# Universidad de las Ciencias Informáticas

# Facultad 4


# Título: Sistema para la Gestión de documentación de los proyectos en la UCI basado en el Expediente de proyecto. Propuesta arquitectónica.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

**Autor:** Yisel Niño Benitez

**Tutor:** Ing. Ramsés Delgado Martínez

**Co-tutor(es):** Ing. Marisleidy Mora Castillo

Ing. Yeilen Morales Román

Ciudad de La Habana, Cuba, Junio, 2009

"Año del 50 Aniversario del Triunfo de la Revolución"

*"La juventud tiene que crear. Una juventud que no crea es una anomalía realmente."*

*Ernesto Guevara*

DATOS DE CONTACTO

**Síntesis del Tutor (Ramsés Delgado Martínez):**

- Graduado de Ingeniero Informático en la CUJAE en el 2006.
- Categoría docente: Instructor recién graduado.
- Especialista de la Dirección de Calidad de Software.
- Correo electrónico: ramsesd@uci.cu

**Síntesis de los cotutores**

**Marisleidy Mora Castillo**

- Graduada de Ingeniero en Ciencias Informático en la Universidad de las Ciencias Informáticas (UCI) en el 2008.
- Profesora Facultad 4.
- Correo electrónico: mmora@uci.cu

**Yeilen Morales Román**

- Graduada de Ingeniero en Ciencias Informático en la Universidad de las Ciencias Informáticas (UCI) en el 2008.
- Correo electrónico: yroman@uci.cu

**AGRADECIMIENTOS**

*A Fidel por ser un incansable soñador.*

*A la UCI por ser más que escuela durante estos cinco años.*

*A Ramsés, Marisleidy (Mora) y Yeilen por la guía, las buenas ideas y sobre todo por la paciencia.*

*A mis padres por haber orientado mi camino, por el aliento ante cada obstáculo, quienes con sus consejos y confianza han hecho de mí lo que soy, por la luz y fuerza con que me nutren a cada paso, por el ejemplo que son, por estar junto a mí aún trascendiendo las distancias.*

*A mi mami Nena por ser más que abuela, por educarme como lo hizo, por todo.*

*A mis hermanos por estar siempre al tanto de todo, por los momentos de risas y de riñas. A Miri.*

*A Yusnier por la vida que compartimos cada día y el espacio que ocupa dentro de mí. Por su ayuda.*

*A Annia y Yadira por su amistad incondicional, por soportar a mi lado las duras y las maduras, por los consejos, las trastadas, y sobre todo por la paciencia.*

*A Yela, Dailin, Sasha, y los muchachos de Calidad-ERP.*

*A los profes que han sembrado su experiencia a lo largo de mi vida.*

*Al 4401 y todas sus iteraciones por ser parte de la palabra "familia" durante estos cinco años, por los amigos que nacieron en su seno y perdurarán.*

*A mis familiares, amigos y aquellos que de una forma u otra han hecho posible la realización de este trabajo.*

*A todos muchas gracias.*

**DEDICATORIA**

*A mis padres como regalo al fruto de sus incansables consejos, sus noches de desvelo y amor.*

*A mi mami Nena.*

*A mis hermanos, Carlitos y Alvarito.*

*A mi BB.*

**RESUMEN**

Los sistemas de Gestión Documental se han convertido en herramientas imprescindibles en las empresas por la necesidad que existe en estas de gestionar el cúmulo de información generada en su accionar diario.

La Universidad de las Ciencias Informáticas no escapa ante la necesidad de un sistema de este tipo que resuelva los problemas existentes en los proyectos de desarrollo de software con respecto a la elaboración y mantenimiento del Expediente de Proyecto, por ello, la Dirección de Calidad de Software en función de poner fin a esta situación indicó la necesidad de desarrollar un sistema de Gestión Documental capaz de gestionar el expediente de forma centralizada, proveyendo a los proyectos de un estándar general que contemple la estructura elemental y plantillas del mismo.

El objetivo de esta investigación es diseñar una propuesta arquitectónica para dicho sistema, teniendo en cuenta estilos y patrones más usados por su eficacia, metodologías de desarrollo de software óptimas además de herramientas libres siguiendo el principio de independencia tecnológica adoptado por el país y la universidad.

PALABRAS CLAVES

Gestión Documental, Expediente de Proyecto

**TABLA DE CONTENIDOS**

**ÍNDICE DE FIGURAS**

**INTRODUCCIÓN**

La actual sociedad de la información y el conocimiento se caracteriza por un uso intenso de la misma en cada una de sus esferas, así como por la necesidad de identificar y utilizar el conocimiento y la información existente en las entidades para ponerlas en función de su misión, objetivos y desarrollo en general.

El crecimiento exponencial de los volúmenes de información en las organizaciones ha colocado a la Gestión Documental (GD) en un lugar privilegiado; volviéndose imprescindible para garantizar el uso adecuado y oportuno de la información.

Incontables son los autores que en diversas bibliografías han tratado de explicar la relación existente entre Gestión de Información (GI) y GD. Sin embargo, en muchas ocasiones resulta un tanto difícil establecer líneas divisorias entre uno u otro concepto, ya que existen puntos de convergencia donde se entremezclan.

Partiendo de esta idea se puede llegar a entender que la información que puede registrarse es, la única que se puede gestionar, y esta sólo se puede registrar de dos formas: en bases de datos (BD) o en documentos.

Gloria Ponjuán se refiere a la GD, como un proceso administrativo que permite analizar y controlar sistemáticamente a lo largo de su ciclo de vida la información registrada que crea, recibe, mantiene y utiliza una organización en correspondencia con su misión, objetivos y operaciones. [Ponjuán, 2004].

La GD, se considera per se, como un proceso para mantener la información en un formato que permita su acceso oportuno, por ello requiere tareas y procedimientos para cada fase y la explotación de esta información registrada, que es evidencia de las actividades y transacciones de las organizaciones, permitiéndole lograr una mayor eficacia.

La GD por tanto, permite el acceso a diversas fuentes, partiendo desde la identificación del documento hasta su archivo, ordenamiento, búsqueda y recuperación. Es la llave para que los usuarios accedan de forma oportuna a la información.

Teniendo en cuenta esto, un sistema de GD posibilita:

- Acceder oportunamente a la información.
- Organizar grandes volúmenes de información.
- Mantener los flujos adecuados de información en la organización.
- Soportar la integridad y seguridad de la información.

Tomando como base lo anteriormente expuesto, no debe obviarse la necesidad que siempre ha representado la GD en nuestro país y más que una necesidad un problema para las organizaciones nacionales, significando gastos en locales, almacenes e infraestructuras para garantizar así el estado de conservación, tiempo dedicado a la organización y búsqueda de documentos, duplicaciones de los mismos, gastos de fotocopias, fax.

La mayoría de estas organizaciones necesitan acceder y consultar de forma frecuente dicha información archivada. En otros casos es la importancia de los documentos o el volumen de información lo que estimula a buscar nuevas soluciones innovadoras que ofrezcan ventajas y valor añadido sobre los sistemas tradicionales de archivo y almacenamiento.

Surgida al calor de la Batalla de Ideas y contemplando entre sus principales actividades la producción de software, la Universidad de las Ciencias Informáticas (UCI) tampoco se encuentra exenta de dicha situación.

Ergo, en beneficio al desarrollo pleno de la elaboración de productos informáticos, tuvo lugar una serie de investigaciones importantes en función de la situación actual que se presenta sobre la GD a la hora de interactuar con la información manejada en el Expediente de Proyecto (EP). Entiéndase por EP el compendio de los documentos obtenidos durante el ciclo de vida de un proyecto, pasando por las fases de iniciación, elaboración, construcción y transición, incluyendo una descripción detallada de cada proceso y su ficha de identificación. De esta forma queda documentado en el EP todo lo referente al desarrollo del software y los artefactos que son generados permitiendo centralizar la totalidad de la información.

Dicho estudio permitió determinar otros problemas a resolver, entre los cuales se destacan:

1. Dificultades a la hora de acceder a las plantillas del EP que deben ser elaboradas en las diferentes etapas de desarrollo del software.
2. No siempre las plantillas del EP satisfacen las necesidades de los especialistas que interactúan directamente con ellas.
3. En ocasiones el tiempo de entrega del producto está limitado y esto conspira contra la correcta elaboración del EP.
4. Se encuentran errores a la hora de hacer la revisión de los documentos del EP en cuanto a los temas relacionados con la elaboración y organización de las plantillas, debido a que estas, en ocasiones no cumplen con el estándar que se establece para la organización de la documentación en estos casos.

Partiendo de la situación problemática anteriormente expuesta se llega al siguiente **problema a resolver**: ¿Cómo contribuir a mejorar el trabajo con el Expediente de Proyecto a partir de la definición de la base y lineamientos para el desarrollo de un sistema de Gestión Documental?

Por tanto el **objeto de estudio** se enmarca en las tecnologías, estilos y patrones arquitectónicos usados en arquitecturas implementadas para sistemas de gestión, definiendo como **campo de acción** las arquitecturas de sistemas de gestión desarrolladas en la UCI.

Teniendo en cuenta los problemas existentes en la UCI con respecto a la GD se plantea como **objetivo general**: diseñar la arquitectura del sistema para la gestión de documentación de proyectos en la UCI.

Se plantean además como **objetivos específicos:**

- Realizar un estudio del estado del arte para fundamentar la investigación y dejar definida la posición del investigador.
- Obtener requisitos del sistema.
- Obtener descripción de la arquitectura.

Para lograr cumplir satisfactoriamente los objetivos trazados se proponen como **tareas**:

1. Realizar un estudio sobre las distintas herramientas existentes en el mercado que se relacionan con la GD, así como las necesarias para lograr definir la arquitectura del sistema GEDEP luego de analizar los distintos patrones existentes, además de estudiar la documentación que compone un EP en la UCI.
2. Realizar entrevistas a los líderes de proyecto con el objetivo de comprender cómo se realiza la gestión del EP.
3. Realizar el modelado de dominio.
4. Identificar los requisitos a partir de los procesos de negocio.
5. Realizar el modelo de casos de uso.
6. Realizar un prototipo de interfaz gráfica de usuario para validar los requisitos.
7. Fundamentar los patrones de arquitectura a utilizar, así como su aplicación en la solución del problema.
8. Diseñar la arquitectura de un sistema que permita la gestión de la documentación de los EP.

Con el cumplimiento de las tareas anteriores se espera como **resultado** una propuesta arquitectónica para la aplicación de GD basada en el EP.

El **método científico de investigación** es la forma de abordar la realidad, de estudiar la naturaleza, la sociedad y el pensamiento, con el propósito de descubrir su esencia y sus relaciones. En la realización del

presente trabajo de diploma se utilizaron los métodos teóricos y los empíricos. Los primeros permiten comprender el fenómeno que se estudia, su evolución y proponer las mejoras respectivas a los problemas identificados.

- Histórico-lógico: Este método permitió realizar la primera parte de la investigación, haciendo un análisis bibliográfico del tema, para determinar a través de la evaluación de la bibliografía conceptos de la temática que permitieron conocer el estado actual del fenómeno.

- Analítico-sintético: Se utilizó en esta investigación para analizar teorías, documentos, y diferentes tipos de bibliografía; permitiendo la extracción de los elementos más importantes que se relacionan con el objeto de estudio permitiendo descomponer el problema de investigación en elementos por separado y profundizar en el estudio de cada uno de ellos, para luego sintetizarlos en la solución de la propuesta.

Los métodos empíricos, por su parte, permitieron describir y explicar las características del fenómeno en estudio. Dentro de estos, se aplicaron métodos particulares con el objetivo de recolectar los datos necesarios para identificar la problemática y las causas de esta, así como determinar la magnitud de su influencia:

- Observación: con el objetivo de ampliar y corroborar la información obtenida a través de las entrevistas realizadas.

- Revisión de documentos: en pos de la determinación del estado del arte del objeto de investigación.

- Entrevista: vitales para establecer los elementos necesarios para la lógica del modelo, avalar los conceptos que se manejan en la investigación, medir el alcance y la importancia que tiene la temática. Propicia además captar la información cualitativa y cuantitativa del fenómeno y conocer los criterios sobre la forma en que se organiza y controla el EP.

## CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

### 1.1 Introducción.

En el presente capítulo se ofrece un estudio sobre el estado del arte correspondiente a la GD a escala mundial y la utilidad que esta puede representar a lo largo del proceso de desarrollo de software en la UCI, enfocándola directamente en la gestión del EP. Se realizará además un estudio detallado de los principales conceptos relacionados con la problemática explicada en el apartado anterior, así como de las tendencias, tecnologías y metodologías actuales, con el fin de proponer las más adecuadas para la solución del problema.

### 1.2 Gestión Documental.

#### 1.2.1 Definiciones.

Una definición con un enfoque simple de los sistemas de GD, es la que brinda Lluís Codina: *"son programas de gestión de bases de datos que disponen de una tecnología idónea para el tratamiento de documentos científicos, culturales y técnicos."* [Codina, 2003].

Un software de GD hace figurar entre sus ventajas la reducción del costo de una empresa para almacenar y manipular cierta documentación. Al implementarlo se ahorra un tiempo valioso debido a que los empleados ya no tienen que valerse de esa misma cantidad de tiempo en la clasificación de archivos y recuperación de documentos.

Para Gloria Ponjuán la GD es, según lo que plantea: *"un proceso para mantener la información en un formato que permita su acceso oportuno, y por ello requiere tareas y procedimientos para cada fase y la explotación de esta información registrada, que es evidencia de las actividades y transacciones de las organizaciones y que les permite lograr una mayor eficacia."* [Ponjuán, 2004].

Diversas fuentes definen la GD como todo un compuesto de normas, técnicas y prácticas usadas para administrar el flujo de documentos que se genera en una organización, permitir recuperar dicha información desde ellos, determinar el tiempo que los documentos deben permanecer archivados o eliminar los que de cierta forma ya han perdido validez asegurando así mismo la conservación indefinida de los documentos más valiosos. [Herrera, 2007].

Patel plantea que la GD: *"proporciona acceso a, y control de, los documentos y otros objetos relacionados a lo largo de su ciclo de vida incluyendo su creación, aprobación, distribución y reutilización."* [Patel, 1997].

En tanto para Cleveland, la GD se enfoca en: *"el control automatizado de documentos electrónicos a lo*

*largo de su ciclo de vida en una organización, desde su creación inicial hasta su archivado final."* [Cleveland, 1995].
Otra definición es la propuesta por la norma ISO 15489:2001, donde se plantea la GD desde la perspectiva siguiente: *"los gestores de documentos e información tienen el cometido de poner en valor los recursos informativos disponibles, para documentar los procesos que se están llevando a cabo en la organización en un momento dado"*. [Werner, 2005].

Desde un punto de vista particular los sistemas de GD son programas de gestión que disponen de la tecnología adecuada para mantener la información que se crea o recibe en un formato que permite su acceso oportuno, administrando así el flujo documental que es generado en una organización.

### 1.2.2 Valoraciones.

El uso de la GD es algo que ha ido ascendiendo gradualmente debido a todos los beneficios que reporta para las organizaciones que adopten este tipo de sistema con el fin de gestionar toda la documentación que le sea importante o primordial, pues de una forma simple, puede tener acceso inmediato a la totalidad de la información necesaria para alguna actividad específica, con las ventajas añadidas de reducción de tiempo de consultas y tareas de archivo o ahorro de espacio físico. Además facilita que la información se comparta y aproveche de forma más eficiente y como un recurso colectivo. Como consecuencia, se reducen drásticamente situaciones como la duplicidad de documentos archivados, fotocopias innecesarias, dobles grabaciones de datos, entre otros.
Gloria Ponjuán, en cuanto a la GD y sus facilidades expone: *"es la llave para que los usuarios accedan de forma oportuna a la información. Se relaciona con la Gestión del Conocimiento (GC) al tratar de colocar a disposición de los integrantes de una organización, las experiencias e ideas explícitas y que se pueden reutilizar en función de un propósito determinado."* [Ponjuán, 2004].
Lluís Codina expone que: *"La industria informática, por motivos de marketing o, simplemente, por mal conocimiento del sector, ofrece como documentales soluciones que sirven perfectamente para gestionar documentos administrativos, pero no para gestionar documentación científico-técnica. Por otro lado, en los departamentos de informática de las empresas, es frecuente que el personal informático desconozca el hecho de que existen programas especialmente diseñados para ese tipo de documentos y, normalmente, tienden a imponer aquellas soluciones que conocen bien, pero que no proporcionan a los documentalistas las herramientas adecuadas para su trabajo. (…) Por ello, en el momento de considerar la adquisición de*

*un sistema documental, es importante disponer de criterios que ayuden, bien a adoptar las decisiones de compra más correctas, bien a disponer de argumentos de negociación con el departamento de informática."* [Codina, 2003].

### 1.2.3 Ámbito internacional.

Encarna González hace un breve estudio referente a la necesidad que existe a nivel mundial, enfocándolo en la oportuna adopción de un sistema de GD: *"Según datos de la consultora IDC, el 82 por ciento de las compañías piensa que los documentos que maneja diariamente son vitales para el éxito de su negocio. Sin embargo, aproximadamente el 90 por ciento no es capaz de calcular lo que gasta en gestionar toda esta información. Esto, unido a la necesidad de disponer de información de forma rápida y sencilla, y a la proliferación de normativas que obligan a tener un mayor control de los datos, hace necesario dotarse de sistemas de gestión documental que ayuden a las empresas en estas actividades. De los equipos escogidos y su administración dependerá el éxito de una correcta gestión documental en beneficio de su negocio."* [González, 2007].

No es nada nuevo el enorme y creciente volumen de datos que día a día han de gestionar las empresas. Es por ello que la adopción de políticas, estrategias, procedimientos y tecnologías asociadas a la creación, gestión y almacenamiento de contenidos y documentos digitales en las empresas está adquiriendo mayor importancia. La gestión de toda la información que manejan las corporaciones implica grandes esfuerzos en catalogación, consultas, acceso e integración, ya que, por lo general, toda esta documentación se encuentra dispersa en distintos soportes, muchos no digitalizados, y en distintos departamentos. No es de extrañar pues, que para los responsables de las organizaciones, la GD sea una necesidad inminente y esté proliferando la búsqueda de nuevas soluciones que ofrezcan ventajas y valor añadido sobre los tradicionales sistemas de archivo y almacenamiento.

La GD ha ido ganando importancia en las organizaciones a medida que han ido creciendo los volúmenes de datos no estructurados, pues se sofistican los formatos de los archivos y la presión normativa exige una mayor transparencia y control respecto a los documentos que genera y maneja una empresa. Ya sea con un sistema propio o a través de uno externalizado, la GD empieza por la captura o indexación de los documentos y acaba en su búsqueda y archivo a través de una solución dividida en componentes de software de servicios de consultoría e integración, así como de un hardware específico.

