no sólo genera trabajo adicional, sino también tiende a dañar la calidad del código (22).

Documentar la arquitectura de software es una tarea complicada y exige un criterio de ingeniería maduro. Documentar de forma concisa es un error habitual, pero si se escriben documentaciones extensas, estas atosigarán al lector y constituirán una carga en el momento de conservarlas. Es esencial documentar sólo los asuntos correctos. La documentación no sirve de ayuda para ningún involucrado si su extensión desalienta a las personas cuando la consultan.

Muchas veces los arquitectos no documentan los proyectos y no acompañan a los desarrolladores en el proceso de documentación. En ocasiones no se involucran de la forma adecuada en cada proyecto y sólo definen de vez en cuando un aspecto general de la arquitectura. A partir de esto, muchas empresas dedicadas al desarrollo de software, actualmente describen su arquitectura mediante un documento de arquitectura de software (22).

1.6 Conclusiones

En este capítulo se realizó un estudio de los principales temas relacionados con la arquitectura de software, el cual permitió sentar las bases teóricas para la correcta realización del trabajo que se pretende llevar a cabo. Se logró profundizar en los conocimientos básicos de la arquitectura tales como: los conceptos fundamentales asociados a esta disciplina, el estudio de las principales corrientes arquitectónicas que permitió concluir que todas tienen como factor común la necesidad de utilización de las Vistas arquitectónicas; sobre las mismas se dieron a conocer los conceptos ofrecidos por los principales conocedores del término. Se realizó un estudio de algunos de los modelos arquitectónicos puestos en práctica en el mundo y en la UCI, para concluir que el más indicado para el desarrollo del presente trabajo es el de las 9+1 vistas, donde se le confiere la debida atención a la Vista de Despliegue, principal objetivo del presente trabajo. Se abordaron además temas de suma importancia para esta vista como son los pasos a seguir para su elaboración, tales como: las estrategias de desarrollo y empaquetamiento, las licencias de los componentes del producto, los tipos de licenciamientos y los temas referentes a la Implantación y el soporte del producto.

Capítulo 2: Licencias y estrategias de GESPRO

Introducción

En el presente capítulo se realiza parte de la propuesta de solución a través de la elaboración de la Vista de Despliegue. Para ello se comenzará caracterizando la solución vista desde las licencias de los componentes que forman el producto. Se definirán además, el tipo de licenciamiento de la solución vista desde la arquitectura, las estrategias de ingreso y empaquetamiento del producto.

2.1 Vista despliegue

En esta vista se trabajan varios conceptos que influyen en el despliegue y la comercialización de las soluciones. Se discute acerca de los modelos de negocios, la composición del paquete de instalación, los requerimientos de instalación y soporte de las soluciones.

Se plantea como elemento fundamental para los modelos de negocios la combinación de modelos de desarrollo basado en código abierto y modelos de desarrollo propietarios, tomando lo mejor de ambos casos. A partir de los modelos de código abierto incrementar la calidad, disminuir los costos y aumentar las oportunidades de negocio por las atractivas ventajas para los clientes. Tomar de los modelos de desarrollo propietario las facilidades para el licenciamiento comercial.

Las mayores implicaciones para los desarrolladores de soluciones que se basan en código abierto están centradas en los siguientes elementos: las licencias de código abierto que incorporan a partir del rehúso de componentes, los modelos de desarrollo que las soportan y en función de esto la determinación de la estrategia de licenciamiento para la solución final y las estrategias para los ingresos. Por este motivo los primeros pasos para la construcción de esta vista tratan estos elementos.

2.2 Pasos de la Vista de Despliegue

- Caracterización de la solución vista desde las licencias de los componentes que la forman.

- Caracterización de la solución vista desde las estrategias de desarrollo empleadas para su elaboración.
- Definición del tipo de licenciamiento de la solución vista desde la arquitectura.
- Definición de las estrategias de ingreso vista desde la arquitectura.
- Definición de la estrategia de empaquetamiento del producto para la implantación.
- Declaración de los requerimientos mínimos de instalación.
- Declaración de la estrategia para el Soporte.
- Definición de la estrategia para la actualización de la solución (36).

2.3 Caracterización de la solución vista desde las licencias de los componentes que la forman.

GESPRO utiliza herramientas con licencias del tipo permisivas y no permisivas. En las siguientes tablas se hace una selección de las mismas.

- **(Permisiva) Licencias de software de código abierto permisivas**

	Academic Free License v.1.2.		Zope Public License v.2.0
	Apache Software License v.1.1.		Open LDAP License v.2.7
	Artistic License v.2.0		Perl License.
	Attribution Assurance license.		Academic Free License v.3.0
✓	BSD License.		Python License v.2.1
✓	MIT License.		PHP License v.3.0
✓	Apache Software License v.2.0.		Q Public License v.1.0
	University of Illinois/NCSA Open Source License.		
	W3C Software Notice and License.		

- **(No Permisiva) Licencias de software de código abierto robustas fuertes o con copyleft fuerte.**

	Common Public License v.1.0.		ECos License v.2.0.
✓	GNU General Public License v.1.0.		Sleepycat Software Product License.
✓	GNU General Public License v.2.0.		Affero License v.1.0.
✓	GNU General Public License v.3.0.		Affero License v.2.0.
✓	ZPL Zimbra Public License.		OpenSSL License
	Eclipse Public License.		

- **(No Permisiva) Licencias de software de código abierto robustas débiles, con copyleft débil/suave o híbridas**.

✓	GNU Lesser General Public License v.1.1.		Apple Source License v.2.0
	GNU Lesser General Public License v.2.1.		CDDL.

	Mozilla Public License.		EUPL
	Open Source License.		

2.3.1 Herramientas que componen el producto

En la siguiente tabla se establece la relación entre las herramientas que componen el producto, el tipo de licencia de las mismas, así como su clasificación atendiendo a los términos y condiciones que plantean cada una de ellas.

Tabla 1 Tabla de Herramientas que componen el producto.