En España (uno de los países pioneros en el tema de la GD) la tendencia que muestra el mercado en la actualidad es la de superar la reticencia inicial a la externalización de procesos. Lo que se busca es

minimizar el tiempo y ganar el máximo de eficiencia operativa. La externalización (outsourcing) ofrece mejoras continuas en la GI, disponiendo de unos centros de digitalización y proceso en ubicaciones remotas con costes muy asequibles tanto de almacenaje como de mano de obra, entre otros. La gestión de documentos en formato digital, ya sean facturas o cualquier otro tipo de ficheros, ofrece una cantidad de ventajas y oportunidades que las empresas no deberían pasar por alto: empleados y procesos más eficientes, procesos más ágiles, clientes más satisfechos. La implantación de un sistema de GD de información digitalizada promueve un entorno que impulsa la productividad y la eficiencia con resultados inmediatos que, como pocos, benefician y ofrecen grandes oportunidades de negocio.

### 1.2.4 Ámbito nacional.

A pesar de las limitaciones, de las barreras objetivas y subjetivas existentes, y teniendo en cuenta el proceso de informatización de la sociedad en el que se encuentra actualmente el país, se le da la posibilidad a la archivística nacional de encontrar soluciones óptimas ante los problemas reales que presenta, ya que se ha demostrado la escasa existencia de archivos en nuestras administraciones y la acumulación de grandes volúmenes documentales sin ningún tipo de tratamiento. Por esta razón, buena parte de los archivos administrativos existentes no pasan de ser depósitos de papel, completamente ajenos a las necesidades informativas de las organizaciones y concebidos en función de la conservación de los documentos para su utilización sólo como fuentes de investigación histórica.

Una de las soluciones nacionales para la GD es el producto AvilaDoc, realizado por el equipo de trabajo de la empresa DESOFT de la provincia Ciego de Ávila. AvilaDoc es una aplicación web, desarrollada sobre la plataforma de software libre con una BD centralizada, destinada a la gestión, tramitación y resguardo de archivos electrónicos y/o digitales; facilitando la búsqueda o recuperación de información de forma rápida y sencilla. Incorpora el fichado de la documentación en un expediente como punto de partida, simulando el flujo de la documentación en una entidad.

Por su parte el Ministerio de Ciencia, Tecnología y Medio Ambiente (CITMA) en Santiago de Cuba se trazó una serie de metas con respecto a la búsqueda de soluciones para la GD en soporte digital, con el objetivo de poder conservar y tener un mejor acceso a los diferentes materiales que se han registrado tanto de archivos históricos como de otras fuentes de información.

En la provincia de Holguín se realizaron estudios en pos de la implementación de un sistema de GD capaz de resolver diferentes problemas existentes en los archivos de oficina tales como: desorganización, pérdida de documentos valiosos, lentitud en la búsqueda y entrega de respuestas a los usuarios.

La creación y funcionamiento de un sistema de GD en la Delegación Territorial del CITMA tiene una importancia significativa pues esto propicia innegables beneficios en ahorro de tiempo y espacio y como institución donde los alumnos del Curso de Técnicos Medio en Gestión Documental realizan sus prácticas laborales.

### 1.3 Soluciones de Software existentes que facilitan la GD.

GIT-DOC es la integración de soluciones que tradicionalmente se habían desarrollado de forma independiente para satisfacer las diferentes demandas de la GD. Estas son principalmente: captura e indexación de documentos, recuperación de la información, gestión de contenidos y automatización de flujos de trabajo (distribución de tareas, seguimiento y archivado).
Hace figurar entre sus beneficios:

1. Rápida localización y consulta de documentos.
2. Reducción de riesgos y costes de transporte y almacenamiento de la información.
3. Realizar cualquier número de copias autorizadas.
4. Creación de nuevos canales de comunicación para empresas con dispersión geográfica.
5. Garantía de un entorno seguro y flexible.
6. Reducción significativa de los costes.
7. Utilización simultánea por más de un usuario de un mismo documento.
8. Control y seguimiento de la distribución de documentos.
9. Control estadístico de conexiones que facilita la gestión y toma de decisiones.

Yerbabuena ECM es una herramienta de código abierto, que permite gestionar la documentación y desplegar aplicaciones web empresariales. En los últimos tiempos el gran volumen, la complejidad y diversidad de información estructurada y no estructurada que maneja una empresa**,** sin importar el tamaño de la misma, ha provocado que apoyarse y ayudarse en herramientas de GD sea algo imprescindible en cualquier tipo de organización para evitar el caos en los sistemas de información. De este modo, con el sistema de GD Yerbabuena ECM, totalmente integrado en el funcionamiento de la empresa, se puede gestionar y controlar el flujo de documentación existente en la misma.

PAPIRO, es un producto informático de código abierto que emplea herramientas igualmente libres y permite conservar documentación de valor histórico al evitar su manipulación; pues, al digitalizarse el

documento, su consulta se realiza en formato electrónico. Esta versión incluye también la gestión de publicaciones periódicas, especialmente revistas.
La consulta de los documentos puede ser realizada a través de una eficiente gestión que permite recuperar información por todas las variables de almacenamiento (campos de obligatoria presencia según la Norma Internacional General de Descripción Archivística (General International Standard Archival Description: ISAD-G por sus siglas en inglés) para intercambio internacional) en la BD; mientras, la posibilidad de colocar en Internet dichas BD y las imágenes de los documentos no sólo democratiza la accesibilidad documental; sino que ahorra recursos materiales y tiempo, haciendo más eficaz y eficiente el proceso de búsqueda e investigación.

Alfresco es un sistema de administración de contenidos de código libre, basado en estándares abiertos y de escala empresarial para Windows y sistemas operativos similares a Unix. Está diseñado para usuarios que requieren un alto grado de modularidad y rendimiento escalable. Incluye un repositorio de contenidos, un marco de trabajo (en lo adelante framework) de portal web para administrar y usar contenido estándar en portales, una interfaz CIFS (Common Internet File System: Sistema de archivos comunes de Internet) que provee compatibilidad de sistemas de archivos en Windows y sistemas operativos similares a Unix, un sistema de administración de contenido web con capacidad de virtualizar aplicaciones web y sitios estáticos vía Apache Tomcat, búsquedas vía el motor Lucene y flujo de trabajo en jBPM. Desarrollado en Java. Utilizado como software de GD para documentos, páginas web, registros, imágenes y desarrollo colaborativo de contenido con un gran éxito.
Es un sistema fácil de usar, que ayuda en la productividad del desarrollador con una de las mejores prácticas de colaboración. Tributa además a la productividad del administrador. Posee un gestor de búsqueda avanzada y una arquitectura distribuida.

Independientemente de los beneficios o ventajas que puedan reportar los diferentes sistemas antes mencionados en uno u otro aspecto, para proponer una solución ante la situación problemática que se presenta en la realización del trabajo y antes analizada, se arriba a la conclusión que no son del todo idóneos para lo que desea el cliente, ya que estos sistemas no logran satisfacer las necesidades del mismo, pues necesita además de gestionar la documentación de un EP, mantener una estructura única para los mismos y lograr establecer un estándar que sea manejado de forma central, garantizando así mayor calidad en los artefactos resultantes.

El objetivo por tanto, es obtener una propuesta de arquitectura para un nuevo sistema de GD que sea capaz de gestionar todos los temas relacionados con el EP, ya sea desde la organización de los diferentes artefactos y estructuras que lo componen hasta lograr una escritura estándar para la documentación del mismo a partir del trabajo de sus plantillas, consiguiendo así que la totalidad de los proyectos de producción de software de la UCI sigan un mismo patrón en cuanto al EP.

## 1.4 Metodologías de desarrollo de software.

En un proyecto de desarrollo de software la metodología que se use es justamente quien define cómo debe llevarse a cabo un proceso determinado: quién debe hacer qué, cuándo y cómo debe hacerlo. Sin embargo, la tarea de seleccionar una metodología específica no es simple gracias a la gran cantidad existente para tener en cuenta a la hora de desarrollar un proyecto de software. Dichas metodologías se encuentran divididas en dos grandes grupos:

- tradicionales o pesadas: centradas fundamentalmente en el control del proceso, basadas en normas provenientes de estándares seguidos por el entorno de desarrollo estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, así como las herramientas y notaciones que se utilizarán, la arquitectura del software es esencial y se expresa mediante modelos.
- livianas o ágiles: centradas mayormente en el factor humano y en el producto de software, dando mayor valor al individuo, a la colaboración con el cliente haciéndolo formar parte del equipo de desarrollo del software y al desarrollo incremental del software con iteraciones muy cortas, hace poco énfasis en la arquitectura del software.

A continuación se muestra una pequeña descripción de las encontradas con mayor frecuencia:

**Extreme Programming (XP).**

La Programación Extrema o eXtreme Programming (XP) como también se le conoce mundialmente, es un enfoque de la ingeniería de software formulado por Kent Beck, autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999). Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, XP se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad, además de caracterizarse por realimentarse de manera continua entre el cliente y el equipo de desarrollo, mantener una comunicación fluida entre todos los participantes, en tanto apuesta que es más sencillo hacer algo

simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a estos cambios en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del mismo e invertir esfuerzos después en controlar los cambios.

**Agile Software Development (ASD).**

Teniendo como impulsor a Jim Highsmith, el Desarrollo Ágil de Software (ASD por sus siglas en inglés) es una metodología de trabajo que consiste en el desarrollo de soluciones informáticas para clientes, quienes solicitan el software a una empresa desarrolladora, la que en un proceso cíclico, que incluye revisiones periódicas del producto, logra cumplir las expectativas de este en períodos más cortos. El objetivo que persigue es que la aplicación se implemente exactamente con los requisitos que el cliente necesita realmente, aún cuando estos vayan evolucionando durante el proceso de desarrollo, lo que en este caso es algo clave.

Sus principales características son: iterativo, orientado a los componentes de software más que a las tareas y tolerante a los cambios. El desarrollo exitoso de sistemas informáticos con ASD se enmarca objetivamente en cinco etapas claves: escribir el código de prueba, realizar diseños simples, revisar constantemente el código fuente, trabajar a un ritmo sostenible y real, en tanto se refina el proceso.

**Dynamic Systems Development Method (DSDM).**

Define el marco para desarrollar un proceso de producción de software. Nace en Gran Bretaña en 1994 con el objetivo de crear una metodología RAD (Desarrollo Rápido de Aplicaciones) y desarrollo iterativo. DSDM reconoce que los proyectos son limitados por el tiempo, los recursos y los planes acorde a las necesidades de la empresa. Para alcanzar estas metas, DSDM promueve el uso del RAD. Ser un proceso iterativo e incremental y en el que el equipo de desarrollo y el usuario trabajan juntos, figuran entre las principales características de DSDM. Su ciclo de vida contempla cinco fases: estudio de viabilidad, estudio del negocio, modelado funcional, diseño y construcción e implementación.

**Feature-Driven Development (FDD).**

Diseñado por Peter Coad, Erich Lefebvre y Jeff De Luca, se basa en un proceso de iteraciones cortas. Está pensado para proyectos con tiempo de desarrollo relativamente cortos. Propone tener etapas de cierre cada dos semanas aproximadamente, lo cual implica que los desarrolladores tendrán nuevas actividades que realizar en dicho período de tiempo, lo que mantendrá al equipo de desarrollo en una constante motivación.

Con FDD, se obtienen porcentajes reales del proceso, lo cual ayuda a demostrar al cliente dónde se encuentra el proyecto, de esta forma pretende dejar satisfechos tanto a los desarrolladores, gerentes y clientes sin afectar al proyecto. Está centrado en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software. Contiene cinco procesos durante los que se diseña y construye el sistema: desarrollo de un modelo global, construcción de la lista de funcionalidades, planeamiento de versiones en base de funcionalidades a implementar, diseño y construcción por funcionalidades.

**SCRUM.**

Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle, SCRUM ha estado presente durante algún tiempo en los círculos orientados a objetos, se focaliza en priorizar el trabajo en función del valor que tenga para el negocio, maximizando la utilidad de lo que se construye y el retorno de inversión. Está diseñada especialmente para adaptarse a los cambios en los requisitos, por ejemplo en un mercado de alta competitividad. Los requisitos y las prioridades se revisan y ajustan durante el proyecto en intervalos muy cortos y regulares. De esta forma se puede adaptar en tiempo real el producto que se está construyendo según las necesidades del cliente en pos de aumentar su satisfacción final.

**Rational Unified Process (RUP).**

El Proceso Unificado de Software (Rational Unified Process, habitualmente RUP por sus siglas en inglés) es un proceso de desarrollo y junto con el Lenguaje Unificado de Modelado (UML por sus siglas en inglés), constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

RUP es una recopilación de prácticas de ingeniería de software que se están mejorando continuamente de forma regular para reflejar los cambios en las prácticas de la industria. Proporciona a un profesional de desarrollo de software un entorno de proceso configurable basado en estándares.

El ciclo de vida de RUP consta de tres características que lo guían:

- Iterativo e incremental: permite dividir el proyecto en pequeños sub-proyectos para desarrollarlo en distintas etapas e iteraciones que resultan en un incremento del producto.
- Dirigido por casos de uso: es uno de los métodos más utilizados y efectivos para reflejar los requisitos. Estos no solo sirven para especificar los requisitos que son definidos de acuerdo a las expectativas de los clientes, sino que ellos son los encargados de guiar el ciclo de vida del proyecto.
- Centrado en la arquitectura: son los casos de uso los encargados de guiar a la arquitectura del sistema y esta influye en la selección de los casos de uso. La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente.

RUP propone nueve flujos de trabajo en los que se definen las secuencias de las actividades, quienes las deben desarrollar y los artefactos que deben generarse en cada una de ellas. Estos son:

- Modelado del negocio: Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
- Requerimientos: Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
- Análisis y diseño: Describe cómo el sistema será realizado a partir de la funcionalidad prevista y los requerimientos, por lo que indica con precisión lo que se debe programar.
- Implementación: Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.
- Prueba: Busca los defectos a lo largo del ciclo de vida.
- Instalación: Produce un release del producto y realiza las actividades de empaque, instalación, asistencia a usuarios, entre otros, para entregar el software a los usuarios finales.
- Administración del proyecto: Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.
- Administración de configuración y cambios: Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización y actualización concurrente de elementos, control de versiones, entre otros.

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

Adheridas a estos nueve flujos de trabajo de RUP, se encuentran las cuatro fases que lo definen:

- Inicio: determina la visión del proyecto.
- Elaboración: define la arquitectura del sistema.
- Construcción: obtiene un producto listo para su utilización que está documentado y tiene un manual de usuario.
- Transición: obtiene el release ya listo para su instalación. Puede implicar reparación de errores.

Posterior a la realización del estudio acerca de las diferentes metodologías de desarrollo de software y las facilidades que brindan, se decide usar RUP para guiar el proceso de desarrollo de la arquitectura del sistema, puesto que, de las estudiadas, RUP es la que más importancia presta al proceso de descripción de la arquitectura siendo este el hito de la segunda fase.

### 1.5 Arquitectura de software.

La Arquitectura de Software (AS) está compuesta por la estructura de los elementos de un programa o sistema, sus interrelaciones y los principios o reglas que gobiernan su diseño y evolución a lo largo del tiempo.

También denominada Arquitectura Lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software. Establece los fundamentos para que el equipo de desarrollo logre trabajar en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades.

Existen muchas definiciones sobre AS. Perry y Wolf en 1992 la definieron como un conjunto de elementos que tienen una forma común. Garlan and Shaw en 1994 y 1996 la conceptualizaron como una colección de componentes y conectores unidos con una descripción de la interacción entre esos componentes y conectores. Bass, Clements, y Kazman en 1998 la definen como la estructura de las estructuras de un sistema, la cual comprende componentes de software, las propiedades visibles externas de estos componentes, y las relaciones entre ellos. La definición “oficial” de AS se ha acordado que sea la que brinda el documento de IEEE Std 1471-2000 que se formula así: *“La Arquitectura de Software es la*

*organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos, el ambiente y los principios que orientan su diseño y evolución."* [KAISER, 2005].

Cada paradigma de desarrollo exige diferente número y tipo de vistas o modelos para describir una arquitectura. No obstante, existen al menos tres vistas absolutamente fundamentales en cualquier arquitectura:

- La visión estática: describe qué componentes tiene la arquitectura.
- La visión funcional: describe qué hace cada componente.
- La visión dinámica: describe cómo se comportan los componentes a lo largo del tiempo y cómo interactúan entre sí.

### 1.5.1 Estilos arquitectónicos.

Un estilo describe una clase de arquitectura, o piezas identificables de las arquitecturas empíricamente dadas [Pimentel, y otros, 2007]. Un estilo arquitectónico es un conjunto coordinado de restricciones arquitectónicas que limita los roles o rasgos de los elementos arquitectónicos y las relaciones permitidas entre esos elementos dentro de la arquitectura que se conforma a ese estilo. Son una generalización y abstracción de los patrones de diseño.

Existe una gran variedad de estilos arquitectónicos las cuales tienen diversas clasificaciones entre las que se encuentran:

- Estilos de flujo de datos:
    - Filtro-Tubería.
    - Secuencial por Lotes.
- Estilos centrados en datos:
    - Bases de Datos.
    - Sistemas de Hipertexto.
    - Arquitectura de Pizarra.
- Estilos de llamada y retorno:
    - Sistema Modular.
    - Arquitectura Orientada a Objetos.
    - Jerarquía de Capas.

    - Arquitectura Cliente – Servidor.
- Estilos de código móvil:
    - Arquitectura de Máquinas Virtuales.
- Estilos peer-to-peer:
    - Arquitecturas Basadas en Eventos.
    - Arquitecturas Orientadas a Servicios.
    - Arquitecturas Basadas en Recursos.
- Estilos heterogéneos:
    - Sistemas de Control de Procesos.
    - Arquitecturas Basadas en Atributos.

**Arquitectura Cliente-Servidor.**

Consiste básicamente en un programa -el cliente- que realiza peticiones a otro -el servidor- que le da respuesta. Aunque esta idea puede ser aplicable a programas que se ejecutan sobre una sola computadora, es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.

En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

Los clientes pueden ser clasificados como clientes “flacos” y/o “gordos”. Los clientes gordos típicamente contienen, además de la lógica de presentación, gran parte de la lógica de negocio de la aplicación. Los clientes flacos manejan usualmente sólo la lógica de presentación lo que adiciona grandes ventajas como permitir que futuros cambios de negocio en la aplicación no afecten al cliente.

**Jerarquía de tres capas.**

La arquitectura de una aplicación es la vista conceptual de la estructura de esta. Toda aplicación contiene código de presentación, código de procesamiento de datos y código de almacenamiento de datos. La arquitectura de las aplicaciones difiere según como está distribuido este código.

En [GS94] Garlan y Shaw definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.