Herramientas	Licencia	Permisiva	No permisiva	Propietaria
Sistema Operativo				
Ubuntu Server 10.04	GPL		X	
Herramientas que soportan los servidores web				
Apache 2	Apache 2.0	X		
Phusion Passenger	Apache 2.0	X		
TomCat 6.0	Apache 2.0	X		
Framework Ruby on Rails v2.3.11	MIT	X		

Máquina virtual Java 6.0	GPL		X	
Gestor de bases de datos y herramientas que soportan la conexión a las bases de datos (ORM)				
PostgreSQL 9.0.1	GPL		X	
PgPool 3.0.2	BSD	X		
Herramientas que soportan la Gestión Documental y del código fuente				
Alfresco v3.0	LGPL		X	
Excriba v2.0	GPL		X	
Herramientas para el control de versiones				
Subversion v1.6	GPL		X	
Herramientas para la generación dinámica de reportes y el análisis estadístico				
PATDSI-CHART Server	GPL		X	
PATDSI-Generador de Reportes	GPL		X	
PATDSI-RServer	GPL		X	
Herramientas para el monitoreo y la administración del entorno				
Munin v1.4.4	GPL 2.0		X	
Bacula Enterprise Edition v5.0.1	GPL 2.0		X	
Awstats	GPL 3.0		X	
Herramienta de correo y mensajería				
Zimbra	ZPL		X	

Herramientas para el teletrabajo y el trabajo colaborativo				
Caxtor v1.0	GPL		X	
Orion v1.0	GPL		X	
Herramientas para la autenticación y seguridad y protección del código				
Acaxia 2.0.1	GPL		X	
Odec v1.0	GPL		X	
Herramientas para el control de las tareas y proyectos (DIP) gestión de recursos				
Redmine v1.1.2	GPL		X	

2.4 Caracterización de la solución vista desde las estrategias de desarrollo empleadas para su elaboración

Después de analizar las posibles estrategias de desarrollo se selecciona la más adecuada en la elaboración del producto.

	Desarrollo de código abierto único proveedor
	Desarrollo comunitario de código abierto
	Desarrollo mixto de código abierto
	Desarrollo completamente cerrado
✓	**Desarrollo híbrido**

✓	**Proveedores híbridos**

	Comunidad hibrida
	Híbrido mixto

Se selecciona como estrategia el desarrollo híbrido, específicamente la categoría proveedores híbridos atendiendo a que GESPRO es distribuido usando licencia de código abierto GPL y al mismo tiempo algunas funcionalidades contienen conocimiento propio del grupo de desarrollo y no se entrega al cliente. Dentro de estas funcionalidades se encuentran los algoritmos para el cálculo de indicadores que abarcan seis áreas del conocimiento de Gestión de Proyectos, basados en técnicas de *softcomputing* (34).

2.5 Definición del tipo de licenciamiento de la solución vista desde la arquitectura

En la siguiente tabla se hace selección, dentro de las posibles variantes, del tipo de licenciamiento de la solución, de acuerdo a las políticas de la entidad.

	Licenciamiento dual
	Licenciamiento Núcleo-Abierto
	Licenciamiento Abierto y Cerrado
✓	**Código abierto**
	Código abierto ensamblado
	Cerrado

Se selecciona el tipo de licenciamiento de código abierto debido a que GESPRO es desarrollado con lenguajes de programación y otras herramientas de software con licencias de código abierto.

2.6 Definición de las estrategias de ingreso vista desde la arquitectura

En la siguiente tabla se seleccionan las estrategias de ingreso a emplear y se enumeran de acuerdo con el orden de prevalencia de las mismas.

	Licenciamiento comercial
2	**Subscripciones**
3	**Servicios y soporte**
	Hardware empotrado
	Software empotrado
4	**Software como servicio (SaaS)**
	Publicidad
	Desarrollo a la medida
1	**Otros productos y servicios**

El modelo de negocio propuesto para GESPRO, atendiendo a la selección de las estrategias de ingresos, es un modelo basado en servicios:

- Servicio de personalización de la Suite GESPRO, bajo licencia GNU GPL v2.0, para el control y seguimiento de proyectos y la toma de decisiones en los siguientes tres niveles: nivel de proyecto, el nivel de entidad ejecutora y nivel gerencial de la corporación.
- Servicio para la configuración y montaje de la plataforma en entorno de alta disponibilidad y rendimiento en Centros Datos, facilidades para el monitoreo y la salva de seguridad.
- Servicio de despliegue de la plataforma por anillos permitiendo la asimilación de la plataforma y migración de los datos de los proyectos existentes para la nueva plataforma.
- Servicio de montaje y personalización de programa para la formación continua y autónoma en gestión de proyectos.
- Servicio de suscripciones a actualizaciones automáticas de extensiones a la Suite GESPRO y servicios de soporte técnico a tres niveles.

- Servicio de consultoría y montaje de proyectos de desarrollo (35).

2.7 Estrategia de empaquetamiento para implantación y actualización

En la definición de la estrategia de empaquetamiento para implantación y actualización del producto se han tenido en cuenta los activos reutilizables requeridos para el entorno de desarrollo (gemas), los activos reutilizables en forma de plugins (plugins) y los activos compuestos que forman parte del producto final.

2.7.1 Activos reutilizables requeridos para el entorno de desarrollo

Tabla 2 Tabla de Gemas.

Gemas	
actionmailer (2.3.11)	Servicio de capa para la entrega de correo electrónico fácil y pruebas.
actionpack (2.3.11)	Paquete utilizado por las aplicaciones web desarrolladas en Ruby on Rails sobre el patrón arquitectónico Modelo Vista Controlador.
activerecord (2.3.11)	Paquete utilizado para el mapeo de la base de datos de la plataforma GESPRO.
activeresource (2.3.11)	Paquete que complementa el active record.
activesupport (2.3.11)	Soporte para el trabajo con la internacionalización, zonas horarias y las pruebas.
capistrano (2.6.0)	Herramienta que facilita el despliegue de la plataforma desde un repositorio de desarrollo.
highline (1.6.1)	Paquete que facilita la conversión de tipos de datos y el trabajo con listas.

i18n (0.4.2)	Conjunto de internacionalizaciones.
json (1.4.6)	Implementación de Json para ruby.
linecache (0.43)	Paquete para leer y debuguear líneas de código.
net-scp (1.0.4)	Implementación para utilizar el protocol scp.
net-sftp (2.0.5)	Implementación para utilizar el protocol sftp.
redmine_client (0.0.1)-	Publicar modelos de la aplicación a través de la URL.
resque (1.15.0)	Paquete para crear, consultar y procesar el trabajo del framework.
ruby-debug-ide (0.4.6)	Librería para debuguear.
systemTimer (1.2.1)	Librería para el trabajo con señales.
net-ssh (2.1.4)	Implementación para el trabajo con el protocolo ssh.
net-ssh-gateway (1.1.0)	Librería simple para crear túneles para la conexión por ssh.
nokogiri (1.4.4)	Paquete para el trabajo con ficheros en formatos XML, HTML y SAX.
rack (1.1.2)	Interfaz modular y adaptable para el desarrollo de aplicaciones en Ruby on Rails.
rails (2.3.11)	Paquetes utilizados por el framework Ruby on Rails y las

	aplicaciones que sustentan el lenguaje.
rake (0.8.7)	Paquete utilizado para el trabajo con tareas en Ruby on Rails.
redis (2.1.1)	Librería para el trabajo con el servidor de notificaciones.
redis-namespace (0.10.0)	Implementación que se utiliza como interfaz para el trabajo con redis.
sinatra (1.1.4)	Paquete para el trabajo con líneas digitales.

2.7.2 Activos reutilizables en forma de plugins

Tabla 3 Tabla de Plugins

Plugins	Descripción preliminar
GESPRO Auto-Progreso	Automatiza el progreso de las tareas pone a 100% el avance cuando cambia a estados terminados.
GESPRO Inicializador	Inicializa varias características internas para GESPRO Plugins.
GESPRO Identificador único	Genera un identificador único para el proyecto con formato previamente definido.
GESPRO Mejora de Rendimiento	Crea capacidades y elimina los datos inconsistentes en GESPRO.
GESPRO Ejecutar script sql	Ejecuta scripts sql en el servidor de base de datos desde el servidor de aplicaciones.
GESPRO Validaciones	Introduce algunas reglas estandarizadas para el completo desarrollo de GESPRO.

Gespro Blogs	Enlaza los blogs en la parte superior del menú.
GESPRO Gestión Documental	Integra los sistemas de Gestión Documental con el Redmine
GESPRO Links	Permite configurar varios links para herramientas.
GESPRO Importador de Costo	Importa costos al Redmine. (Basadado en el plugin Importer v0.4 de Martin Liu).
GESPRO Importador de tareas	Importa peticiones al Redmine. (Basado en el plugin Importer v0.4 de Martin Liu).
GESPRO Integrador de importadores.	Integra en una vista los plugins de importar datos a nivel de proyecto.
GESPRO Cargador	Lee los archivos básicos del proyecto (Basado en el redmine_loader v0.0.11).
GESPRO Organizador de menús	Reorganiza GESPRO.
GESPRO Importador de recursos	Importa recursos al Redmine. (Basadado en el plugin Importer v0.4 de Martin Liu).
UCI Redmine barra lateral	Facilita la inserción de contenido en la barra lateral.
UCI Redmine noticias	Adiciona noticias en la barra lateral
GESPRO Lineas-bases	Interfaz para la gestión de problemas en las líneas de base

Redmine Iteraciones de los proyectos	Este es un plugin para Redmine que se utiliza para mostrar más información dentro de la página roadmap y pone en práctica los milestones que ofrecen.
GESPRO Notificaciones	Hace que ciertas modificaciones se notifiquen en tiempo real a los usuarios.
GESPRO Reportes	Integra el servicio de reportes de PATDSI en el Redmine.
GESPRO Panel de control	Se utiliza para la toma de decisiones.
GESPRO Indicadores de proyectos	Gestiona los macro indicadores de los proyectos.
GESPRO Sincronización de proyectos	Copia ante-proyectos, hitos de ejecución del ante-proyecto y crea la línea base de los proyectos.
GESPRO Gerencial	Permite darle seguimiento a los proyectos desde el GESPRO Gerencial.
GESPRO Asistencia	Controla la asistencia.
GESPRO Encuestas	Permite insertar encuestas.
GESPRO Interesados	Para el Redmine.
GESPRO Requerimientos	Plugin de requisitos.
GESPRO Recuersos	Plugin para la gestión de recursos.
GESPRO Taza de recursos	El plugin Rate proporciona una API que puede ser utilizada para

humanos	encontrar el ¨Rate¨ de un miembro de un proyecto en una fecha específica. También almacena los ¨Rate¨ históricos para mantener los cálculos.
GESPRO Alcance	Crea una interfaz ExtJS para la gestión de las tareas de forma colaborativa.
GESPRO Riesgos	Plugin para la gestión de riesgos
GESPRO Configuración de Despliegue	Configura los campos personalizados, reportes e indicadores en dependencia del escenario en que se ejecute el despliegue.

2.7.3 Activos compuestos que forman parte del producto final

Tabla 4 Tabla de activos compuestos.

Nombre	Breve descripción	Formato en que se entrega.	Soporte en que se entrega.
Manual de Usuario de la solución.	Manual de Usuario Personalizado.	PDF	DVD
Ayuda Online de la solución.	Ayuda montada sobre la Web y accesible desde la ayuda del sistema.	-	Online
Montaje del sistema actualizaciones.	Sistema de actualizaciones basado en Capistrano.	Ejecutable	Instalado
Paquete de plugins con extensiones por cada actualización.	Paquetes de plugins que conforman la actualización.	PDF	DVD

Paquete de reportes con extensiones por cada actualización.	Paquete de funciones. Paquete de reportes.	PDF	DVD
Servidor Gerencial configurado en el Centro de datos.	Servidor Gerencial Montado en Centro de Datos.	Ejecutable	Instalado
Servidores Desarrollo configurados en el Centro de Datos.	Este servidor montado depende de las necesidades reales de los usuarios según el diseño especificado.	Ejecutable	Instalado
Infraestructura para el monitoreo configurada en el Centro de Datos.	Este sistema de monitoreo basado en MUNIN.	Ejecutable	Instalado
Infraestructura para la Salva de Seguridad configurada en el Centro de Datos.	Este sistema de Salva y seguridad basado en Bacula.	Ejecutable	Instalado
Paquete para el Despliegue.	Paquete para la transferencia que incluye: • Paquete de conferencias para la transferencia • Paquete de sistemas para el despliegue	PDF	DVD

	Manuales de instalación y configuración para el despliegue.		
Plataforma GESPRO Personalizada, Desplegada por anillos.	Plataforma de desarrollo desplegada en las entidades de cada anillo.	Ejecutable	Instalado

2.8 Conclusiones

Durante la realización de este capítulo se logró desarrollar una parte de la propuesta de solución a la Vista de Despliegue de la Arquitectura del Sistema GESPRO 12.05. Para ello se comenzó caracterizando la solución vista desde las licencias de los componentes que forman el producto. Se definieron además, el tipo de licenciamiento de la solución, así como la definición de las estrategias de ingreso y empaquetamiento del producto.

Capítulo 3: Despliegue y soporte de GESPRO. Validación de resultados

Introducción

En el presente capítulo se finaliza la propuesta de solución de la Vista de Despliegue, donde se definirán los requerimientos mínimos de instalación, así como las estrategias necesarias para el soporte y la actualización de la solución. Además se validará la propuesta poniendo en práctica el método de evaluación preexperimento, para arribar a resultados del nivel de factibilidad existente en la aplicación de la misma.

3.1 Requerimientos mínimos de instalación

En las siguientes tablas se plasman los requerimientos mínimos de hardware y software para la instalación del producto.