Las tres capas que propone esta arquitectura son:

- Lógica de Presentación: acomoda la parte del código que interactúa con un dispositivo como una computadora o terminal de autoservicio. Se encarga de tareas como la disposición de los elementos gráficos en la pantalla, escribir los datos en pantalla, manejo de ventanas, entre otros.
- Lógica de Negocio: en ella se codifican las reglas del negocio. Además, residirán todas las entidades de dominio y los componentes encargados de manejar la lógica de negocio de los mismos.
- Lógica del Procesamiento de los Datos: En esta capa se oculta la forma en que se consultan o almacenan los datos persistentes en la BD.

**Modelo – Vista – Controlador.**

Un propósito común en numerosos sistemas es el de tomar datos de un almacenamiento y mostrarlos al usuario. El patrón Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

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

### 1.5.2 Estilos y Patrones. Relación entre ellos.

Una vez que se ha hecho un estudio minucioso y se han seleccionado los estilos a usar en el desarrollo de la aplicación, los patrones de arquitectura que tengan relación con estos ayudarán a derivar de esa formulación arquitectónica el diseño y posteriormente la programación correspondiente.

**Patrones, un enfoque más amplio.**

Los patrones fueron sugeridos por Christopher Alexander en 1977, el cual escribió una serie de libros y artículos procurando dar respuesta a la interrogante: ¿Qué son los patrones? Los textos encontrados con

más frecuencia se refieren a patrones arquitectónicos en cuanto a arquitectura real, o sea de edificios y ciudades, no así para arquitectura de aplicaciones de software.

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

Los patrones pueden dividirse o clasificarse en:

- Patrones de Arquitectura**:** relacionados a la interacción de objetos dentro o entre niveles arquitectónicos. Resuelve problemas arquitectónicos, de adaptabilidad a requerimientos cambiantes, performance, modularidad, acoplamiento.
- Patrones de Diseño: relacionado con conceptos de ciencia de computación en general, independientemente de la aplicación para la que fueron construidos, teniendo en cuenta los problemas existentes con la claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, proponiendo como solución: Comportamiento de factoría, Clase-Responsabilidad-Contrato (CRC).
- Patrones de Análisis: usualmente específicos de aplicación o industria.
- Patrones de Proceso o de Organización: relacionados con el desarrollo o procesos de administración de proyectos, o técnicas, o estructuras de organización. Resuelve problemas de productividad, comunicación efectiva y eficiente.
- Patrones de Idioma: reglan la nomenclatura en la cual se escriben, se diseñan y desarrollan nuestros sistemas, o sea los estándares de codificación y proyecto.

Los elementos de un patrón son:

- Nombre.
    - Define un vocabulario de diseño.
    - Facilita la abstracción.
- Problema.
    - Describe cuando aplicar el patrón.
    - Conjunto de fuerzas objetivas y restricciones.
    - Prerrequisitos.
- Solución.
    - Elementos que constituyen el diseño (plantilla).
    - Forma canónica para resolver fuerzas.

- Consecuencias.
    - Resultados.
    - Extensiones.
    - Consensos.

## 1.6 Herramientas, lenguaje de desarrollo y tecnologías actuales.

### 1.6.1 Herramientas CASE.

Las herramientas CASE (Computer Aided Software Engineering: Ingeniería de Software Asistida por Ordenador) son aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, el cálculo de costes, la implementación de parte del código automáticamente con el diseño dado, así como la compilación automática, documentación o detección de errores.

**Rational Rose.**

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

**Visual Paradigm Suite.**

Visual Paradigm es una Suite de herramientas que figura también dentro de las llamadas CASE. Soporta el ciclo de vida completo del desarrollo de software: negocio, análisis y diseño orientado a objetos, construcción, pruebas y despliegue. Proporciona además abundantes tutoriales, demostraciones interactivas y proyectos UML, lenguaje de modelado UML ayuda a una rápida construcción de aplicaciones con mayor calidad y a un menor coste.

Distinto a muchas herramientas de modelado pueden extenderse sus diagramas hechos desde el Visio hasta los del Rational Rose. Soporta actualmente la última versión de UML 2.1 y BPMN, permite realizar ingeniería tanto directa como inversa.

Permite generar:

- Diagramas de Casos de Uso.
- Diagramas de Clases.
- Diagramas de Secuencia.
- Diagramas de Comunicación.
- Diagramas de Estado.
- Diagramas de Componentes.
- Diagramas de Despliegue.
- Diagramas de Objetos.
- Diagramas de Interacción.
- Diagramas de Entidad Relación.
- Diagramas ORM (Object Role Modeling, Modelado de funciones de objetos).
- Diagramas de Procesos del Negocio.
- Diagramas de EJB (Enterprise Java Bean).
- Diagramas de visión general.

Además se integra con IDEs (Eclipse, JBuilder, NetBeans, IntelliJ IDEA, JDeveloper and WebLogic Workshop) para soportar la fase de implementación de desarrollo de software. La transición desde el análisis al diseño y después a la implementación es cuidadosamente integrada dentro de la herramienta CASE, de esta manera se reduce significativamente el esfuerzo en todas las etapas del ciclo de vida del desarrollo del software. Permite generar los EJB y así mismo los descriptores de despliegue para varios servidores de aplicación.

Atendiendo a las cuestiones anteriores, se ha decidido escoger el Visual Paradigm como herramienta para el modelado del GEDEP.

### 1.6.2 Zend Studio.

Zend Studio o Zend Development Environment es uno de los ambientes de desarrollo integrado o Integrated Development Environment (IDE) disponible para desarrolladores profesionales que agrupa

todos los componentes necesarios para un ciclo de desarrollo de aplicaciones PHP. Escrito en Java, y disponible para las plataformas Microsoft Windows, Mac OS X y GNU/Linux.

El programa, además de servir de editor de texto para páginas PHP, proporciona una serie de ayudas que van desde la creación y gestión de proyectos hasta la depuración de código, lo que acelera los ciclos de desarrollo y simplifica los proyectos complejos.

Zend Studio consta de dos partes en las que se dividen las funcionalidades de parte del cliente y las del servidor. Permite además hacer depuraciones simples de scripts, aunque para disfrutar de toda la potencia de la herramienta de depuración habrá que disponer de la parte del servidor, que instala Apache y el módulo PHP o, en caso de que estén instalados, los configura para trabajar juntos en depuración.

Aunque Zend Studio fue diseñado para usarse con el lenguaje PHP ofrece sin embargo soporte básico para otros lenguajes web como HTML, JavaScript y XML.

#### 1.6.2.1 Zend Studio for Eclipse.

Zend Studio para Eclipse IDE es lo más novedoso en la familia Zend Studio. Diseñado para desarrolladores profesionales de PHP, esta nueva versión combina un IDE versátil y potente con las capacidades de expansión del ecosistema del proyecto Eclipse. Dispone de un entorno mucho más flexible y profesional para controlar todo el ciclo de vida de un proceso de desarrollo.

Entre sus funcionalidades, destacaría las capacidades de refactorización del código fuente, funcionalidad que permite adecuar el comportamiento externo de una función o clase sin cambiar el funcionamiento interno, que junto a los nuevos winzards y capacidades de generación de código facilitarán el trabajo a los desarrolladores.

### 1.6.3 Lenguajes de programación web.

**ASP.**

ASP (Active Server Pages: Páginas de Servicio Activo) es una tecnología de Microsoft del tipo "lado del servidor" para páginas web generadas dinámicamente, que ha sido comercializada como un anexo a la IIS (Internet Information Services: Servicios de Información de Internet).

La tecnología ASP está estrechamente relacionada con el modelo tecnológico de su fabricante. Intenta ser solución para un modelo de programación rápida, por supuesto con muchas limitaciones. Lo interesante de este modelo tecnológico es poder utilizar diversos componentes ya desarrollados como algunos controles ActiveX así como componentes del lado del servidor.

**JSP.**

JavaServer Pages (Servidores de Páginas de Java: JSP por sus siglas en inglés) es una tecnología Java que permite generar contenido dinámico para la web, en forma de documentos HTML, XML o de otro tipo, desarrollada por la compañía Sun Microsystems. La especificación JSP 1.2 fue la primera que se liberó y en la actualidad está disponible la especificación JSP 2.1.

Las JSP's permiten la utilización de código Java mediante scripts. Además es posible utilizar algunas acciones JSP predefinidas mediante etiquetas. Estas etiquetas pueden ser enriquecidas mediante la utilización de librerías de etiquetas (TagLibs o Tag Libraries) externas e incluso personalizadas.

La principal ventaja de JSP frente a otros lenguajes es que el lenguaje Java es un lenguaje de propósito general que excede el mundo web y que es apto para crear clases que manejen lógica de negocio y acceso a datos de una manera prolija. Esto permite separar en niveles las aplicaciones web, dejando la parte encargada de generar el documento HTML en el archivo JSP.

**PHP.**

Se trata de un lenguaje de creación relativamente reciente que ha tenido una gran aceptación en la comunidad de desarrolladores debido, sobre todo, a la potencia y simplicidad que lo caracterizan. Permite embeber pequeños fragmentos de código dentro de la página HTML y realizar determinadas acciones de una forma fácil y eficaz, sin tener que generar programas elaborados íntegramente en un lenguaje distinto al HTML. Ofrece una amplia gama de funciones para la explotación de BD de forma sencilla.

Entre sus características resaltan la velocidad, estabilidad, seguridad y simplicidad.

- Velocidad: provee gran velocidad de ejecución y no crea demoras en la máquina, ya que no consume muchos recursos del sistema operativo.
- Estabilidad: PHP utiliza su propio sistema de administración de recursos y dispone de un sofisticado método de manejo de variables, conformando un sistema robusto y estable.
- Seguridad: el sistema debe poseer protecciones contra ataques, por lo que permite diferentes niveles de seguridad, estos pueden ser configurados desde el propio archivo.
- Simplicidad: se les debe permitir a los programadores generar código productivamente en el menor tiempo posible.

Sin dejar de mencionar además que es libre, lo que implica menos costes y servidores más baratos que otras alternativas. Es muy rápido. Su integración con la BD MySQL y el servidor Apache, le permite

constituirse como una de las alternativas más atractivas del mercado. Su librería estándar es realmente amplia, lo que permite reducir los llamados "costes ocultos", uno de los principales defectos de ASP. Tiene una de las comunidades más grandes en Internet, con lo que no es complicado encontrar ayuda, documentación, artículos, noticias, y más recursos.

Por todas lo anteriormente expuesto se ha decido utilizar para la implementación del GEDEP como lenguaje de programación PHP.

### 1.6.4 Symfony: marco de trabajo para PHP.

Symfony es un framework diseñado para optimizar el desarrollo de las aplicaciones web mediante algunas de sus principales características.

Algo a destacar es que logra separar la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. Desarrollado en PHP5 y multiplataforma, incorpora además el patrón MVC, soporta AJAX y un gran número de BD.

Con una década ya de experiencia, Symfony está inspirado en los mejores frameworks de su tiempo, como Mojavi, Propel, Prado y Ruby on Rails, y utiliza lo mejor de cada uno de ellos.

#### 1.6.4.1 Propel.

Propel es un framework programado en PHP5 de persistencia de datos y consulta, lo que significa que brinda un mecanismo para almacenar objetos PHP en una BD y un sistema para búsqueda y restauración de objetos PHP desde una BD, todo esto a partir de un mecanismo de abstracción de los datos y utilizando ORM. Propel permite realizar consultas complejas y manipulación de datos sin escribir una sola cláusula SQL. Hace más fácil la escritura de aplicaciones, el despliegue y mucho más migrar la aplicación a otro gestor de BD si alguna vez la situación lo amerita. La última versión estable es la 1.2 y corre sobre PHP 5.0.4.

Propel está dividido en dos componentes principales:

- Un motor generador para construir sus clases y archivos SQL (generador-propel).

- Un ambiente de ejecución que proporciona herramientas para construir consultas SQL, ejecutando consultas compiladas, y herramientas para el manejo de conexiones para múltiples BD simultáneamente.

Propel viene integrado con el framework Symfony, y este a su vez permite a los desarrolladores la opción de utilizarlo o no. En caso positivo tiene las herramientas necesarias para a partir de un motor de generación crear el archivo XML necesario para el ORM de la BD a partir de la lectura de la misma. Esta ventaja quita de encima la compleja tarea de tener que escribir el fichero de configuración a mano cada vez que sea necesario crear un nuevo modelo, haciendo mucho más fácil esta tarea.

#### 1.6.4.2 ExtJS.

ExtJS es un framework de JavaScript que permite realizar aplicaciones web enriquecidas basándose en tecnología AJAX, JSON, DHTML y DOM. Ext 2.0 está patentado bajo licencia LGPL lo que posibilita su uso para aplicaciones empresariales privativas de código cerrado, brinda además, la posibilidad de utilizar un gran número de componentes visuales que mejoran considerablemente la calidad de las aplicaciones. Brinda la posibilidad de validaciones de formularios de todo tipo, basándose en expresiones regulares y tipos de datos.

La utilización de un framework de JavaScript como ExtJS facilita la separación de las capas de la vista con la del controlador desde el punto de vista productivo ya que el código utilizado en la primera es solamente JavaScript y no es necesario utilizar ningún tipo de código PHP.

### 1.6.5 Gestores de Base de Datos.

**PostgreSQL.**

PostgreSQL es un servidor de BD relacional orientado a objetos, libre, publicado bajo la licencia BSD. Es la alternativa más cercana a MySql cuando se precisa de operaciones avanzadas como transacciones, procedimientos almacenados, vistas, o de una BD que soporte gran cantidad de información.

Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una sola compañía sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales las cuales trabajan en su desarrollo. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

PostgreSQL es uno de los servidores más utilizado por todos aquellos programadores que realizan aplicaciones Cliente - Servidor, complejas o críticas. Soporta distintos tipos de datos; además del soporte

para los tipos base, también soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes (MAC, IP...), cadenas de bits, así como el uso de índices, reglas y vistas. También permite la creación de tipos propios.
Permite la declaración de funciones propias, así como la definición de disparadores, la gestión de diferentes usuarios y los permisos asignados a cada uno de ellos, incluye herencia entre tablas.
Entre las negativas que contempla se puede mencionar la gran cantidad de recursos del sistema que consume, al punto de ser dos o tres veces más lento que MySQL.

**MySQL.**

Es un sistema de gestión de BD libre, multihilo y multiusuario. Rápido en la lectura cuando utiliza el motor no transaccional MyISAM, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación.

**SQL Server.**

SQL Server es un sistema de BD muy completo y potente. Posee una gran velocidad y soporta un volumen de datos muy grande. También, presenta mecanismos que le permiten realizar sentencias complicadas, lo que lo hace perfectamente adecuado para aplicaciones críticas y con cualquier grado de complejidad. Por otro lado reserva una parte de la BD para guardar el registro de transacciones con los comandos pendientes, lo que asegura que independientemente de que el programador use o no transacciones en su código, en ningún caso la BD quedaría en un estado inconsistente debido a una ejecución parcial de comandos. Posee el inconveniente que corre solamente sobre una plataforma, Windows, además de ser un software privativo con un alto costo.

**Oracle.**

Es considerado el SGBD más completo que existe. Sus características más destacadas son el soporte de transacciones, su gran estabilidad y seguridad, su escalabilidad, además es un sistema multiplataforma. Su elevado precio hace que sólo se vea en empresas muy grandes y multinacionales en forma general. En el desarrollo de páginas web pasa lo mismo ya que como es un sistema muy caro no está tan extendido como otras BD, por ejemplo, Access, MySQL, SQL Server, entre otras.

A pesar de consumir gran cantidad de recursos del sistema, por todas las ventajas que brinda frente a otros gestores de BD, se ha seleccionado como gestor de BD para la aplicación PostgreSQL, por ser de gran potencia y libre.

### 1.6.6 Servidor Web Apache.

Un servidor web es un programa que implementa el protocolo HTTP (HyperText Transfer Protocol). Este protocolo pertenece a la capa de aplicación del modelo OSI (Open System Interconnection: Modelo de Interconexión de Sistemas Abiertos) y está diseñado para transferir hipertextos, páginas web o páginas HTML (HyperText Markup Language).

Apache es un servidor web HTTP de código abierto y multiplataforma. Es modular, extensible, gratuito, y popular, lo que hace que en ocasiones se hace muy sencillo conseguir ayuda y soporte.

### 1.6.7 TortoiseSVN.

**TortoiseSVN.**

Es un cliente gratuito de código abierto para el sistema de control de versiones Subversion. Se encuentra bajo la licencia GNU GPL. Maneja ficheros y directorios a lo largo del tiempo. Los ficheros se almacenan en un repositorio central. El repositorio es prácticamente lo mismo que un servidor de ficheros ordinario, salvo que recuerda todos los cambios que se hayan hecho a sus ficheros y directorios. Esto permite que pueda recuperar versiones antiguas de sus ficheros y examinar la historia de cuándo y cómo cambiaron sus datos, y quién hizo el cambio. Es fácil de usar, permite ver el estado de los archivos desde en el explorador de Windows y permite también crear gráficos de todas las revisiones asignadas.

## 1.7 Conclusiones.

Se realizó un pequeño estudio sobre las diversas metodologías de desarrollo de software, así como las herramientas, tecnologías y conceptos concernientes a estilos y patrones de la AS, en pos de seleccionar los idóneos a aplicar en la propuesta arquitectónica para el GEDEP.

## CAPÍTULO 2: GEDEP: CARACTERÍSTICAS DEL SISTEMA

### 2.1 Introducción.

En el presente capítulo se ofrece una breve descripción de la situación actual presente en la UCI por la que surge la necesidad de un sistema de GD para la gestión del EP, se detalla el modelo del dominio para el GEDEP y se exponen además los requisitos, tanto funcionales como no funcionales, en los que básicamente estará regido todo el proceso de desarrollo de la arquitectura del mismo, así como la descripción de los casos de uso del sistema que se evaluaron como críticos o arquitectónicamente significativos.

### 2.2 Breve descripción de la situación actual.

Con el fin de establecer un punto común entre la documentación que compone a un EP, ya sea respecto a los diferentes campos de información que estas plantillas contienen o a características propias de formato que se tengan en cuenta a la hora de introducir información de acuerdo a un estándar con respecto a ello, se propone la arquitectura para un sistema que permita la gestión de dicha documentación, pues:

- No se ha establecido un estándar genérico para el formato de la información que se recrea en las plantillas del EP.
- Hay plantillas que no satisfacen las expectativas y necesidades de un proyecto, pues cuentan con campos que resultan innecesarios o porque no se encuentran en el EP que el nuevo proyecto debe llevar.
- En los proyectos noveles las personas con poca experiencia no tienen un sitio que pueda permitir consultas a la hora de construir los artefactos del EP.
- La Dirección de Calidad de Software (DCS) no tiene conocimiento del avance del EP de un proyecto hasta que los artefactos le llegan directamente o se hacen revisiones y auditorías en el marco del mismo.

Dadas estas circunstancias, el GEDEP propone brindar una solución a estos problemas, tributando así a un incremento en el control y la calidad con la que se generen los artefactos y en general del EP.

### 2.3 Descripción del sistema propuesto.

La propuesta que se hace para mediante un sistema de GD remediar los problemas de gestión de la documentación que compone un EP para la UCI, es la de una aplicación web con una BD de trasfondo

soportada en PostgreSQL que almacena la integridad de los documentos del EP, con el fin de que estos estén disponibles cuando sea necesario su uso por algún líder de proyecto u otro rol que tenga que interactuar con alguna plantilla o documento específico en un momento determinado. Igualmente, en caso que se necesite hacer consultas técnicas por parte de miembros de distintos proyectos productivos a los documentos que ya han sido debidamente escritos y salvados, estos podrán realizarlas sin presentar dificultad.

Los estilos arquitectónicos propuestos para la implementación de este sistema son: jerarquía de capas, MVC y Cliente – Servidor, estos dos últimos entrelazados para complementar sus funciones. El sistema tendrá como lenguaje para su implementación PHP, utilizando además el framework Symfony y teniendo como IDE de desarrollo el Zend Studio for Eclipse.

La solución estará diseñada para ofrecer al personal de la DCS y a los equipos de los diferentes proyectos productivos un servicio que les permita gestionar las diferentes plantillas que componen el EP, así como visualizar el contenido del EP en su totalidad.

## 2.4 Definición del dominio.

Teniendo en cuenta que los datos que ofrece el cliente inicialmente no son suficientemente amplios para desarrollar el modelado del negocio, se hará la descripción a través del modelo de dominio.

Un modelo de dominio captura los tipos más importantes de objetos que existen o los eventos que suceden en el entorno donde estará el sistema. Se considera que un escenario apropiado para esta alternativa es aquel donde el objetivo primario es la gestión y presentación de información, tales como sistemas de gestión de órdenes y sistemas bancarios.

Resulta ser una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio del problema así como conceptos del mundo real, no de los componentes de software.

### 2.4.1 Definición de las entidades y los conceptos principales.

En el marco de la captura de requisitos desarrollada con la DCS se lograron identificar y definir las principales entidades o conceptos para lograr la gestión del EP en cada uno de los proyectos productivos de la UCI. Así mismo se lograron definir las principales relaciones existentes entre las mismas.

**Proyecto:**

Elemento organizativo a través del cual se gestiona el desarrollo de software. El resultado de un proyecto es una versión de un producto. Un proyecto contendrá datos significativos que lo identifiquen del resto de los proyectos productivos que se estén desarrollando. Un proyecto posee un EP.

**Expediente de proyecto:**

Entiéndase por EP el compendio de todos los documentos que se obtienen durante el ciclo de vida de un proyecto, pasando por las fases de iniciación, elaboración, construcción y transición, incluyendo una descripción detallada de cada proceso y su ficha de identificación. Un EP está compuesto por varias plantillas y pertenece a un único proyecto.

**Plantilla:**

Una plantilla es una forma de dispositivo que proporciona una separación entre la forma o estructura y el contenido, es un diagrama o documento para crear contenido. A partir de las plantillas se obtendrán los artefactos generados en los diferentes flujos de trabajo ingenieriles. Es un medio o un instrumento que permite guiar, portar o construir un diseño o esquema predefinido.

**Personas:**

Las personas son integrantes de un proyecto de producción de software y ocupan diversos roles, entre ellos figuran los arquitectos, desarrolladores, ingenieros de prueba, y el personal de gestión que les da soporte. Son las encargadas de llenar las diferentes plantillas a lo largo de todo el transcurso del proceso de desarrollo.

**Componente:**

Un componente será el que contendrá el contenido de la plantilla, entiéndase por ejemplo: gráficos, campos de textos, tablas, imágenes. Una plantilla contiene varios componentes y un mismo componente puede estar contenido en varias plantillas.

Los diagramas que se muestran a continuación reflejan la interacción entre el trabajador del dominio y las entidades definidas, así como la relación que existe entre estas últimas.

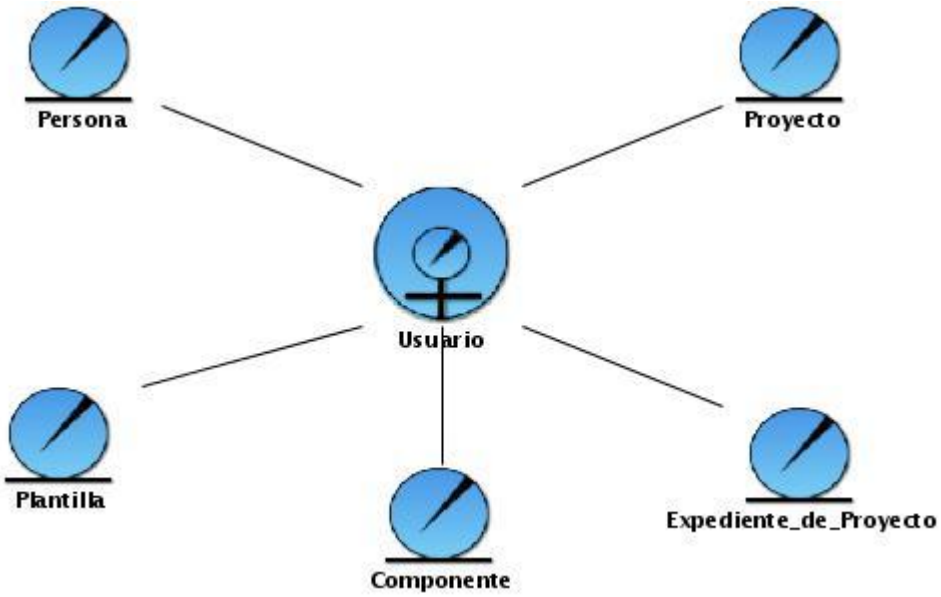


**Figura 1 Modelo de objetos.**

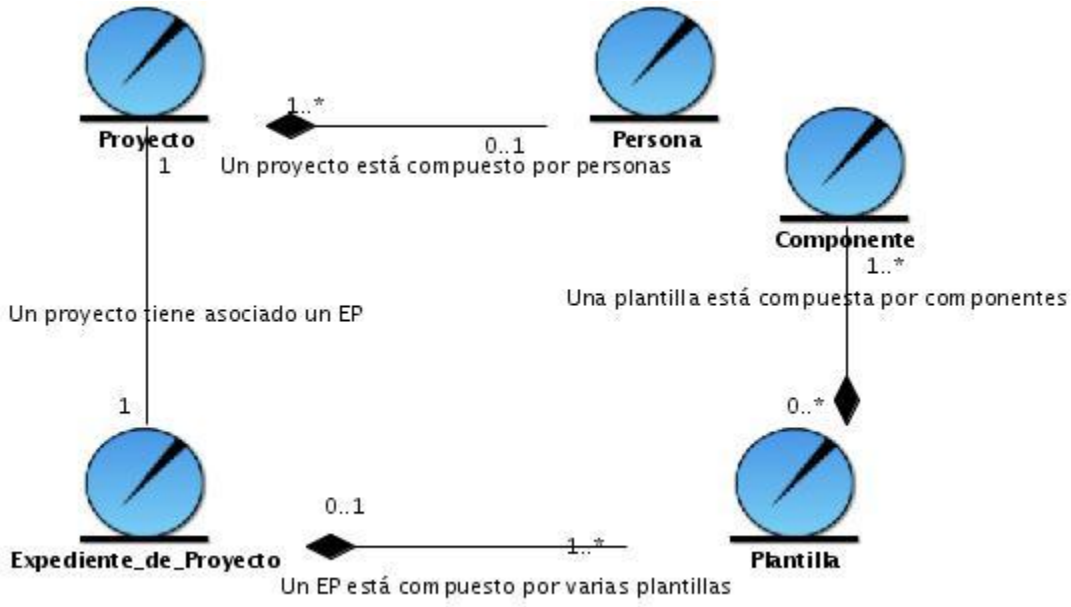


**Figura 2 Relación entre entidades.**

### 2.4.2 Representación del modelo de dominio.

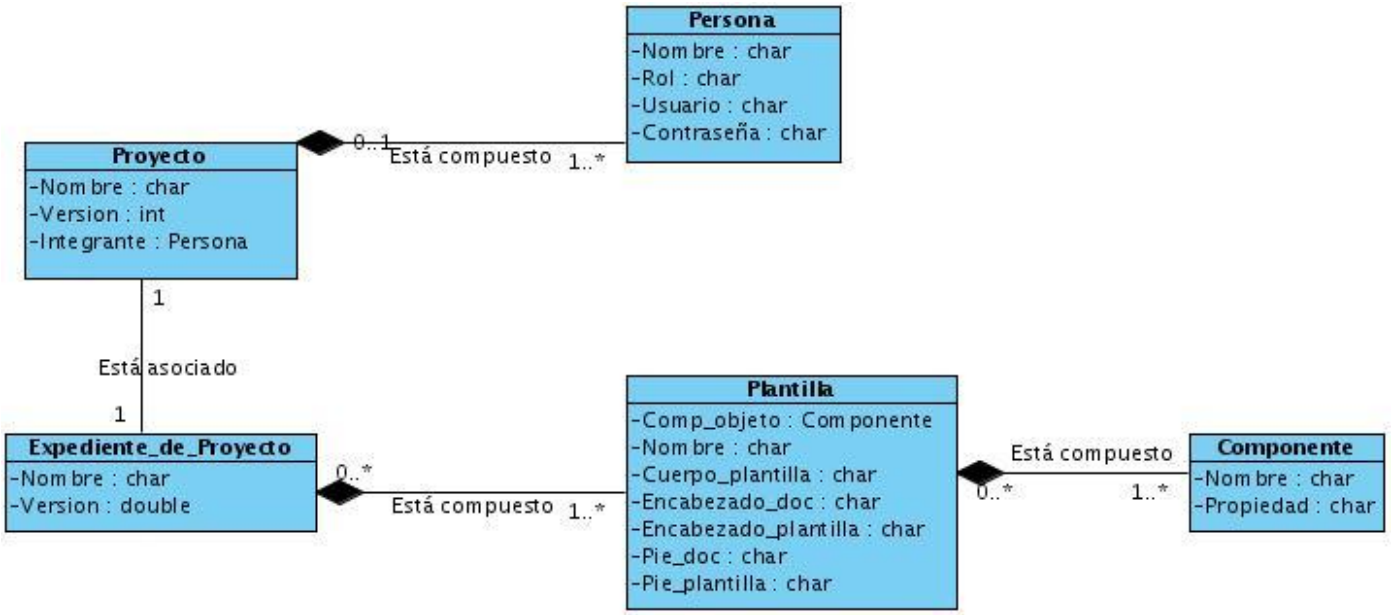


**Figura 3 Modelo de dominio.**

## 2.5 Especificación de los requisitos del sistema.

La Ingeniería de Requisitos (IR) cumple un primordial papel en el proceso de producción de software, ya que se enfoca en la definición de lo que se desea producir. Su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente y compacta, el futuro comportamiento del sistema; de esta manera, se pretende minimizar el diámetro de problemas que puedan ocurrir relacionados al desarrollo de sistemas, al tener especificado de forma clara lo que el cliente desea.

### 2.5.1 Requisitos funcionales.

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir específicamente y en su finalidad no alteran la funcionalidad del producto, esto quiere decir que se mantienen invariables sin importar con qué propiedades o cualidades se relacionen.

Respecto a este tema, y tras la captura de requisitos con el cliente, los que se definieron fueron desglosados teniendo en cuenta las necesidades de la DCS y el personal del proyecto respectivamente,

con el fin de tener una mejor visión del sistema a desarrollar. Siendo así, los mismos se enumeran a continuación:

R1: Gestionar plantilla.

- R1.1: Crear plantilla.
  - R1.1.1: Gestionar contenidos de plantilla.
    - R1.1.1.1: Adicionar contenido.
    - R1.1.1.2: Modificar contenido.
    - R1.1.1.3: Eliminar contenido.
- R1.2: Modificar plantilla.
- R1.3: Eliminar plantilla.
- R1.4: Buscar plantilla.

R2: Crear expediente de proyecto.

R3: Revisar proyecto.

R4: Visualizar expediente de proyecto.

R5: Controlar versiones del expediente de proyecto.

R6: Exportar expediente de proyecto.

R7: Buscar expediente de proyecto.

R8: Gestionar artefacto.

- R8.1: Crear artefacto.
- R8.2: Modificar artefacto.
- R8.3: Eliminar artefacto.
- R8.4: Visualizar artefacto.

R9: Autenticar usuario.

R10: Gestionar permisos de usuario.

- R10.1: Adicionar usuario.
- R10.2: Modificar usuario.
- R10.3: Eliminar usuario.

R11: Comunicar con herramienta de gestión de proyectos IP.

### 2.5.2 Requisitos de referencia del dominio.

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Estos forman una parte significativa de la especificación. Son importantes para que clientes y usuarios puedan valorar las características no funcionales del producto.

Diversas son las aplicaciones que pudieran desarrollarse en condiciones semejantes al dominio referido en este caso, destacando el uso de patrones similares u otras condiciones que pudieran crear cierta compatibilidad a la hora de proponer una solución ante un problema dado. Dichos problemas pueden llegar a converger en requisitos comunes o estándares y que serán especificados posteriormente.

#### 2.5.2.1 Software:

Deberá garantizarse el uso de navegadores web libres ya sea Mozilla Firefox, IceWeasel, Opera, Konqueror, Netscape Navigator, u otro de este tipo.

#### 2.5.2.2 Hardware:

Se necesitan como requerimientos mínimos una PC con procesador Pentium II o superior.

#### 2.5.2.3 Apariencia o interfaz externa:

El sistema deberá mostrar una interfaz cómoda y amigable para cualquier tipo de usuario, evitando así que estos encuentren algún tipo de dificultad en ella. Contará además, con una estructura de carpetas que será muy cómoda para todos los usuarios, facilitando el acceso en el caso de las plantillas que compondrán el EP. Todos los textos aparecerán en idioma español.

#### 2.5.2.4 Seguridad:

Dado que es uno de los requerimientos que puede provocar la mayor cantidad de riesgos, en la aplicación se hace necesario manejar de forma adecuada la seguridad, tratando de forma simultánea la confidencialidad, la integridad y la disponibilidad de los datos que serán manejados por los distintos usuarios y que pudiera afectar la integridad del EP, para lograrlo estos se deberán autenticar debidamente y accederán así al espacio de información permisible para su nivel de accesibilidad, garantizando la protección ante acciones no autorizadas. La autenticación de los usuarios estará validada por LDAP.

#### 2.5.2.5 Usabilidad:

Estos requerimientos describen los niveles apropiados de usabilidad según los usuarios finales del producto, por tanto teniendo en cuenta que el sistema será usado por personas con conocimientos informáticos y asumiendo que sus niveles de experiencia, sean acordes con la tarea a desempeñar, se considera que:

- No requiere de un proceso de aprendizaje muy extenso para los usuarios.
- Debe proporcionar un incremento de la capacidad de controlar un proyecto de software.
- Debe presentar consistencia en la interfaz de usuario.
- Debe presentar un documento donde se expliquen las principales funcionalidades del sistema y cómo ser usado.

#### 2.5.2.6 Soporte:

Se requiere un servidor de BD que soporte grandes volúmenes de información y velocidad de procesamiento, con un tiempo de respuesta rápido ante llamadas concurrentes. Por parte del cliente se requiere un navegador capaz de interpretar código JavaScript. Versión de PHP 5.0 o superior.

#### 2.5.2.7 Restricciones en el diseño y en la implementación:

Se debe realizar la implementación del servicio web en PHP y para el tratamiento de la BD se utiliza PostgreSQL. Se debe utilizar para la modelación del sistema RUP 2003 y para el modelado como herramienta de apoyo a la metodología Visual Paradigm Suite.

#### 2.5.2.8 Portabilidad:

El sistema podrá ejecutarse en entornos basados GNU/Linux siguiendo los principios del software libre y de independencia tecnológica. Además podrá ser instalado y usado en Windows teniendo en cuenta que tanto el lenguaje de programación como las herramientas y el gestor de BD son multiplataforma.

### 2.6 Modelación del sistema.

De acuerdo a lo explicado anteriormente en la descripción y conceptualización del sistema que se propone, se definen las principales funcionalidades de este en casos de uso (CUS).

Para poder determinar qué CUS se desarrollarán en cada release de construcción, RUP propone clasificarlos de acuerdo al impacto que tienen para la arquitectura en:

- Críticos: Más importantes para los usuarios porque cubren las principales tareas o funciones que el sistema ha de realizar. Definen la arquitectura básica.
- Secundarios: Sirven de apoyo a los CUS críticos, involucran funciones secundarias y tienen un impacto más modesto sobre la arquitectura, a pesar de ello deben implementarse pronto porque responden a requerimientos de interés para los usuarios.
- Auxiliares: No son claves para la arquitectura y completan CUS críticos o secundarios.
- Opcionales: Responden a funcionalidades que pueden o no estar en la aplicación, pero que no son imprescindibles en las primeras versiones.

Siendo así se definieron como CUS críticos:

- Gestionar permisos de usuario.
- Autenticar usuario.
- Gestionar plantilla.
- Gestionar contenido.
- Crear expediente proyecto.
- Gestionar artefacto.

CUS secundarios:

- Visualizar expediente.
- Exportar expediente de proyecto.
- Controlar versiones de expediente.
- Buscar expediente de proyecto.
- Guardar expediente.
- Guardar plantilla.

- Guardar artefacto.
- Realizar auditorías.
- Realizar revisiones técnicas formales.
- Comunicar con la aplicación.