Tabla 5 Tabla de requerimientos mínimos de Hardware

Hardware (Servidor de aplicación)	
Cantidad	3
CPU	4 x 2.33 GHz (Intel Xeon 5140 Core2 2.33 GHz)
RAM	2 Gb
HDD	250 Gb RAID 5
LAN	2 x NIC (1 Gbit)
SAN	-
Hardware (Servidor de Base de Datos)	
Cantidad	2
CPU	4 x 2.33 GHz (Intel Xeon 5140 Core2 2.33 GHz)
RAM	2 Gb
HDD	250 Gb RAID 5
LAN	2 x NIC (1 Gbit)

SAN	1 Tb

Tabla 6 Tabla de requerimientos mínimos de Software

Software (Servidor de Aplicación)	
Sistema operativo	Ubuntu Server 10.04 LTS, o LTS Superior.
Servidor de aplicación	Apache2.
Módulos básicos para el despliegue de Redmine	Ruby1.8, ri1.8, libpgsql-ruby1.8, ruby1.8-dev, librmagick-ruby1.8, libopenssl-ruby1.8, build-essential, apache2-threaded-dev, rake, libxslt-dev, libxml2-dev.
Aplicación base	Redmine 1.2.
Framework de Desarrollo	Ruby on Rails.
Módulos de PHP	php5, libapache2-mod-php5, php5-cli, php5-gd, php5-mysql, php5-pgsql, php5-sqlite, php5-xsl.
Componentes de PATDSI-Generador de Reportes	Módulo RServer, Módulo ChartServer, Módulo rjson.
Control de versiones	Subversion 2.6.
Gestor Documental	eXcriba 2.0/Alfresco.
Módulo de Información Geográfica	Módulo GeneSIG.
Software (Servidor de almacenamiento de información)	
Gestor de Base de Datos	PostgreSQL versión 9.0.

Sistema de Alta Disponibilidad y Balance de Carga	pgpool-II 3.0.1, Hot Standby, Streaming Replication y Heartbeat.

3.2 Estrategia para el soporte

En la siguiente tabla se propone la estrategia para el soporte de la solución, a través de la definición de cómo se prevé garantizar los tres niveles de soporte de los productos del proyecto y el tiempo previsto de duración del mismo.

Tabla 7 Tabla de niveles de soporte

Nombre	Nivel 1 de soporte	Nivel 2 de soporte	Nivel 3 de soporte	Tiempo previsto de soporte
GESPRO	Lo provee el centro de soporte y soportado desde dos escenarios. Escenario apoyo telefónico y el escenario de sistema para el reporte y el seguimiento de incidencias.	Lo provee el grupo de asesoría técnica de la dirección técnica que además coordina cursos avanzados de gestión de proyectos y herramientas para la gestión de proyectos. Este grupo constituye la Entidad 4 de la Línea de Productos de Software establecida.	Lo provee el equipo que desarrolla GESPRO quien constituye la entidad 1, encargada de la ingeniería del dominio y básicamente corrige las no conformidades en los activos desarrollados.	Sólo por 1 año.

3.2.1 Detalles de soporte nivel 3

El escenario de soporte que propone GESPRO se basa en los siguientes principios (35):

1. El Laboratorio de Gestión de Proyectos brinda soporte nivel 3.

2. El laboratorio de Gestión de Proyectos tiene un servicio de transferencia para el soporte Nivel 1 y Nivel 2 para la empresa cliente.

- **Inicio del servicio:** a partir de que se comience la explotación del sistema con el cliente final y después de las pruebas de aceptación.

- **Duración del servicio:** 180 días a partir del comienzo del servicio.

- **Ámbito de trabajo**

  - ✓ Recepción de las incidencias a través de correo electrónico.
  - ✓ Para el caso de incidencias de mayor impacto en el sistema que constituyan cambios se deberá realizar una reunión de coordinación para realizar un plan de resolución de las mismas entre el equipo de soporte Nivel 1 y 2, el cliente y el Laboratorio de gestión de proyectos para el Soporte Nivel 3.
  - ✓ Disponibilidad del servicio: El servicio estará disponible en los horarios que se especifican a continuación:

Tabla 8 Horarios de servicio

Servicio	Período	Horario
Recepción de incidencias por correo electrónico	Lunes – Viernes	9:00am a 4:30pm

Tabla 9 Medios de acceso al servicio

Medio	Acceso
Correo Electrónico	maestriagp@uci.cu

Lugar de prestación del servicio: Laboratorio de Gestión de proyectos.

- ✓ Modificaciones del servicio: Cualquiera de Las Partes podrá proponer modificaciones en el ámbito, naturaleza, disponibilidad, tiempos de respuesta, etcétera. De mutuo acuerdo, Las Partes deberán aprobar los cambios propuestos.

- **Rendimiento y monitorización del servicio**

✓ Medidas de referencia, objetivos y métricas a utilizar: Reportes Básicos que se brindarán al Centro de Desarrollo para monitorear la calidad del servicio:

- Cantidad de incidencias registradas en un período de 30 días.
- Cantidad de incidencias resueltas por técnicos de soporte en un período de 30 días.
- Cantidad de incidencias pendientes en un período de 30 días.
- Cantidad de incidencias resueltas en menos del número de días determinado.

✓ Reuniones de revisión del servicio: Las reuniones de revisión del servicio serán trimestralmente, los aspectos a tratar en las mismas serán acordados entre las partes. Para los casos de emergencias, afectaciones o incumplimientos en el servicio se realizará una reunión extraordinaria para resolver lo ocurrido.

- **Gestión de incidencias**

✓ Incidencia: Se considera una incidencia de Soporte Nivel 3 a aquellas que afectan el correcto funcionamiento del sistema y su respuesta está relacionada con la corrección o cambio en el código.

- Debe tener una descripción del error.
- Secuencia de pasos que realizó el usuario cuando ocurrió el error.
- Juego de datos para reproducir el error.

✓ Tiempos de respuesta: El laboratorio de Gestión de proyectos tiene cuatro días hábiles para estudiar el impacto de la incidencia y proponer un plan para su resolución. La resolución estará ajustada en el plan de resolución.

✓ Finalización en los términos acordados: La finalización del acuerdo se realiza según lo establecido en la “Fecha inicio” y “Duración del acuerdo”.

3.3 Estrategia para el despliegue y actualización de la solución

En la siguiente tabla se hace una selección, dentro de las posibles estrategias, de cuál es la más adecuada para garantizar el despliegue de GESPRO.

Tabla 10 Selección de estrategia para garantizar el despliegue.

	Despliegue se hace en cada una de la estaciones de trabajo de los clientes e implica un despliegue trasladándose hasta los emplazamientos de los clientes.
✓	Se hace en un servidor central y no es necesario trasladarse hasta cada uno de los emplazamientos de los clientes.

En la siguiente tabla se hace una selección, dentro de las posibles estrategias, de cuál es la más adecuada para garantizar la actualización de GESPRO.

Tabla 11 Selección de estrategia para garantizar la actualización.

	Actualización de la aplicación es automática, las aplicaciones en las estaciones de trabajo de los usuarios finales se conectan a un servidor central y se actualizan.
	Actualización de la aplicación es manual y se debe repetir el proceso de despliegue para cada una de las actualizaciones.
	El sistema de actualizaciones está intrínsecamente integrado al sistema operativo lo que permite que se actualicen las aplicaciones con la actualización del propio sistema operativo.
✓	La actualización de la aplicación es semiautomática, asistida por herramientas informáticas que garantizan la instalación a partir de un repositorio. Está intrínsecamente integrado a la plataforma de desarrollo.

A continuación se hace una representación de cómo debe llevarse a cabo, según la estrategia seleccionada, la actualización de los diferentes centros de desarrollo.