### 2.6.1 Actores del sistema.

El actor del sistema no es otro sino un tercero fuera del sistema que interactúa con él, para este caso se define:

| Nombre del actor | Descripción |
|---|---|
| Usuario | Dentro del sistema posterior a su autenticación, solamente tendrá permiso de lectura en los diversos EP. |
| Revisor | El revisor tendrá acceso a todos los EP, pues será el encargado de revisarlos para llevar a cabo las auditorías o las revisiones técnicas formales (RTF) a cada uno de los diferentes proyectos productivos de la UCI. |
| Trabajador_DCS | Es un usuario que se crea dentro de la jerarquía para simplificar la comprensión del sistema, de acuerdo a sus permisos es el encargado de gestionar las plantillas que componen el expediente de proyecto y su contenido respectivo ya sea para el EP que propone la DCS o el que adecúa el líder de proyecto de acuerdo a sus necesidades. Además es el encargado de visualizar dicho expediente en cada caso. |
| Admin_EP | Será el encargado de gestionar un EP que sirva de base para los proyectos de la UCI, además proveerá al mismo de las plantillas necesarias y asignará permisos a los líderes de proyecto dentro del sistema para que estos puedan realizar cambios dentro del EP según sus necesidades. Se encargará de visualizar el EP de cada proyecto para llevar un control de calidad del mismo y estará al tanto de los cambios que sean realizados en ese marco. |

| | |
|---|---|
| Directivo_DCS | Será el encargado de gestionar la comunicación entre el GEDEP y la aplicación de gestión de proyectos en la dirección. |
| Lider_Proyecto | Tiene como responsabilidad establecer los permisos dentro de los integrantes del proyecto para que interactúen con el sistema de acuerdo a su rol en el proceso de desarrollo del software. Crea el EP propio de su proyecto tomando como base el que se propone por la DCS y adecuándolo a sus necesidades reales. |
| Desarrollador_proyecto | Este actor es el que de forma directa interactúa con las plantillas del EP para generar los artefactos correspondientes a cada flujo de trabajo. |

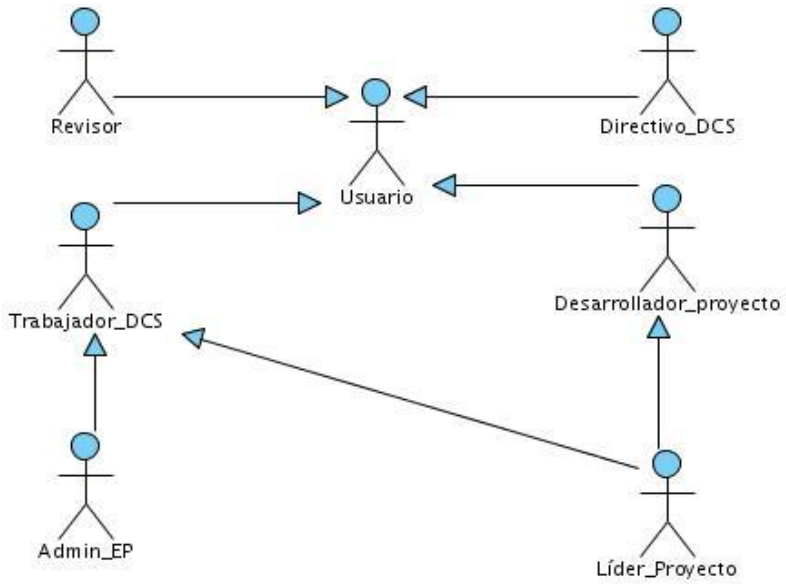


**Figura 4 Relación entre actores.**

### 2.6.2 Diagramas de casos de uso del sistema.

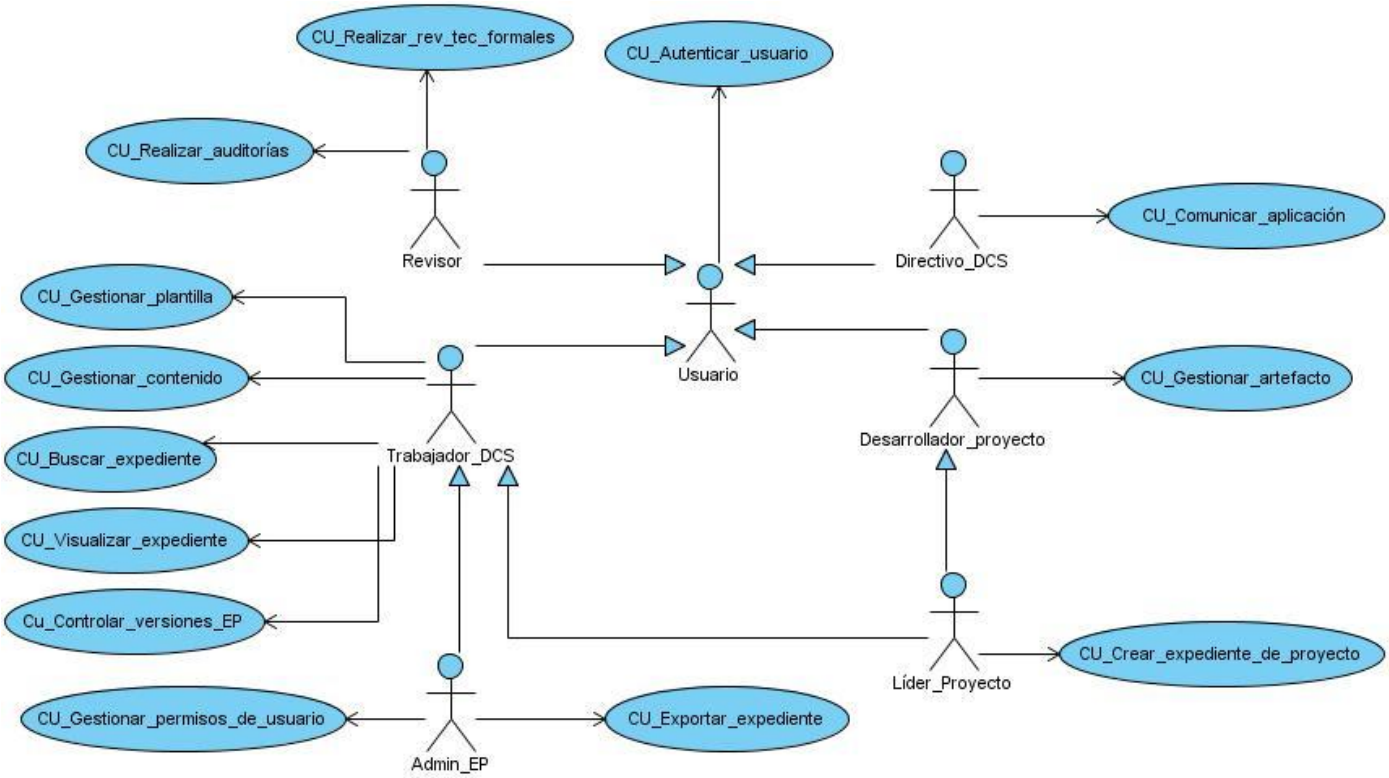


**Figura 5 Diagrama de CUS.**

### 2.6.3 Descripción de CUS.

| **Descripción del caso de uso** | |
|---|---|
| **Nombre del caso de uso** | Gestionar_permisos_de_usuario. |
| **Objetivo** | Permitir al Admin_EP mantener el control sobre los usuarios del sistema y establecer sus respectivos permisos. |
| **Actores** | Admin_EP (inicia). |

| | |
|---|---|
| **Resumen** | El CUS inicia cuando el Administrador del sistema accede al mismo para realizar alguna gestión sobre los usuarios, entiéndase adicionar un nuevo usuario, eliminar o modificar los permisos de un usuario existente. El CUS finaliza con la realización de alguna de estas acciones. |
| **Precondiciones** | Para modificar o eliminar el usuario, este debe existir en el sistema con sus respectivos permisos y relaciones. |
| **Pos condiciones** | Si se desea insertar, se crea la instancia del usuario, si se desea modificar, quedan modificados los atributos del usuario, si la acción que se realizó fue la de eliminar, el usuario será eliminado y ya no constará en el listado de estos. |
| **Referencias** | R9, R10. |

| **Curso normal de los eventos.** | |
|---|---|
| **Acción del actor** | **Respuesta del sistema** |
| 1. El Admin_EP selecciona la acción a realizar. | 2.<br>a) Si elige la opción adicionar usuario ver sección “Adicionar usuario”.<br>b) Si elige la opción de modificar los usuarios existentes ver sección “Modificar usuarios”.<br>c) Si elige la opción eliminar usuario ver sección “Eliminar usuario”. |
| **Sección “Adicionar usuario”** | |
| 1. El Admin_EP introduce los datos completos del nuevo usuario y sus respectivos permisos de acuerdo al rol que ocupa y presiona el botón Aceptar. | 2. El sistema valida los datos de entrada. |
| | 3. El sistema muestra un mensaje de confirmación satisfactorio y regresa al paso 1 mostrando el listado de los usuarios actualizado debidamente. |

| | |
|---|---|
| **Flujo alternativo "Cancelar adicionar usuario"** | |
| 1. a) El Administrador introduce los datos en el sistema y presiona el botón Cancelar. | 1. b) El sistema cancela la operación y retorna al paso 1. |
| 1. c) El Administrador introduce datos incorrectos o incompletos y presiona el botón Aceptar. | 1. d) El sistema emite un mensaje de error al respecto y regresa al paso 1. |
| **Sección "Modificar usuario"** | |
| | 1. El sistema muestra los datos del usuario que se desea modificar dando la opción de hacer cambios en los datos del formulario. |
| 2. El Admin_EP introduce los nuevos cambios en el perfil del usuario y presiona el botón Aceptar. | 3. El sistema valida los datos de la entrada. |
| | 4. El sistema muestra un mensaje de confirmación satisfactorio y regresa al paso 1 mostrando el listado de los usuarios actualizado debidamente. |
| **Flujo alternativo "Cancelar modificar usuario"** | |
| 2. a) El Administrador introduce los nuevos datos y presiona el botón Cancelar. | 2. b) El sistema cancela la operación y mantiene los datos antiguos, volviendo al paso 1. |
| **Sección "Eliminar usuario"** | |
| | 1. El sistema muestra un mensaje de confirmación para esta acción. |
| 2. El Admin_EP presiona el botón Aceptar. | 3. El sistema retorna al paso 1 y muestra el listado actualizado, o sea sin el usuario que se eliminó anteriormente. |

| | |
|---|---|
| **Prioridad** | Crítico. |
| **Prototipo** | |

| **Descripción del caso de uso** | | |
|---|---|---|
| **Nombre del caso de uso** | Autenticar_usuario. | |
| **Objetivo** | Autenticar el usuario en el sistema. | |
| **Actores** | Usuario (inicia). | |
| **Resumen** | El CUS se inicia cuando el cliente se autentica en el sistema para entrar al mismo con su nombre de usuario y contraseña accediendo una vez concluido el proceso e autenticación a módulos restringidos del sistema. El sistema comprueba los datos de entrada permitiendo el acceso para finalizar así el CUS. | |
| **Precondiciones** | El usuario debe existir en la BD del sistema. | |
| **Pos condiciones** | El usuario puede acceder a un área restringida. | |
| **Referencias** | R9. | |
| **Curso normal de los eventos.** | | |
| **Acción del actor** | | **Respuesta del sistema** |
| 1. El usuario introduce su nombre de usuario y contraseña y presiona el botón Entrar. | | 2. El sistema valida los datos de entrada, verificando la existencia del usuario en la BD. |
| | | 3. El sistema permite al usuario acceder a un área según sus permisos. |
| **Flujo alternativo “Datos con errores o incompletos”** | | |
| **Acción del actor** | | **Respuesta del sistema** |
| 1. a) El usuario introduce datos incompletos o incorrectos y presiona el botón Entrar. | | 1. b) El sistema valida los datos de entrada y muestra un mensaje de error aclaratorio y retorna al paso 1. |

| | |
|---|---|
| 1. c) Usuario inexistente en la BD. | 1. d) El sistema no permite el acceso al usuario. |
| **Prioridad** | Crítico. |
| **Prototipo** | |

| **Descripción del caso de uso** | | |
|---|---|---|
| **Nombre del caso de uso** | Gestionar_plantilla. | |
| **Objetivo** | Permitir a un usuario con privilegios gestionar las plantillas que contendrá el EP. | |
| **Actores** | Trabajador_DCS (inicia). | |
| **Resumen** | El CUS inicia cuando un trabajador de la DCS accede al sistema para gestionar las plantillas del EP. El CUS finaliza con la adición, modificación o eliminación de una de estas. | |
| **Precondiciones** | Para poder gestionar plantillas el usuario debe tener los permisos requeridos para ejecutar la acción. | |
| **Pos condiciones** | La plantilla se guarda en el sistema una vez que se crea por primera vez, en caso de ser modificada conserva los cambios respectivos, si es eliminada la plantilla desaparece de la BD y se actualiza el listado de estas que se le muestra al cliente del sistema. | |
| **Referencias** | R1 | |
| **Curso normal de los eventos.** | | |
| **Acción del actor** | | **Respuesta del sistema** |
| 1. El usuario selecciona la opción a realizar. | | 2.<br>a) Si elige la opción adicionar plantilla ver sección “Adicionar plantilla”.<br>b) Si elige la opción de listar plantillas ver sección “Listado de plantillas”. |
| **Sección “Adicionar plantilla”** | | |

| | |
|---|---|
| | 1. El sistema muestra los diferentes componentes de la plantilla y un espacio destinado a su creación. |
| 2. El usuario crea una nueva plantilla para el EP, introduciendo los componentes que esta contendrá. Elije la dirección en la que ubicará la plantilla dentro del EP y selecciona el botón Guardar. | 3. El sistema muestra el listado de las plantillas actualizado y regresa al paso 2. |
| **Flujo alternativo "Cancelar adicionar plantilla"** | |
| 2. a) El usuario crea la plantilla, introduce los componentes que esta contendrá y presiona el botón Cancelar. | 2. b) El sistema cancela la operación y regresa al paso 1. |
| **Sección "Listar plantillas"** | |
| 1. El usuario selecciona el botón Ver todas: (Mi expediente/Ver todas). | 2. El sistema muestra el listado de las plantillas existentes con las dos posibles acciones a llevar a cabo: modificar o eliminar. |
| 2. El usuario selecciona la acción que llevará a término. | 3.<br>a) Si elige modificar ver la sección "Modificar plantilla".<br>b) Si elige eliminar ver la sección "Eliminar plantilla". |
| **Sección "Modificar plantilla"** | |
| | 1. El sistema muestra la plantilla selecciona por el usuario con sus respectivos atributos o contenidos. |
| 2. El usuario realiza los cambios pertinentes. Selecciona la opción | 3. El sistema actualiza el contenido de la plantilla y el listado de plantillas, |

| | |
|---|---|
| Guardar. | regresando al paso 2. |
| **Flujo alternativo "Cancelar modificar plantilla"** | |
| 2. a) El usuario realiza las modificaciones en la plantilla y selecciona la opción Cancelar. | 2. b) El sistema cancela la opción de modificar la plantilla y regresa al paso 1 del flujo normal de eventos. |
| **Sección "Eliminar plantilla"** | |
| | 1. El sistema muestra un mensaje de confirmación al usuario para esta acción. |
| 2. El usuario ejecuta la opción Aceptar. | 3. El sistema actualiza el listado de las plantillas regresando al paso 2 de la sección "Listar plantillas". |
| **Prioridad** | Crítico. |
| **Prototipo** | |

| **Descripción del caso de uso** | |
|---|---|
| **Nombre del caso de uso** | Gestionar_contenido. |
| **Objetivo** | Permitir a un usuario con privilegios gestionar los contenidos que tendrán las plantillas que sean creadas como parte del EP. |
| **Actores** | Trabajador_DCS (inicia). |
| **Resumen** | El CUS inicia cuando un trabajador de la DCS accede a una plantilla previamente creada para gestionar los contenidos que esta contendrá. |
| **Precondiciones** | Para poder gestionar los contenidos el usuario debe tener los permisos requeridos para ejecutar la acción y haber seleccionado la plantilla en la que se efectuarán los cambios. |
| **Pos condiciones** | La plantilla queda en el sistema con los cambios respectivos, en caso de eliminar pues la plantilla se borra del contenido del EP. |
| **Referencias** | R1, R1.1.1 |
| **Curso normal de los eventos.** | |

| Acción del actor | Respuesta del sistema |
|---|---|
| 1. El usuario selecciona la opción a realizar. | 2.<br>a) Si elige la opción adicionar contenido ver sección “Adicionar contenido”.<br>b) Si elige la opción de listar contenidos ver sección “Listar contenidos”. |
| **Sección “Adicionar contenido”** | |
| 1. El usuario adiciona el contenido a la plantilla. Selecciona la opción Aceptar. | 2. El sistema muestra el listado de contenidos de la plantilla y regresa al paso 2. |
| **Flujo alternativo “Cancelar adicionar contenido”** | |
| 1. a) El usuario adiciona el contenido a la plantilla y presiona el botón Cancelar. | 1. b) El sistema cancela la operación y regresa al paso 2. |
| **Sección “Listar contenidos”** | |
| | 1. El sistema muestra el listado de los diferentes contenidos existentes en una plantilla con las dos posibles acciones a llevar a cabo: modificar o eliminar. |
| 2. El usuario selecciona la acción que llevará a término. | 3.<br>a) Si elige modificar ver la sección “Modificar contenido”.<br>b) Si elige eliminar ver la sección “Eliminar contenido”. |
| **Sección “Modificar contenido”** | |
| | 1. El sistema muestra el contenido seleccionado por el usuario. |
| 2. El usuario realiza los cambios pertinentes en el contenido. | 3. El sistema actualiza el contenido y el listado de contenidos, |

| | |
|---|---|
| Selecciona la opción Aceptar. | regresando al paso 1 de la sección “Listar contenidos”. |
| **Flujo alternativo “Cancelar modificar contenido”** | |
| 1.a) El usuario realiza las modificaciones en el contenido y selecciona la opción Cancelar. | 2.b) El sistema cancela la acción de modificar el contenido y regresa al paso 1 de la sección “Listar contenidos”. |
| **Sección “Eliminar contenido”** | |
| 1. El usuario selecciona el contenido que desea eliminar. | 2. El sistema muestra un mensaje de confirmación al usuario para esta acción. |
| 3. El usuario ejecuta la opción Aceptar. | 4. El sistema actualiza el listado de los contenidos regresando al paso 1 de la sección “Listar contenidos”. |
| **Prioridad** | Crítico. |
| **Prototipo** | |

| **Descripción del caso de uso** | |
|---|---|
| **Nombre del caso de uso** | Crear_expediente_de_proyecto. |
| **Objetivo** | Crear el expediente para un proyecto productivo que comienza su proceso de desarrollo. |
| **Actores** | Lider_proyecto (inicia) |
| **Resumen** | El CUS comienza cuando el líder de proyecto selecciona entre las plantillas gestionadas, las que le son necesarias en su proceso de desarrollo de software y confecciona así su EP, dentro del cual estarán dichas plantillas y los artefactos que se generen sobre la base de estas. En caso de que las plantillas no sean suficientes puede crear una nueva plantilla con su contenido. |

| | |
|---|---|
| **Precondiciones** | Tener los permisos requeridos para acceder a esta área del sistema. Se deben haber creado las plantillas. |
| **Pos condiciones** | Queda creado el EP para ese proyecto productivo en el sistema. |
| **Referencias** | R2 |

| **Curso normal de los eventos.** | |
|---|---|
| **Acción del actor** | **Respuesta del sistema** |
| 1. El usuario selecciona la opción “Nuevo expediente”. | 2. El sistema muestra el formulario a completar con los datos identificativos del expediente. |
| 3. El usuario completa los datos del formulario y selecciona la dirección en la que se guardará su EP. Presiona el botón “Guardar EP”. | 4. El sistema guarda el nuevo expediente de proyecto inicialmente sin ninguna plantilla o artefacto. Ver sección “Configurar expediente”. |
| **Flujo alternativo “Cancelar”** | |
| 3. a) El usuario presiona el botón Cancelar. | 3. b) El sistema cancela la operación y regresa al paso 1. |
| **Sección “Configurar expediente”** | |
| 1. El usuario selecciona la opción “Configurar expediente” para seleccionar las plantillas que formarán parte de su EP. | 2. El sistema muestra el listado de las plantillas en el EP base. |
| 3. El usuario selecciona las plantillas que desea como parte de su EP y presiona el botón Aceptar. | 4. El sistema muestra un listado actualizado de las plantillas existentes como parte de su EP. |
| **Flujo alternativo “Cancelar configurar expediente”** | |
| 3. a) El usuario selecciona las plantillas y presiona el botón Cancelar. | 3. b) El sistema cancela la operación y regresa al paso 1 del flujo normal de eventos. |
| **Prioridad** | Crítico |

| | |
|---|---|
| **Prototipo** | |

| **Descripción del caso de uso** | |
|---|---|
| **Nombre del caso de uso** | Gestionar_artefacto. |
| **Objetivo** | Permitir a un desarrollador la gestión de los diferentes artefactos que serán generados dentro del EP. |
| **Actores** | Desarrollador_proyecto (inicia). |
| **Resumen** | El CUS inicia cuando el desarrollador encargado del artefacto, una vez ya dentro del sistema accede a una plantilla de su EP y puede crear, modificar o eliminar artefactos que se generan sobre la base de las plantillas que ya han sido gestionadas. |
| **Precondiciones** | Para poder gestionar los artefactos el desarrollador debe encontrarse en "Mi expediente", tener los permisos requeridos para poder hacer las modificaciones requeridas en su área de trabajo. Se deben haber definido las plantillas que compondrán el EP y los contenidos que estas tendrán. |
| **Pos condiciones** | El artefacto se crea a partir de una plantilla ya como resultado de un flujo de trabajo en el proceso de desarrollo. |
| **Referencias** | R8 |
| **Curso normal de los eventos.** | |
| **Acción del actor** | **Respuesta del sistema** |
| 1. El usuario selecciona la opción Adicionar artefacto. | 2. El sistema muestra el listado de las plantillas existentes en el EP. |
| 3. El usuario selecciona la plantilla a partir de la cual generará el | 4. El sistema muestra la plantilla seleccionada ofreciendo la |

| | |
|---|---|
| artefacto. Presiona el botón Nuevo artefacto. | posibilidad de editar los diferentes campos de la misma. |
| 5. El usuario selecciona el botón Editar campo. | 6. El sistema muestra la página de edición del campo. |
| 7. El usuario introduce los nuevos datos y presiona el botón Aceptar. | 8. El sistema regresa al paso 4. |
| 9. Al finalizar el proceso de edición de la plantilla el usuario presiona el botón Aceptar. | 10. El sistema guarda el artefacto en la BD y muestra el listado actualizado de los artefactos dando la opción de modificar o eliminar respectivamente.<br>a) Ver sección "Modificar artefacto".<br>b) Ver sección "Eliminar artefacto". |
| **Flujo alternativo "Cancelar"** | |
| 7.a) El usuario introduce los nuevos datos y presiona el botón Cancelar. | 7.b) El sistema cancela la acción y regresa al paso 4. |
| **Sección "Modificar artefacto"** | |
| | 1. El sistema muestra el artefacto seleccionado por el usuario ofreciendo la posibilidad de editar los diferentes campos del mismo. |
| 2. El usuario selecciona el botón Editar campo. | 3. El sistema muestra la página de edición del campo. |
| 4. El usuario realiza los cambios pertinentes en el artefacto. Selecciona la opción Aceptar. | 5. El sistema actualiza el campo en el artefacto. |
| 6. El usuario presiona el botón Aceptar. | 7. El sistema regresa paso 10. |
| **Flujo alternativo "Cancelar modificar artefacto"** | |

| | |
|---|---|
| 4.a) El usuario realiza las modificaciones en el artefacto y selecciona la opción Cancelar. | 4.b) El sistema cancela la opción de modificar el artefacto. Y regresa al paso 1. |
| **Sección "Eliminar artefacto"** | |
| | 1. El sistema muestra un mensaje de confirmación al usuario para esta acción. |
| 2. El usuario ejecuta la opción Aceptar. | 3. El sistema actualiza el listado de los contenidos regresando al paso 10. |
| **Prioridad** | Crítico. |
| **Prototipo** | |

**2.7 Conclusiones.**

En este capítulo se ha presentado una breve descripción de la propuesta del sistema, se han identificado los principales conceptos o entidades en los que se soporta el dominio del mismo y se han descrito las funcionalidades de este. Quedaron detallados además los requisitos tanto funcionales como no funcionales. Se identificaron los CUS, obteniendo el diagrama de estos y las descripciones de los críticos para la arquitectura del sistema.

## CAPÍTULO 3: ARQUITECTURA DEL SISTEMA

### 3.1 Introducción.

La AS está compuesta por la estructura de los elementos de un programa o sistema, sus interrelaciones y los principios o reglas que gobiernan su diseño y evolución a lo largo del tiempo. Muestra una visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo más importantes para su construcción, los cimientos que son necesarios como base para comprenderlo y desarrollarlo.

En este capítulo se define la línea base de la arquitectura del sistema, así como los diferentes patrones y estilos arquitectónicos que la conforman.

### 3.2 Línea base de la arquitectura.

El IEEE 610.12/1990 define una línea base como una especificación o producto que se ha revisado formalmente y sobre los que se ha llegado a un acuerdo, de ahí en adelante sirve como base para un desarrollo posterior y que puede cambiarse solamente a través de procedimientos formales de control de cambios.

La línea base de la arquitectura es el esqueleto del producto que se desarrollará, es la primera visión que se tiene de lo que será, reflejando todos los modelos y diagramas que son generados desde el flujo de trabajo de modelo de negocio hasta diseño, obteniendo como resultado una aplicación ejecutable que responde a los CUS que la comprometen. En ella se describen los estilos y patrones arquitectónicos que se utilizan en la aplicación. Su alcance dentro del sistema a desarrollar está descrito en la estructura organizativa que se le da al mismo en función de los propios estilos y patrones que se empleen en su implementación.

#### 3.2.1 Alcance.

La línea base de la arquitectura describe la estructura del sistema en un nivel superior de abstracción, así mismo se hace cargo de detallar el organigrama de la arquitectura según los estilos arquitectónicos que se han de utilizar.

Entre los pasos que diversas fuentes han definido, para determinar una línea base figuran:

- Definir la infraestructura con la que se cuenta, los dispositivos, sistemas operativos, soporte, entre otros, e identificar las propiedades del sistema global.

- Seleccionar los patrones arquitectónicos adecuados con el tipo de aplicación que se ha de desarrollar, el tipo de infraestructura con la que se cuenta, o las diferentes tecnologías. Posteriormente se ha de lograr la integración entre todos los patrones que se han seleccionado.
- Identificar los subsistemas funcionales utilizando los métodos del análisis y diseño convencionales en función de la distribución que se escogió desarrollar, e integrarlos con la infraestructura del sistema.

Estos pasos definen en cierta manera el alcance de la línea base. En capítulos anteriores se han identificado algunas de las características del sistema, se propone por tanto la utilización de los patrones y estilos que serán utilizados, en pos de dar cumplimiento a las diversas restricciones del sistema.

## 3.3 GEDEP: Estilos y patrones para su arquitectura.

Los estilos arquitectónicos son una generalización y abstracción de los patrones de diseño. Caracterizan una familia de sistemas que están relacionados por compartir propiedades estructurales y funcionales.

Los patrones, por su parte, se definen como una solución estándar para un problema común de programación, una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios, un lenguaje de programación de alto nivel y una forma más práctica de describir ciertos aspectos de la organización de un programa y las conexiones entre componentes de este.

Optar por una forma arquitectónica no solamente tiene valor distintivo, sino que define una situación pragmática.

### 3.3.1 Cliente – Servidor.

La arquitectura Cliente - Servidor es un modelo para el desarrollo de sistemas de información, en el que las transacciones se dividen en procesos independientes que cooperan entre sí para intercambiar información, servicios o recursos. Los principales componentes de este esquema son entonces los clientes, los servidores y la infraestructura de comunicaciones.

Un cliente es el que inicia un requisito de servicio. El requisito inicial puede convertirse en múltiples requisitos de trabajo. La ubicación de los datos o de las aplicaciones es totalmente transparente para el cliente, este frecuentemente se comunica con procesos auxiliares que se encargan de establecer conexión con el servidor, enviar el pedido, recibir la respuesta, manejar las fallas y realizar actividades de sincronización y de seguridad.

Los servidores, por su parte, proporcionan un servicio al cliente y devuelven los resultados. En algunos casos existen procesos auxiliares que se encargan de recibir las solicitudes del cliente, verificar la protección, activar un proceso servidor para satisfacer el pedido, recibir su respuesta y enviarla al cliente. Además, deben manejar los interbloqueos, la recuperación ante fallas, y otros aspectos afines.

Para que los clientes y los servidores puedan comunicarse se requiere una infraestructura de comunicaciones, la cual proporciona los mecanismos básicos de direccionamiento y transporte.

Entre las principales características de esta arquitectura, se pueden destacar las siguientes:

- El servidor presenta a todos sus clientes una interfaz única y bien definida.
- El cliente no necesita conocer la lógica del servidor, sólo su interfaz externa.
- El cliente no depende de la ubicación física del servidor, ni del tipo de equipo físico en el que se encuentra, ni de su sistema operativo.
- Los cambios en el servidor implican pocos o ningún cambio en el cliente.

En este caso, el cliente será el especificado en los requerimientos no funcionales. Por la parte servidora se contará con el servidor de BD PostgreSQL, por sus características vistas en capítulos anteriores.

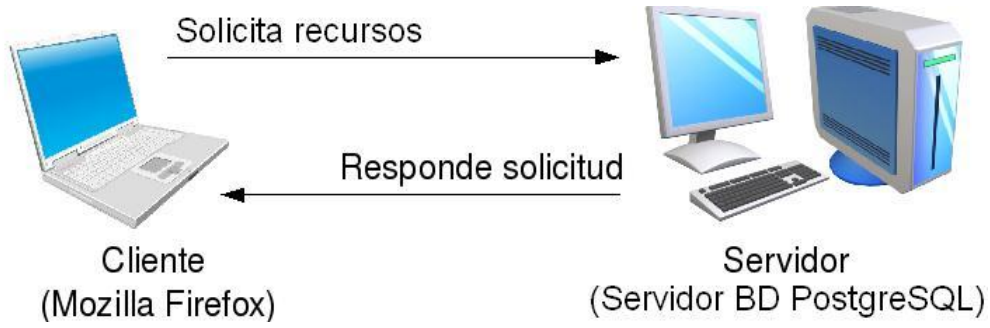


**Figura 6 Modelo Cliente – Servidor.**

### 3.3.2 Arquitectura en capas.

La arquitectura en capas, como se explicó en capítulos anteriores, es uno de los estilos que se utiliza con mayor frecuencia en el desarrollo de aplicaciones web por su propia naturaleza. Se ha definido como una estructura jerárquica donde cada capa proporciona servicios a la superior y se nutre de las prestaciones de la inmediatamente inferior. Para este estilo el número mínimo de las capas es dos, sin embargo en este caso se propone trabajar sobre la propuesta de la jerarquía en tres capas, siendo estas: Presentación, Lógica de negocio y Acceso a datos respectivamente.

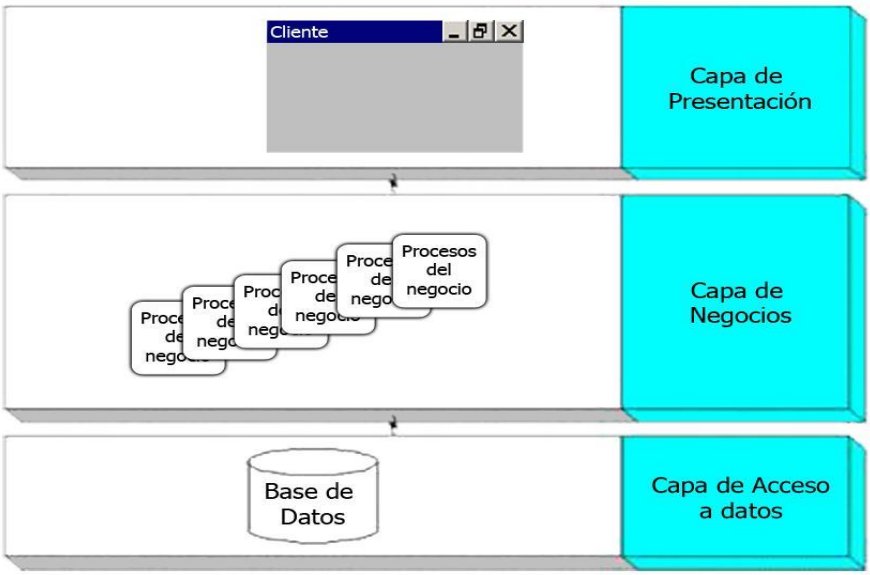


**Figura 7 Jerarquía tres capas.**

Entre las ventajas de este estilo figuran: ofrecer soporte de diseño basados en niveles de abstracción creciente, admitir de manera muy naturalmente las optimizaciones y los refinamientos, además de proporcionar amplias reutilizaciones.

Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capas estándares, a partir de las cuales se pueden construir extensiones o prestaciones específicas.

La capa de Presentación es la encargada de obtener la información del usuario, enviarla a los servicios de negocios para su procesamiento, recibir los resultados del mismo y presentarlos al usuario. Contará por tanto con las interfaces necesarias para que fluya de manera estable la comunicación entre el usuario y el sistema, estas estarán contenidas en la lógica de presentación. El cliente proporcionará el contexto de presentación, mediante un navegador que permite ver los datos remotos a través de una capa de presentación HTML.

La capa Lógica de negocio es el “puente” entre un usuario y los servicios de datos. Responde a peticiones del usuario (u otros servicios de negocio) para ejecutar una tarea de este tipo. Una tarea de negocio es una operación definida por los requisitos de la aplicación. Esta capa es la encargada por tanto de recibir la

entrada del nivel de presentación, interactuar con los servicios de datos para ejecutar las operaciones de negocio y enviar el resultado procesado al nivel de presentación.

En este caso, almacenará las clases controladoras que representen algún tipo de comportamiento para la aplicación según los CUS. Guardará además, todas las transformaciones y procesos del sistema.

La capa de Acceso a datos se encarga de manipular toda la información que tiene implícito entre sus funciones el soporte de interfaces de acceso a datos (Objeto de Acceso Datos: DAO por sus siglas en inglés), está formada por la Interfaz de Datos y por el Manipulador de Datos, este último es el encargado de interactuar con la BD para realizar todos los procesos de persistencia. Extrae también los datos de la BD y los convierte en objetos persistentes propios del sistema. La interfaz de datos no es más que el conjunto de las clases persistentes de las cuales sus objetos transitan por las demás capas de la aplicación (preferentemente la capa del negocio).

Añadido a estos patrones arquitectónicos podemos encontrar como parte del framework que será utilizado para el desarrollo de la aplicación el patrón MVC, parte de la familia de los patrones de diseño. En ocasiones se pudiera llegar a pensar que estos patrones (Jerarquía tres capas y MVC) son totalmente afines, incluso puede surgir la confusión en ciertas personas que lo ven como el mismo patrón, algo que es erróneo. Sin embargo pudieran implementarse de modo que uno llegase a enriquecer o fortalecer al otro. Si vemos lo siguiente desde un entorno práctico y cotidiano como por ejemplo el ámbito militar, se puede llegar a definir el patrón de arquitectura Jerarquía tres capas como una estrategia que se puede utilizar de forma general, en tanto el MVC viene siendo como la maniobra táctica que se pone en práctica en una batalla específica y en un momento determinado.

### 3.3.3 Modelo Vista Controlador (MVC) según Symfony.

En capítulos anteriores se abordaba el tema del MVC y parte de sus ventajas, aspectos considerables para la aplicación del mismo en la aplicación, además de ser el patrón que implementa el framework que será utilizado por parte del equipo de desarrollo.

Se define como modelo la representación física del dominio, este se encarga además de administrar los comportamientos y datos del propio dominio; por lo tanto se puede decir que la capa de dominio como también se le suele llamar al modelo, añade significado a los datos.

Por su parte la vista maneja la visualización de la información, presentándola en un formato adecuado y cómodo para interactuar con la interfaz de usuario (IU).

El controlador tiene entre sus prioridades interpretar las instrucciones de entrada y la comunicación entre el modelo y la vista, pudiendo llegar a implicar cambios en estos. Dentro de él se encuentran los componentes del lado del servidor.

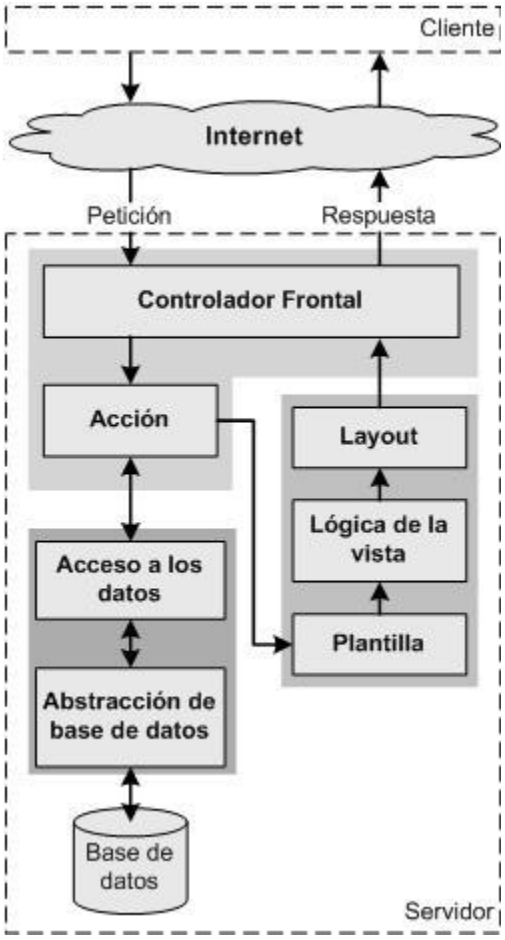


**Figura 8 Modelo-Vista-Controlador en el flujo de trabajo de Symfony.**

En la figura se muestra la especificación de la implementación que realiza Symfony del MVC, pues toma lo mejor de esta arquitectura y lo pone en práctica buscando ofrecer un desarrollo rápido y sencillo para el programador. La parte del modelo de Symfony se puede hacer con la ayuda del ORM Propel. Basado en la descripción de la BD esta genera las clases para los objetos, formularios, y filtros. Propel también concibe las sentencias SQL utilizadas para crear las tablas en la BD. La configuración de la BD se puede

hacer con una tarea o editando un archivo de configuración. Además de su configuración, también es posible hacer la inyección inicial de datos, gracias a los archivos de datos.