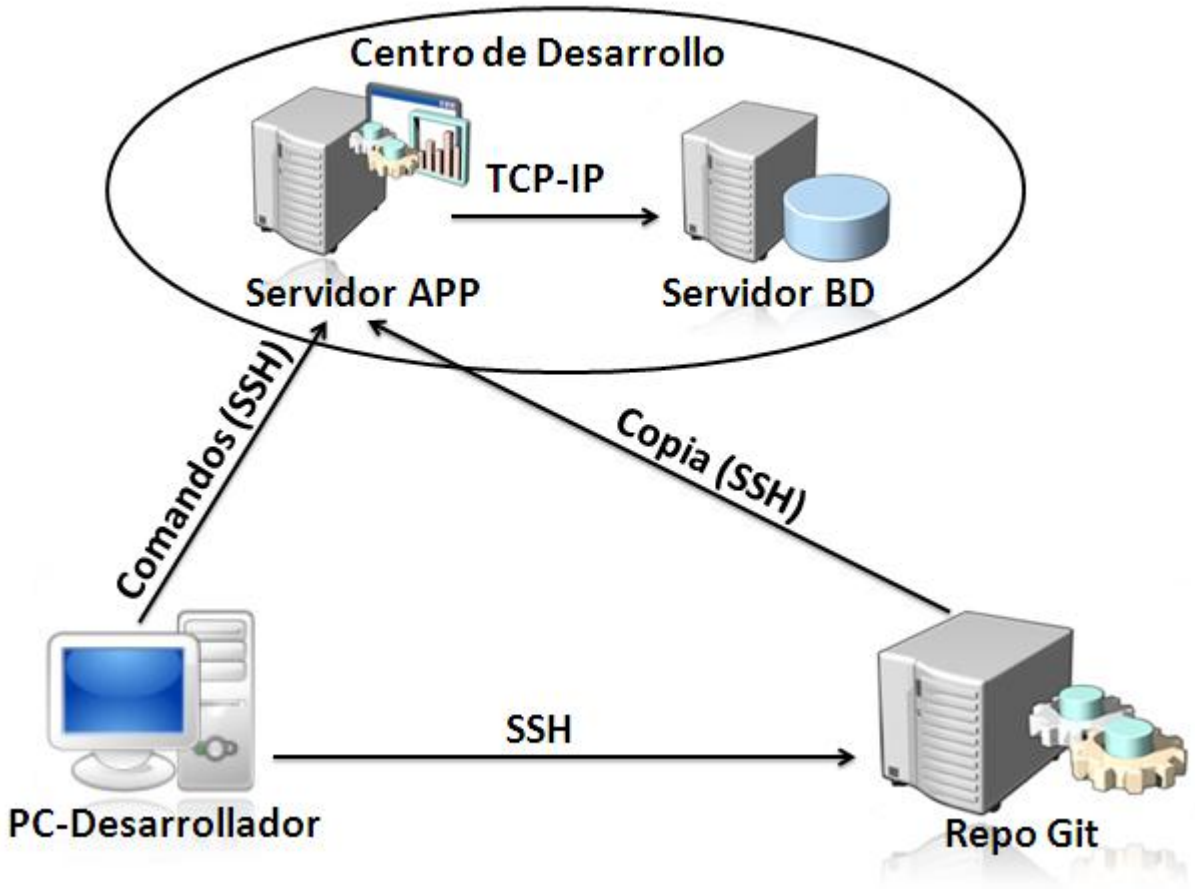

Fig. 2 Proceso de actualización de GESPRO.

Actualmente la actualización de la plataforma de gestión de proyectos GESPRO se realiza mediante la herramienta Capistrano en su versión 2.6.0. El mismo contiene un conjunto de ficheros de configuración donde se especifican los servidores en los cuáles se realizará el despliegue o actualización.

Capistrano posee además un sinnúmero de funcionalidades programadas en forma de tareas que facilitan la automatización de un proceso que anteriormente y de forma manual duraba 40 minutos por servidor, aproximadamente 56 horas hombres para los 22 servidores. Dichas funcionalidades comprenden actualización del código fuente de la plataforma, migraciones en la base de datos así como la ejecución de tareas de Ruby on Rails, funciones SQL del lado del servidor y el reinicio del servidor de aplicaciones. Hoy en un tiempo que rara vez supera 1 hora hombre se actualizan 22 servidores de la Red de Centros de la Universidad de las Ciencias Informáticas dejando atrás el tedioso procedimiento manual de conectarse a cada servidor, actualizar el código de la aplicación, dar los permisos y reiniciar el servidor web.

La herramienta de actualización y despliegue además se encuentra integrada a un repositorio *git* permitiendo que el despliegue se realice a partir de la integración del trabajo de todos los desarrolladores. Esto minimiza los tiempos de integración y despliegue hacia un entorno de pruebas en el cual se prueban las funcionalidades incorporadas permitiendo luego la actualización a la red de centros del mismo código en todos los servidores.

El Capistrano respetando los patrones arquitectónicos que implementa GESPRO define una estructura de carpeta de forma tal que se actualicen aquellas porciones del código que se ejecutan y dejando en un directorio compartido ficheros de configuración de la aplicación así como los ficheros subidos a la plataforma por partes de los usuarios. La Fig 3 refleja la estructura de carpetas que define Capistrano para sus despliegues en el directorio raíz de la aplicación.

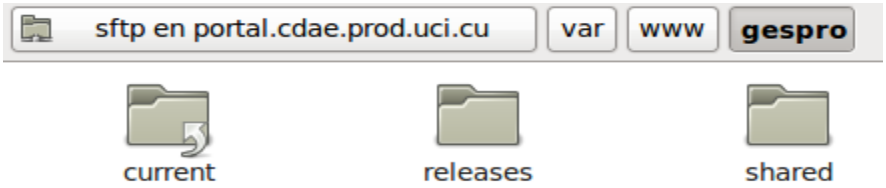

Fig. 3 Estructura de carpetas en el directorio raíz de la aplicación.

El fichero *current* representa un enlace al *release* que contiene la última versión del código fuente. La carpeta *shared* contiene los ficheros del directorio compartido descrito en el párrafo anterior y su contenido es mostrado en la Fig 4.

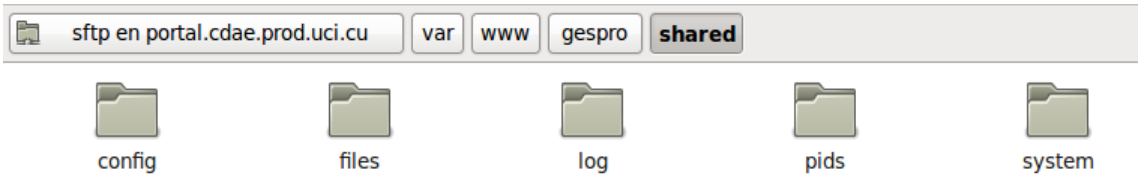

Fig. 4 Contenido del directorio *shared.*

Por su parte la carpeta *releases* contiene las diferentes versiones del código como se muestra en la Fig 5.