De forma predeterminada, la capa de la vista de la arquitectura MVC utiliza archivos PHP planos como plantillas. Estas pueden utilizar helpers para tareas recurrentes como crear una dirección URL o un enlace (link). Una plantilla puede ser decorada por un layout para abstraerse del encabezado y pie de las páginas. Para acelerar las cosas, se puede utilizar el sub-framework del cache para guardar en cache una página entera, la acción, o tan sólo partes o componentes.

La parte del controlador es gestionado por los controladores frontales y las acciones, tal y como se había visto anteriormente en la figura. Las tareas se pueden utilizar para crear módulos que pueden ser CRUD o módulos de administración para las clases del modelo. Estos últimos permiten construir una aplicación completamente funcional sin codificación alguna.

Symfony propone una estructura interna de proyecto para una mejor organización del mismo, dicha estructura agrupa de forma lógica las operaciones dentro de la aplicación y mantiene una estructura organizacional que permite una mejor comprensión y organización del mismo.

Además de la estructura interna que propone, Symfony ordena su directorio raíz y aunque pudiera pensarse que llega a ser excesivo sin embargo está pensado para los proyectos web profesionales, tributando con ello a una mejor organización del código. Esta estructura, por demás no es necesariamente estática, ya que puede cambiar la ubicación de los directorios e incluso el nombre de acuerdo a las especificidades del cliente en cuestión.

```
Apps/
  frontend/
  backend/
batch/
cache/
config/
data/
  sql/
doc/
lib/
  model/
```

log/
plugins/
test/
  unit/
  functional/
web/
  css/
  images/
  js/
  uploads/

#### 3.3.3.1 Controlador frontal.

Todas las peticiones web son manejadas por un solo controlador frontal, que es el punto de entrada único de toda la aplicación en un entorno determinado.

Cuando el controlador frontal recibe una petición, utiliza el sistema de enrutamiento para asociar el nombre de una acción y el nombre de un módulo con la URL escrita o seleccionada por el usuario. Se encarga, entonces, de despachar las peticiones, lo que implica algo más que detectar la acción que se ejecuta. De hecho, ejecuta el código común a todas las acciones. El controlador además:

- Define las constantes del núcleo.
- Localiza la librería de Symfony.
- Carga e inicializa las clases del núcleo del framework.
- Carga la configuración.
- Decodifica la URL de la petición para determinar la acción a ejecutar y los parámetros de la petición.
- Si la acción no existe, re direccionará a la acción del error 404 (El mensaje de error 404 o No encontrado es un código de error HTTP que indica que el navegador web ha sido capaz de comunicarse con el servidor, pero no existe el fichero que ha sido pedido).
- Activa los filtros (por ejemplo, si la petición necesita autenticación).
- Ejecuta los filtros, primera vez.

- Ejecuta la acción y produce la vista.
- Ejecuta los filtros, segunda vez.
- Muestra la respuesta.

El controlador frontal por defecto, llamado index.php se encuentra ubicado en el directorio web/ del proyecto.

#### 3.3.3.2 Acciones.

Las acciones son el corazón de la aplicación, puesto que contienen toda la lógica de la aplicación. Utilizan el modelo y definen variables para la vista. Cuando se realiza una petición web en una aplicación Symfony, la URL define una acción y los parámetros de la petición. Estas son métodos con el nombre executeNombreAccion de una clase llamada nombreModuloActions que hereda de la clase sfActions y se encuentran agrupadas por módulos. La clase que representa las acciones de un módulo se encuentra en el archivo actions.class.php, en el directorio actions/ del módulo.

Para los casos en los que es necesario realizar un grupo de procedimientos antes o después de la llamada de todas las acciones, las clases sfActions y sfAction permiten el uso de los métodos preExecute y postExecute respectivamente. Esto ayuda a la reutilización del código que sea común para un gran grupo de acciones en el caso de las clases que hereden de sfActions y para el caso de las acciones que hereden de sfAction el código que se desee sea ejecutado justo antes o después de la llamada de la acción.

#### 3.3.3.3 Acceso a los datos.

La capa de acceso a datos no contiene funciones dependientes de ningún sistema gestor de BD, por lo que es independiente de la BD utilizada. Además, las funciones creadas en la capa de abstracción de la BD se pueden reutilizar en otras funciones del modelo que necesiten acceder a la BD.

#### 3.3.3.4 Abstracción de la BD.

La capa del modelo se puede dividir en la capa de acceso a los datos y en la capa de abstracción de la BD. De esta forma, las funciones que acceden a los datos no utilizan sentencias ni consultas que dependen de una BD, sino que utilizan otras funciones para realizar las consultas. Así, si se cambia de sistema gestor de BD, solamente es necesario actualizar la capa de abstracción de la BD.

#### 3.3.3.5 Layout.

El layout es un archivo, que también se denomina plantilla global, almacena el código HTML común a todas las páginas de la aplicación, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el layout, o si se mira desde el otro punto de vista, el layout decora la plantilla.

La plantilla global puede ser adaptada completamente para cada aplicación. Se puede añadir todo el código HTML que sea necesario. Normalmente se utiliza el layout para mostrar la navegación, el logotipo del sitio, entre otras. Incluso es posible definir más de un layout y decidir en cada acción el layout a utilizar.

Los archivos del Layout se encuentran en el directorio .../templates/ de cada aplicación, en esta se pueden crear tantos Layout sean necesarios o redefinir el Layout por defecto que crea Symfony para cada caso.

#### 3.3.3.6 Lógica de la vista.

Las páginas web suelen contener elementos que se muestran de forma idéntica a lo largo de toda la aplicación: cabeceras de la página, el layout genérico, el pie de página y la navegación global. Normalmente sólo cambia el interior de la página. Por este motivo, la vista se separa en un layout y en una plantilla. Casi siempre el layout es global en toda la aplicación o al menos en un grupo de páginas. La plantilla sólo se encarga de visualizar las variables definidas en el controlador. Para que estos componentes interaccionen entre sí correctamente, es necesario añadir cierto código: plantilla-vista-layout. La vista en este aspecto es quien contiene las variables de cabecera y título de la página, además de una variable con el contenido de la misma.

#### 3.3.3.7 Plantilla.

La plantilla por su parte es la encargada de contener el cuerpo de la página, las variables que el controlador envía como respuesta a la petición o la consulta realizada al modelo. Lo que quiere decir que es la encargada de contener el código HTML correspondiente a la salida de una acción determinada. El uso de código PHP debe ser el mínimo posible. Sólo llamadas a las variables definidas en la acción o en los objetos sfRequest, sfResponse, sfContext, sfUser y sfFlash o la llamada a los HELPERS. Las plantillas dan respuestas a las acciones, y es el primer eslabón en construir la respuesta que verá el cliente después que se ejecute la acción correspondiente.

## 3.4 Patrones de diseño.

Los patrones de diseño contribuyen a reutilizar diseño, identificando aspectos claves de la estructura que puede ser aplicado a una gran cantidad de situaciones, por lo que reduce de forma notable los esfuerzos de desarrollo y mantenimiento, mejoran además la seguridad, eficacia y consistencia de los diseños y proporcionan un considerable ahorro en la producción.

La importancia de la reutilización del diseño no es despreciable, ya que ésta nos provee de numerosas ventajas: mejorando la flexibilidad, modularidad y extensibilidad, factores internos e íntimamente relacionados con la calidad percibida por el usuario. Estos patrones se dividen en una gran familia compuesta inicialmente por los patrones de creación, estructurales y de comportamiento.

### 3.4.1 Fachada (Facade).

El patrón Fachada es uno de los llamados patrones de diseño que se encuentra ampliamente vinculado con el MVC, por tanto su atención en este sentido es primordial. El Fachada figura entre los llamados patrones estructurales proporcionando una interfaz unificada para un conjunto de interfaces de un subsistema. Define además, una interfaz de alto nivel que hace que el subsistema sea más fácil de usar. La “fachada” satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas. Se presenta básicamente ante el problema de cómo acceder a diversas clases mediante una única clase. Por lo que su propósito es simplificar el acceso a un conjunto de clases o interfaces y proporcionar una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. De esta forma se consigue satisfacer a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que las necesiten. Fachada, se relaciona además con el patrón Controlador, perteneciente a la familia de los GRASP.

Symfony mediante el sistema de configuración de archivos YML implementa este patrón permitiendo acceder a diferentes configuraciones del Framework desde un único lugar. Además de que el acceso a la aplicación es a través del controlador frontal que implementa este patrón.

### 3.4.2 Decorador (Decorator).

El propósito del Decorador es añadir responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la especialización mediante herencia, cuando se trata de añadir funcionalidades, respondiendo este a la siguiente interrogante: ¿cómo añadirle una nueva característica a una clase, pudiendo así obtener gran funcionalidad combinando piezas sencillas?.

Con el uso de este patrón se evita concentrar en lo alto de la jerarquía, clases guiadas por las responsabilidades, es decir, que pretenden satisfacer todas las posibilidades. De esta forma las nuevas funcionalidades se componen de piezas simples que se crean y se combinan con facilidad, independientemente de los objetos cuyo comportamiento extienden.

### 3.4.3 Solitario (Singleton).

El propósito del Solitario es garantizar que una clase sólo tiene una única instancia, proporcionando un punto de acceso global a la misma.

En el controlador frontal ya se ha visto una llamada a sfContext::getInstance(). En una acción, el método getContext() devuelve el mismo *singleton*. Se trata de un objeto muy útil que guarda una referencia a todos los objetos del núcleo de Symfony relacionados con una petición dada, y ofrece un método de acceso para cada uno de ellos:

- sfController: El objeto controlador (->getController())
- sfRequest: El objeto de la petición (->getRequest())
- sfResponse: El objeto de la respuesta (->getResponse())
- sfUser: El objeto de la sesión del usuario (->getUser())
- sfDatabaseConnection: La conexión a la BD (->getDatabaseConnection())
- sfLogger: El objeto para los logs (->getLogger())
- sfI18N: El objeto de internacionalización (->getI18N())

Se puede llamar al singleton sfContext::getInstance() desde cualquier parte del código.

## 3.5 Patrones GRASP.

Los General Responsability Assignment Software Patterns (GRASP) son patrones generales de software para asignación de responsabilidades a objetos expresados en forma de patrones. Aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendables en el diseño de software.

### 3.5.1 Controlador.

El Controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control.

Responde ante el problema: ¿Quién debería ser el responsable de gestionar un evento de entrada al sistema? Proponiendo como vía de solución la asignación de la responsabilidad de recibir o manejar un mensaje de un evento del sistema a una clase que representa una de las opciones siguientes:

- Representa el sistema global, dispositivo o subsistema.
- Representa un CUS en el que tiene lugar el evento del sistema a menudo denominado <nombre del CUS> Manejador, <nombre del CUS> coordinador, <nombre del CUS> Sesión.

Se recomienda dividir los eventos del sistema en el mayor número de controladores para poder aumentar la cohesión y disminuir el acoplamiento, además de utilizar la misma clase controladora para todos los eventos del sistema en el mismo escenario de un CUS.

### 3.5.2 Alta cohesión.

Tiene como objetivo conseguir que la información que maneje una clase sea lo más coherente posible y esté en gran medida relacionada con dicha clase. Por tanto el problema que se plantea se refleja en: ¿Cómo mantener la complejidad manejable?

Como solución propone asignar una responsabilidad de manera que la cohesión permanezca alta entre las clases. Symfony por su parte agrupa las clases por funcionalidades que son fácilmente reutilizables, bien por su uso directo o por herencia.

### 3.5.3 Bajo acoplamiento.

Este patrón se encuentra muy vinculado al Alta Cohesión y el problema que resuelve es: ¿Cómo soportar bajas dependencias, un bajo impacto del cambio e incremento de la reutilización? El problema anterior se soluciona con tan solo asignar una responsabilidad de manera que el acoplamiento permanezca bajo.

### 3.6 Estándar y estilo de codificación.

Los patrones de idiomas son patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto, utilizados en los flujos de trabajo de implementación y despliegue, así como en las diferentes transiciones y versiones que se hagan del producto. Su objetivo es definir de forma clara y establecer un estándar en la codificación de manera que se logre una estabilidad en la escritura y en el entendimiento del producto final.

De esta forma se toma el estilo Allman para tratar la identación a la hora de interactuar con el código en el sistema. Allman propone usar los sangrados para indentar el código, nunca espacios. Poner las llaves de control en la línea subsiguiente.

Se añade un salto de línea después del cierre de los paréntesis de los parámetros y un salto de línea después un punto y coma, cuando termina la sentencia.

Se usan espacios en blanco a ambos lados del operador de símbolos, después de comas y después de las declaraciones, así como en las separaciones de los bloques de código aún en el mismo método.

En cuanto a la longitud de la línea de código, si esta ya presenta una longitud mayor a 80 caracteres se debe cortar y presentar un salto de línea después de una coma, o de un operador, y alinear la nueva línea con el principio de la expresión en el mismo nivel en la línea anterior.

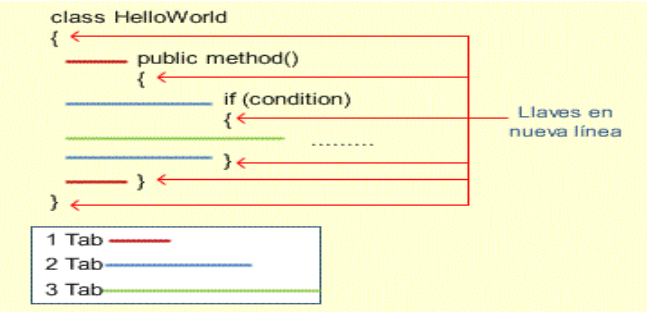


**Figura 9 Identación.**

Para el tratamiento de los nombres de las clases estos comienzan con la palabra genérica clase, seguida del nombre en mayúscula. Ejemplo: class claseNombre. Con respecto al nombre de los métodos se utiliza el patrón Mayúsculas y minúsculas Camel. Este propone que la primera letra del identificador está en minúscula y la primera letra de las siguientes palabras concatenadas en mayúscula. Por ejemplo: "**m**étodo**C**amel".

### 3.7 Descripción de la arquitectura.

La descripción de la arquitectura es el conjunto de modelos que describen la línea base de la arquitectura. El papel de esta descripción es guiar al equipo de desarrollo a través del ciclo de vida del producto y puede llegar a adoptar diferentes formas, ser un extracto (un conjunto de vistas), o una reescritura de los extractos de forma que sea más fácil leerlos.

La descripción de la arquitectura tiene cinco secciones, una para cada modelo: una vista del modelo de CUS, otra del modelo de análisis (opcional), del modelo de diseño, el modelo de despliegue, y finalmente del modelo de implementación.

Cada una de estas vistas se refiere a un conjunto de intereses de diferentes stakeholders (interesados) del sistema y se diseñan mediante un proceso centrado en la arquitectura, motivado por escenarios y desarrollado iterativamente.

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

Teniendo en cuenta las diferentes vistas de la descripción de la arquitectura propuestas por RUP y que la idea del desarrollo del sistema de GD está basada en el uso del framework Symfony se hace necesario el conocimiento de cómo funcionan las aplicaciones construidas con el mismo y fundamentalmente, cómo funcionará en el caso del GEDEP.

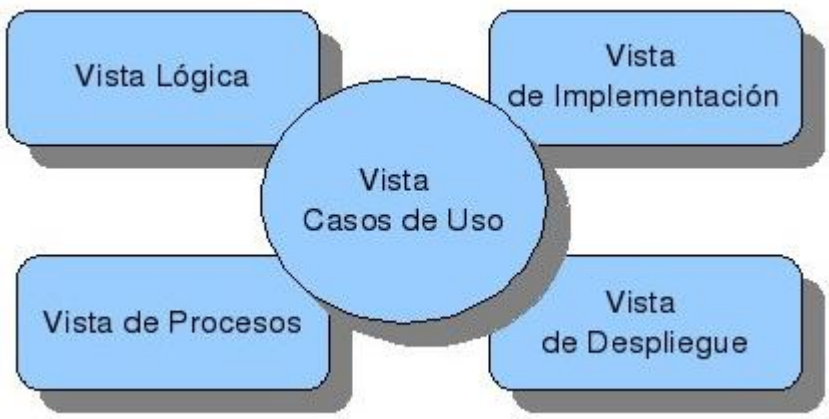


**Figura 10 Vistas de la Arquitectura.**

### 3.7.1 Vista de CUS.

La vista de CUS brinda información de los escenarios y CUS que resultan ser arquitectónicamente significativos, estos son aquellos que sirven para validar la arquitectura propuesta y describen las funcionalidades imprescindibles para el sistema.

#### 3.7.1.1 Módulo de seguridad.

Este módulo se responsabiliza de la gestión de los usuarios que interactuarán con el sistema y de los permisos de accesibilidad que tendrán los mismos para lograr un nivel de seguridad estable.

- CU Autenticar usuario: El CUS se inicia cuando el cliente se autentica en el sistema para acceder con su nombre de usuario y contraseña a módulos restringidos del mismo. El sistema comprueba los datos de entrada permitiendo o no el acceso para finalizar así el CUS.
- CU Gestionar permisos de usuario: El CUS inicia cuando el Administrador del sistema accede al mismo para realizar alguna gestión sobre los usuarios, entiéndase crear un nuevo usuario, eliminar o modificar los permisos de un usuario existente. El CUS finaliza con la realización de alguna de estas acciones.

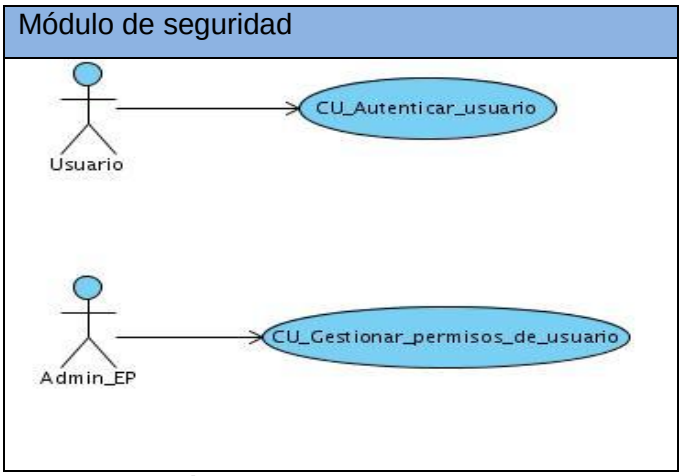


**Figura 11 Módulo de seguridad.**

#### 3.7.1.2 Módulo de gestión base.

Este módulo será el encargado de gestionar las plantillas que formarán parte del EP y los contenidos que estas contendrán respectivamente.