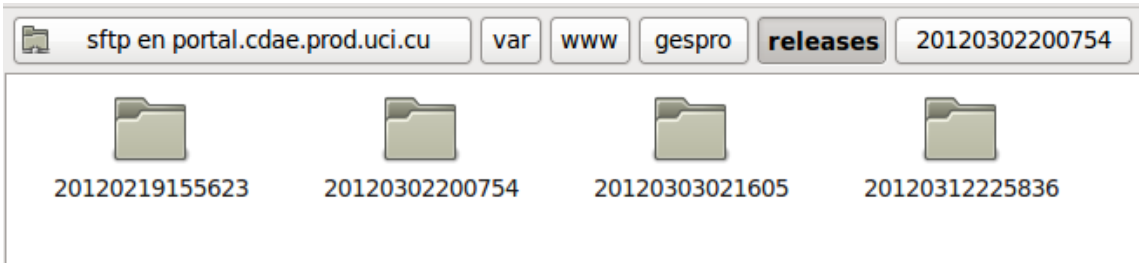

Fig. 5 Contenido del directorio *releases*.

Para no ocupar demasiado espacio en el disco duro del servidor en versiones del código de la aplicación el Capistrano brinda una función que permite la eliminación de los *releases* más antiguos dejando solamente los cinco que representan las últimas actualizaciones. En caso de haber ocurrido algún error durante el proceso de despliegue o sencillamente se haya desplegado alguna versión defectuosa del código se cuenta con una función que permite regresar la aplicación a su versión de código anterior. Se cuenta además con una consola que permite ejecutar diferentes comandos en los servidores al mismo tiempo.

Todas las funciones para la actualización son ejecutadas desde el terminal de Ubuntu de una máquina de escritorio perteneciente al Laboratorio de Soluciones e Investigaciones Avanzadas en Gestión de Proyectos con dirección de Protocolo de Internet estática y cuya llave de conexión por SSH para el usuario en cuya sesión se despliega ha sido copiada con anterioridad a cada servidor.

A través de esta herramienta se actualizan además los portales de GESPRO que se encuentran en los Centros de Desarrollo de Software de Holguín y Villa Clara, además de las tres Facultades Regionales de la Universidad de las Ciencias Informáticas.

3.4 Validación de la propuesta

Tipos de evaluación

Se analizaron varios tipos de evaluaciones que existen, para escoger los más factibles y aplicarlos a la propuesta.

Método de consulta a expertos. Método Delphi:

- Este método desde sus inicios en los años 50 ha sido utilizado frecuentemente como sistema para obtener información sobre las ocurrencias de un fenómeno en el futuro. Consiste en la selección de un grupo de expertos a los que se les encuesta su opinión sobre cuestiones referidas a sucesos del futuro. El método se basa en la utilización

sistemática de un juicio intuitivo emitido por un grupo de expertos, que se obtiene encuestando a este grupo mediante un cuestionario. Es un método fiable y muy utilizado actualmente pero necesita del estudio de competencias en los participantes del panel. (referenciar esto)

- No es aplicable a la propuesta debido a la poca existencia de personal con experiencia en el tema en la UCI.

Grupo focal:

- Básicamente es la selección de un grupo de personas con conocimientos sobre el tema, deben ser especialistas, expertos, de distintos niveles y categorías, que se reúnen en un lugar a una hora determinada, donde se discute en forma de grupo debate sobre el procedimiento, siendo este debate dirigido por los autores, y centrado en lo que se quiere conocer sobre el procedimiento.
- Imposibilidad de realizar esta actividad por falta de personal.

Triangulación:

- La triangulación entendida como técnica de confrontación y herramienta de comparación de diferentes tipos de análisis de datos (triangulación analítica) con un mismo objetivo puede contribuir a validar un estudio de encuesta y potenciar las conclusiones que de él se derivan. Se hace a través de datos estadísticos recogidos sobre resultados de la puesta en práctica.

- Es imposible de aplicar pues los datos deben ser tomados, recogiendo estados de opinión de los involucrados en la propuesta, cosa que tampoco se hace posible por falta de tiempo.

Preexperimento:

- Los llamados preexperimentos son aquellos en los cuales no existe un grupo de control (patrón o testigo) para comparar. Por tanto, no hay, o se reducen las posibilidades de manipular las variables independientes y las conclusiones son extraídas en el mejor de los casos por la variación de la variable dependiente en relación con su historia anterior.

- Este fue el seleccionado para la validación por las ventajas y oportunidades que ofrece. con respecto a la variación que ha sufrido la variable dependiente desde la aplicación de la propuesta, esto se evidencia en la tabla que se ofrece a continuación.

3.4.1 Validación por preexperimento.

En la siguiente tabla se hace una comparación del proceso de despliegue y configuración de la plataforma GESPRO antes y después de la aplicación de la propuesta. Se utilizan variables medibles para arribar a conclusiones sobre el impacto de la aplicación de la misma.

Tabla 12 Validación de la Propuesta de Despliegue de la Arquitectura de GESPRO

Marco de tiempo	**Antes**	**Ahora**	
Método	Manual	Semi-automático	Manual
Escenario que aplica	UCI UH	UCI FR-Granma FR-Atermisa FR-Ciego Ávila CDS-DESOFT-Villa Clara CDS- DESOFT-Holguín CDS- DESOFT-Santiago de Cuba	DTS-MININT DESOFT-Habana
Tiempo Despliegue	El despliegue duraba como promedio dos horas hombre por servidor. **Total:** 56 horas hombres para 22	Un despliegue completo requiere en **total:** 1 hora hombre para 22 servidores.	Requiere en **total:** 0.2 horas hombre.

	servidores.		
Tiempo de Configuración	La configuración duraba tres horas hombre por servidor. **Total:** 66 horas hombre para 22 servidores.	La configuración de los flujos de trabajo y permisos requiere en **total:** 0.16 horas hombre por servidor. **Total:** 3.52 horas hombre para 22 servidores.	La configuración de los flujos de trabajo y permisos requiere en **total**: 0.16 horas hombre por servidor.
Complejidad	**Alta:** Existía una guía con un orden estricto a seguir. Cualquier equivocación del especialista podía incidir de manera negativa en el despliegue y aumentar la duración del mismo.	**Baja:** La herramienta Capistrano para realizar la actualización cuenta con todas las instrucciones necesarias a ejecutar. Sólo se debe especificar la dirección de los servidores para los cuáles se va a desplegar, y este paso sólo debe ser ejecutado una sola vez. La plataforma GESPRO cuenta con un módulo de configuración avanzada para la gestión de campos personalizados, configuración de reportes e indicadores.	**Media**: Sobrescribir el código de la aplicación. Ejecutar migrates (ficheros que realizan cambios en la BD) en caso de que sean necesarios para realizar cambios en la base de datos. Se reinicia el servidor de aplicaciones. La plataforma GESPRO cuenta con un módulo de configuración avanzada para la gestión de campos personalizados, configuración de reportes e indicadores.
Versión del código	Era difícil tener la	El Capistrano se conecta	La actualización del

fuente	certeza de cuál versión del código se estaba desplegando. En una actualización ejecutada por más de un especialista no siempre era posible determinar si cada uno desplegaba el mismo código fuente. Incluso cuando se sabía que todos tenían la misma versión era difícil demostrar que se trataba de la más actualizada.	al repositorio que contiene el código fuente de la plataforma GESPRO, el cuál es único y se encuentra actualizado. Esto garantiza el despliegue de una versión única para todos los servidores.	código fuente se realiza a partir descarga previa del repositorio lo que garantiza una versión única para estos escenarios.

3.4.2 Gráfico de los tiempos de despliegue y configuración de la Plataforma GESPRO antes y después de la implantación de la propuesta

En el siguiente gráfico se muestra la disminución de los tiempos de despliegue y configuración de la Plataforma GESPRO para una muestra de 22 servidores.

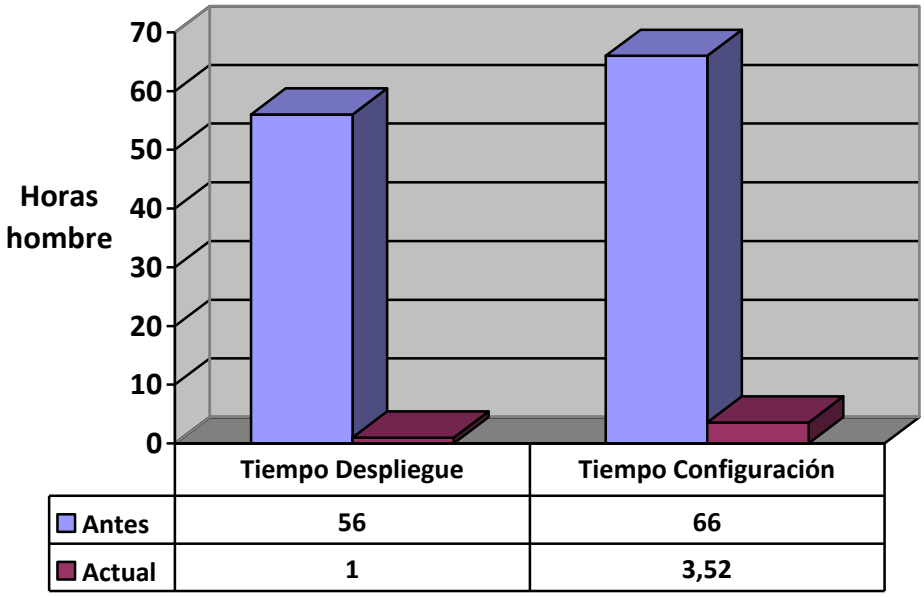

Fig. 6 Gráfico de los tiempos de despliegue y configuración.

3.5 Conclusiones

Con la realización de este capítulo se culminó la propuesta de solución de la Vista de Despliegue de la Arquitectura del Sistema GESPRO 12.05. Para ello se definieron los requerimientos mínimos para la instalación del producto, así como las estrategias necesarias para el soporte y la actualización del mismo. Se logró además validar la propuesta demostrándose la importancia de la aplicación de la misma con la obtención de los resultados que prueban la factibilidad de su puesta en práctica.

Conclusiones Generales

La investigación desarrollada con la realización de este trabajo de diploma y el análisis de la documentación estudiada permitió el cumplimiento de los objetivos trazados. Luego del extenso estudio realizado y las razones expuestas en el contenido, se logró establecer un marco teórico para la Arquitectura de Software en la actualidad.

Se desarrolló la propuesta de solución, que permitió definir e implantar la Vista de Despliegue de la Arquitectura del Sistema GESPRO. Con la propuesta se logró caracterizar la solución vista desde las licencias de los componentes que la forman, y desde las estrategias de desarrollo; de ingresos y de empaquetamiento del producto. Se definió además el tipo de licenciamiento, las estrategias para el soporte y actualización de la solución, y se establecieron además los requerimientos mínimos para la implantación.

Con la aplicación de la propuesta fue posible demostrar la funcionalidad de la misma pues se logró disminuir los tiempos de despliegue, actualización y configuración del Sistema, garantizándose así el cumplimiento de los objetivos trazados para el presente trabajo de diploma.

"El mayor beneficio que brinda la evaluación de una arquitectura, es que descubre los problemas que si se hubiesen dejado sin descubrir, habría sido mucho, más costoso corregirlos luego. En breve, una evaluación produce una mejor arquitectura" (31).

Recomendaciones

Se recomienda, apoyado en el estudio realizado en este trabajo:

- Refinar las características metodológicas de los resultados presentados en el trabajo.

- Emplear el resultado presentado en el trabajo para el aprendizaje y/o enseñanza de la disciplina de la Arquitectura de Software.

- Se recomienda que el resultado propuesto en la investigación sea implantado como uso cotidiano y sea introducido en todos los Sistemas de Gestión de la universidad.

- Se recomienda continuar la investigación sobre el resultado obtenido y proponerlo como base referencial o material auxiliar en el desarrollo de futuros trabajos asociados al objeto de estudio de la Arquitectura de Software.

- Se recomienda la realización de un paquete de instalación que contenga los componentes de software que conforman la Plataforma GESPRO cumpliendo con los fundamentos teóricos de la presente investigación.

Referencias Bibliográficas

1. **Sandor, Hasznos.** *The Software Deployment Mystery .* s.l. : IBM RedBooks , 2004.

2. **Reynoso, Billy Reynoso: Billy.** 26 de 6 de 2006. [Citado el: 28 de 12 de 2007.].

3. **Shaw, Mary y Clements, Paul.** *A field guide to Boxology: Preliminary classification of architectural styles for software systems. Computer Science Department and Software Engineering Institute.* s.l. : Carnegie Mellon University, 1996.

4. **Osvaldo Díaz Verdecia, Virgen Damaris Quevedo.** *Tesis de Grado: "Una guía práctica de Arquitectura de Software".* 2009.

5. **Jacobson, Ivar, Grady, Booch y Rumbaugh, James.** *The UML Modeling Language User Guide.* s.l. : Addison-Wesley , 1999.

6. **Pressman, Roger S.** *Ingeniería del Software. Un enfoque prático. .* s.l. : Connecticut , 2002. .

7. **Billy Reynoso, Carlos.** *Introducción a la Arquitectura de Software.* Buenos Aires : Microsoft Press , 2004. .

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

9. **Abdurazik., Aynur.** Suitability of the UML as an Architecture Description Language with applications to testing. *George Mason University.* Febrero de 2000, Reporte ISE-TR-00-01.

10. **Philippe, Kruchten.** The 4+1 View Model of Architecture. IEEE Software, Noviembre de 1995. pp. pp. 42-50.

11. **Lazo, René.** *Tesis de maestria Modelo de referencia para el desarrollo arquitectónico de sistemas de software en dominios de gestión.* 2011.

12. **Clements, Paul.** *A Survey of Architecture Description Languages. Proceedings of the International Workshop on. .* Alemania : s.n., 1996.

13. **Bass, Len, Clements, Paul and Kazman, Rick.** *Software Architecture in Practice .* s.l. : Addison Wesley, 2003. 0-321-15495-9.

14. **Abdurazik, Aynur.** *Suitability of the UML as an Architecture Description Language with applications to testing.* s.l. : George Mason University, 2002.

15. **Grady Booch, James Rumbaugh e Ivar Jacobson.** *El Lenguaje Unificado de Modelado. .* Madrid : Addison-Wesley, 1999.

16. **Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. .** *Design Patterns: Elements of 80 reusable objectoriented. .* s.l. : Addison-Wesley, 1995.

17. **Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad y Michael Stal.** *Pattern-oriented software. A system of patterns. .* s.l. : John Wiley & Sons,, 1996.

18. **R., Pressman.** *Ingeniería del Software. Un enfoque práctico.* s.l. : La Habana Cuba: Félix Varela, 2005. Parte 1.

19. **C., King G.** *Iberneig in Action. .* Estados Unidos de América : Manning Publications, 2005.

20. **Fowler.** *Patterns of Enterprise Application Architecture.* s.l. : Addison Wesley, 2002.

21. **Pérez., Leinier Balmaseda.** *Tesis de Grado: Arquitectura para el Sistema de Registro y Control de Cuadros.* 2008.

22. **Hurtado.., Jorge Luis Naranjo.** *Tesis de Grado: Vista de la arquitectura de sistema y datos del subsistema Capital Humano del proyecto ERP-Cuba.* 2010.

23. **E. Perry, Dewayne y L. Wolf, Alexander.** *Foundations for the study of software architecture. .* s.l. : ACM SIGSOFT Software Engineering Notes, , 1992. . págs. 40–52..

24. **2000, IEEE Std 1471-.** *Recommended Practice for Architectural Description for Software-Intensive Systems.* pág. 3 .

25. **Michael González Jorrín, Pedro Y. Piñero Pérez, René Lazo Ochoa, Nadia Porro Lugo.** *Documento ¨Impacto del uso de los modelos arquitectónicos en el aprendizaje de la arquitectura de software para sistemas de Información¨ .* 2012.

26. **Mulet, Ing. Henry Cruz.** *Tesis de Maestría "Personalización y extensión de un modelo de desarrollo basado en Líneas de Producción de Software para el Centro de Desarrollo Territorial de la UCI en Holguín" .* 2011.

27. **Pupo., Ing. Iliana Pérez.** *Tesis de Maestría "Propuesta de metodología para el diseño e implantación de repositorios de activos de software reutilizables".* 2011.

28. **Montilva., J.A.** *Desarrollo de Software Basado en Líneas de Productos de Software.* Mérida – Venezuela : Universidad de Los Andes. Facultad de Ingeniería. Departamento de Computación. IEEE Computer Society Región 9, 2006.

29. **García., Leonardo Muro.** Caracas, Venezuela : Universidad Nueva Esparta, Facultad de Ciencias, 2007.

30. **Jordan Borjas, E., Enamorado.** *Herramientas para la Gestión de Proyectos en la UCI. .* s.l. : (J. Lugo García, Entrevistador), 2011.

31. **Clements, Paul, Kazman, Rick y Klein, Mark. .** *Evaluating Software Architectures. .* s.l. : SEI. Series in Software Engineering. Adisson Wesley, 2002.

32. **García, Leonardo Muro.** *21 de Noviembre de 2007.* Caracas, Venezuela : Universidad Nueva Esparta, Facultad de Ciencias.

33. **Piñero, Pedro Y.** *Entrevista personal acerca de Evolución de los sistemas de gestión de proyectos en la Universidad de las Ciencias Informáticas,*2010.

34. **Franqueiro, Ernesto A. Mederos.** *Entrevista personal. Laboratorio de GESPRO.* 2012.

35. **Orizondo, Arturo Cesar Arias.** *Ficha para oportunidades de negocios CORPOELEC.* 2012

36. **René Lazo Ochoa, Michael González Jorrín, Pedro Y. Piñero Pérez Manuel Vázquez Acosta, Miguel Sancho Fernández** *16 de octubre de 2010 Arquitectura de Software Vista de despliegue Proyecto GESPRO 12.05*

Glosario de Términos

ADLs: Architecture Description Language (Lenguajes de Descripción Arquitectónica). Lenguaje descriptivo de modelado que se focaliza en la estructura de alto nivel de la aplicación antes que en los detalles de implementación de sus módulos concretos.

API: Interfaz de programación de aplicaciones.

CMM: Modelo de evaluación de los procesos de una organización. La visión general de los modelos basados en la madurez de las capacidades: Modelo de Capacidad y Madurez.

Concerns: Características relacionadas.

Current: Enlace simbólico al *release* actual.

ExtJS: Es una biblioteca de JavaScript para el desarrollo de aplicaciones web interactivas.

Framework: Marco de trabajo.

Git: Software para el control de versiones.

HTML: HyperText Markup Language (Lenguaje de Marcas de Hipertexto). Lenguaje de marcado predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

ISO, IEEE: Son los organismos encargados de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación. Su función principal es la de buscar la estandarización de normas de productos y seguridad para las empresas u organizaciones a nivel internacional.

Log: Fichero que guarda el informe de acceso a la plataforma.

Microsoft, RUP: Principales compañías que llevan la delantera en lo referente a Arquitectura de Software.

Milestones: Iteraciones, hitos.

Plugins: Complemento, conector o extensión.

Rate: Taza, equivalente.

Releases: Versiones del código fuente de la plataforma GESPRO.

Roadmap: Ruta a seguir.

SEI: Software Engineering Institute (SEI). Instituto federal estadounidense de investigación y desarrollo, fundado por el Congreso de los Estados Unidos en 1984 para desarrollar modelos de evaluación y mejora en el desarrollo de software, que dieran respuesta a los problemas que generaba al ejército estadounidense la programación e integración de los sub-sistemas de software en la construcción de complejos sistemas militares. Financiado por el Departamento de Defensa de los Estados Unidos y administrado por la Universidad Carnegie Mellon.

Shared: Directorio que contiene ficheros de configuración, logs y otros ficheros que guarda la plataforma GESPRO.

SSH: Protocolo de conexión.

UML: Unified Modeling Language. Lenguaje Unificado de Modelado.

XML: Extensible Markup Language (Lenguaje de Marcas). Metalenguaje extensible de etiquetas, no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.