- CU Gestionar plantilla: El CUS inicia cuando un trabajador de la dirección accede al sistema para crear, modificar, eliminar o buscar una plantilla en el EP. El CUS finaliza con la realización de alguna de estas acciones.
- CU Gestionar contenido: El CUS inicia cuando un trabajador de la DCS accede a una plantilla previamente creada ya en el sistema para gestionar los contenidos que esta contendrá, dígase crear, modificar o eliminar contenidos. El CUS termina con la realización de una de estas acciones por parte del actor.

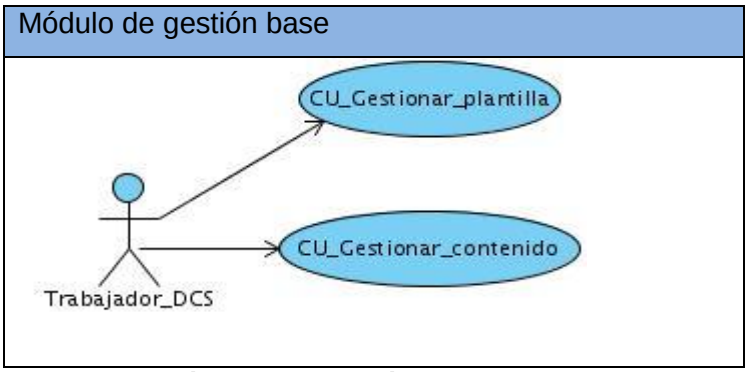


**Figura 12 Módulo de gestión base.**

### 3.7.1.3 Módulo de configuración del proyecto.

Este módulo contará con los CUS que se definieron como críticos y que forman parte de la configuración que debe tener un EP para cada proyecto productivo en específico, en este caso se tienen los anteriores CUS del módulo de gestión además de los propios, ya que cada proyecto puede adecuar las plantillas definidas por la DCS a sus necesidades reales y la gestión de los artefactos que se generan en su ciclo de desarrollo.

- CU Crear expediente de proyecto: El CUS comienza cuando el líder de proyecto selecciona entre las plantillas y contenidos gestionados anteriormente, las que le son necesarias en su proceso de desarrollo de software y confecciona así su propio EP a partir de la base del que propone la DCS, dentro del cual estarán dichas plantillas y los artefactos que se generen sobre la base de estas. En caso de que las plantillas no sean suficientes puede crear una nueva plantilla con su contenido. El CUS finaliza con la culminación satisfactoria de la acción.
- CU Gestión de artefactos: El CUS inicia cuando el desarrollador encargado de generar los artefactos determinados por parte de cada proyecto accede, una vez estando dentro del sistema, a una plantilla de su EP y puede crear, modificar o eliminar artefactos que se generan sobre la base de las plantillas que ya han sido gestionadas.

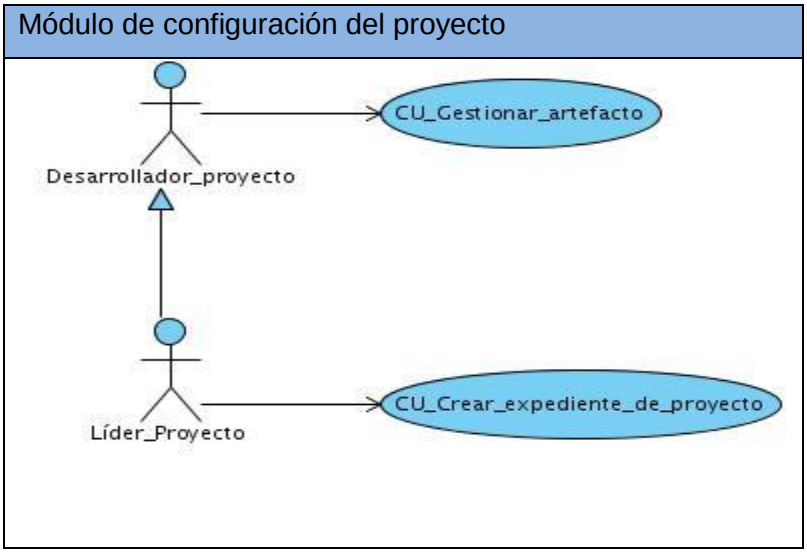


**Figura 13 Módulo configuración del proyecto.**

### 3.7.2 Vista lógica.

Esta vista es propicia para la descripción de las clases más importantes que formarán parte del ciclo de desarrollo del sistema. Se describen también los paquetes más abstractos que lo conforman y las relaciones de dependencia o de uso que existen entre ellos. Esto se hace con el fin de potenciar el análisis funcional e identificar mecanismos y elementos de diseño comunes a diversas partes del mismo.

Posteriormente, con la siguiente figura se pretende mostrar la división que se hace del sistema en módulos, con el objetivo de promover la reusabilidad y facilitar el mantenimiento de este ante cambios en los procesos de negocio, garantizando que sólo se modifique el módulo al que pertenece la funcionalidad afectada por el cambio. También se facilita el trabajo del equipo de desarrollo ya que permite que se puedan desarrollar módulos paralelos, con sus dependencias.

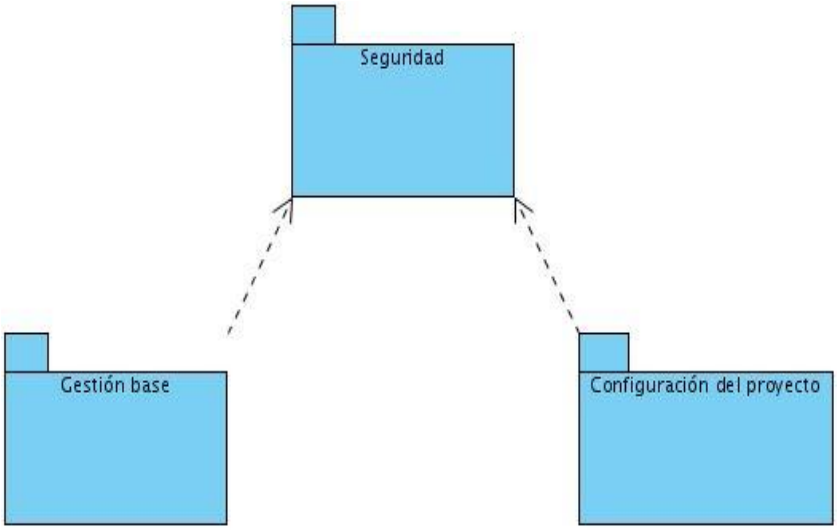


**Figura 14 División del sistema en módulos.**

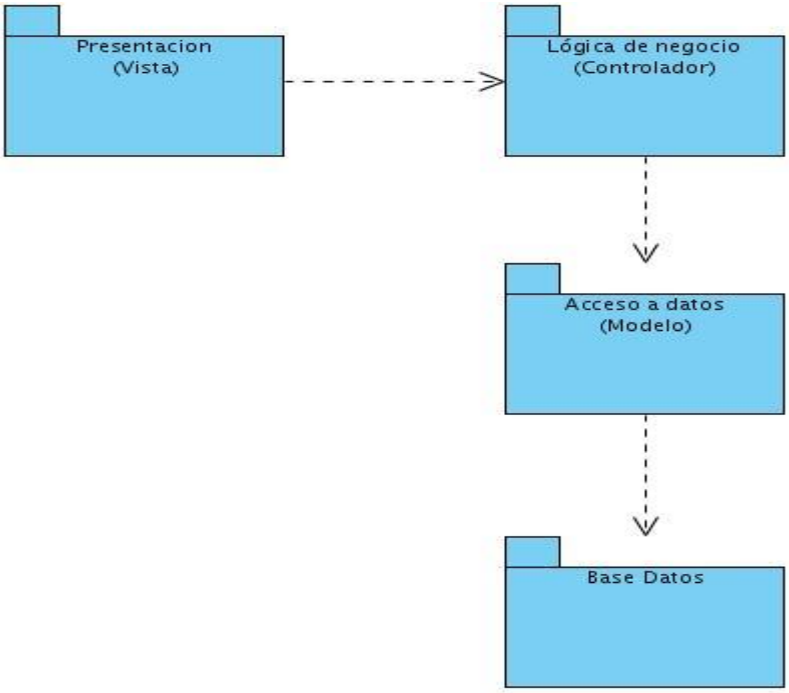


**Figura 15 Distribución de la aplicación por capas según el MVC.**

| **Descripción de los paquetes** | | |
|---|---|---|
| **Nombre** | **Estereotipo** | **Descripción** |
| Presentación (Vista) | Paquete | Conjunto de archivos que componen las páginas clientes del sistema. |
| Lógica de negocio (Controlador) | Paquete | Conjunto de clases y archivos de configuración encargados de gestionar los procesos de la lógica del negocio. |
| Acceso a datos (Modelo) | Paquete | Conjunto de clases que controla el tratamiento de datos. |
| Base Datos | Paquete | Conjunto de tablas relacionales que contienen los datos. |

**Detalles de cada una de las vistas que componen la vista lógica.**

**Capa de Presentación (Vista).**

La capa de presentación queda dividida en dos partes, primeramente se muestra el paquete con la Vista Aplicación que contiene la vista global de la misma y encierra todo lo que es común para ella, dígase los layout, las librerías ExtJS, los archivos CSS así como las librerías y CSS del menú, además de los archivos .js con la presentación ExtJS de cada módulo.

Por su parte la Vista Módulos se encarga de contener las plantillas, donde se cargan los .js con la presentación ExtJS de cada módulo, así como los slot, partial y los componentes.

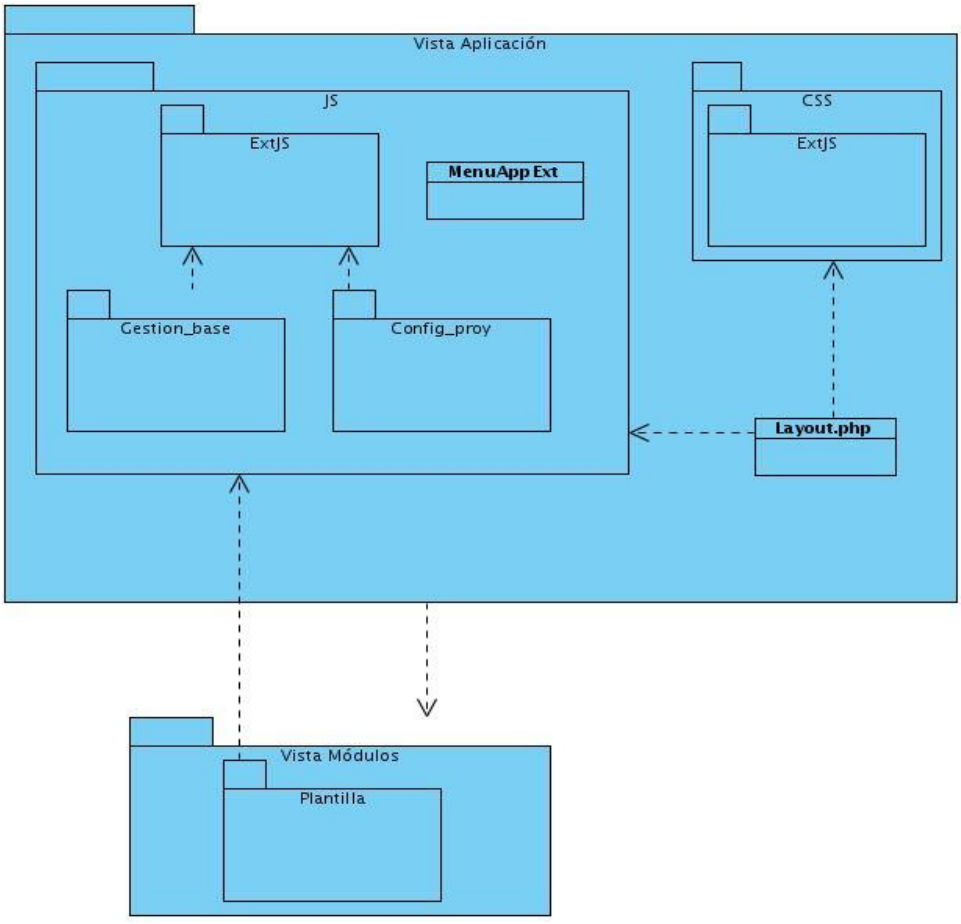


**Figura 16 Descripción en paquetes de la vista.**

| Descripción de la Vista Aplicación | | |
|---|---|---|
| **Nombre** | **Estereotipo** | **Descripción** |
| JS | Paquete | Agrupa los archivos y clases JS que son comunes para toda la aplicación, el fichero _menu.xml del menú y los archivos .js con la presentación de los módulos. |
| CSS | Paquete | Agrupa las hojas de estilo .css común para toda la aplicación. |
| Layout.php | Archivo | Contiene el código común a todas las páginas clientes. |

| Paquete JS de la Vista Aplicación | | |
|---|---|---|
| **Nombre** | **Estereotipo** | **Descripción** |
| ExtJS | Paquete | Librerías y clases del ExtJS. |
| Gestion_base | Paquete | Conjunto de archivos .js con la presentación ExtJS del módulo Gestión base. |
| Config_proy | Paquete | Conjunto de archivos .js con la presentación ExtJS del módulo Configuración del proyecto. |
| MenuAppExt | Fichero | Contiene las funciones para generar el menú ExtJS. |

| Paquete CSS de la Vista Aplicación | | |
|---|---|---|
| **Nombre** | **Estereotipo** | **Descripción** |
| ExtJS | Paquete | Contiene las hojas de estilo de ExtJS (ExtJS_all.js). |

| Descripción de la Vista Módulos | | |
|---|---|---|
| **Nombre** | **Estereotipo** | **Descripción** |
| Plantilla | Paquete | Contiene las plantillas .php en las que se va a cargar la presentación del módulo. |

**Capa Lógica (Controlador).**

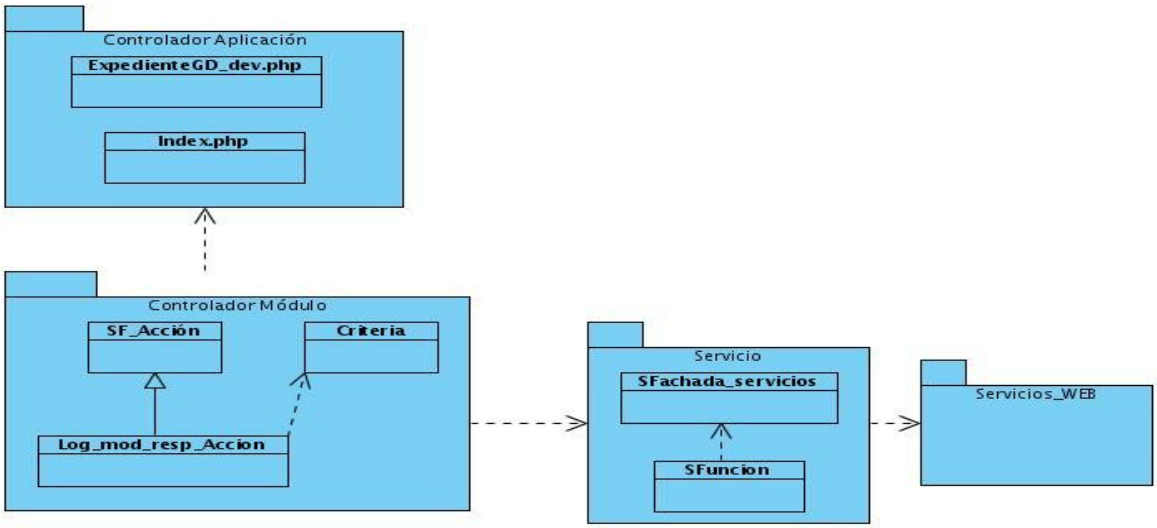


**Figura 17 Descripción en paquetes del controlador.**

La lógica del negocio, en este caso el Controlador, queda dividida en dos partes: el Controlador Aplicación y el Controlador Módulos.

El Controlador Aplicación contiene los controladores frontales que van a ser las vías de entrada y salida de la aplicación y los encargados de cargar todas las configuraciones del sistema.

El Controlador Módulos es quien se va a encargar de desarrollar toda la lógica referente a los módulos.

| **Descripción del Controlador Aplicación** | | |
|---|---|---|
| **Nombre** | **Estereotipo** | **Descripción** |
| Index.php | Archivo | Controlador frontal encargado de gestionar el entorno de producción. |
| ExpedienteGD_dev.php | Archivo | Controlador frontal encargado de gestionar el entorno de desarrollo. |

| **Descripción del Controlador Módulos** | | |
|---|---|---|
| **Nombre** | **Estereotipo** | **Descripción** |
| SF_Acción | Clase | Clase del núcleo de Symfony mediante la cual se |

| | | |
|---|---|---|
| | | accede a los objetos sfRequest, sfView, sfUser más usados. |
| Log_mod_resp_Accion | Clase | Clase responsable de la lógica del negocio del módulo. |
| Criteria | Clase | Clase del núcleo de Symfony empleada para la elaboración de consultas SQL que interactúa con las clases del modelo de Propel. |

| **Descripción de Servicio** | | |
|---|---|---|
| **Nombre** | **Estereotipo** | **Descripción** |
| Sfachada_servicios | Clase | Clase que implementa al patrón Fachada y actúa como tal para gestionar y controlar los servicios web. |
| SFuncion | Clase | Clase de funciones de los servicios web. |

**Capa de acceso a datos (Propel).**

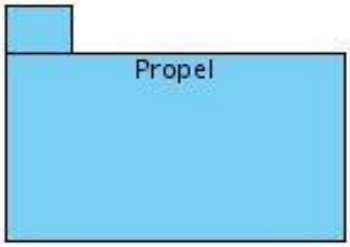


**Figura 18 Descripción en paquetes del modelo.**

Este paquete contiene todas las clases de mapeo de la BD generada por Propel, ya que provee un sistema para almacenar objetos en una BD y un sistema para búsqueda y restauración de objetos desde esta. Además le permite realizar consultas complejas y manipulación de BD sin escribir una sola cláusula SQL. Propel puede ser descrito como un mapeado objeto-relacional, una capa DAO, o una capa objeto persistente.

### 3.7.3 Vista de implementación.

La vista de implementación proporciona una descripción de las principales capas y subsistemas de componentes de la aplicación. También provee una vista de la trazabilidad de los elementos de diseño de la vista lógica en función de la implementación.

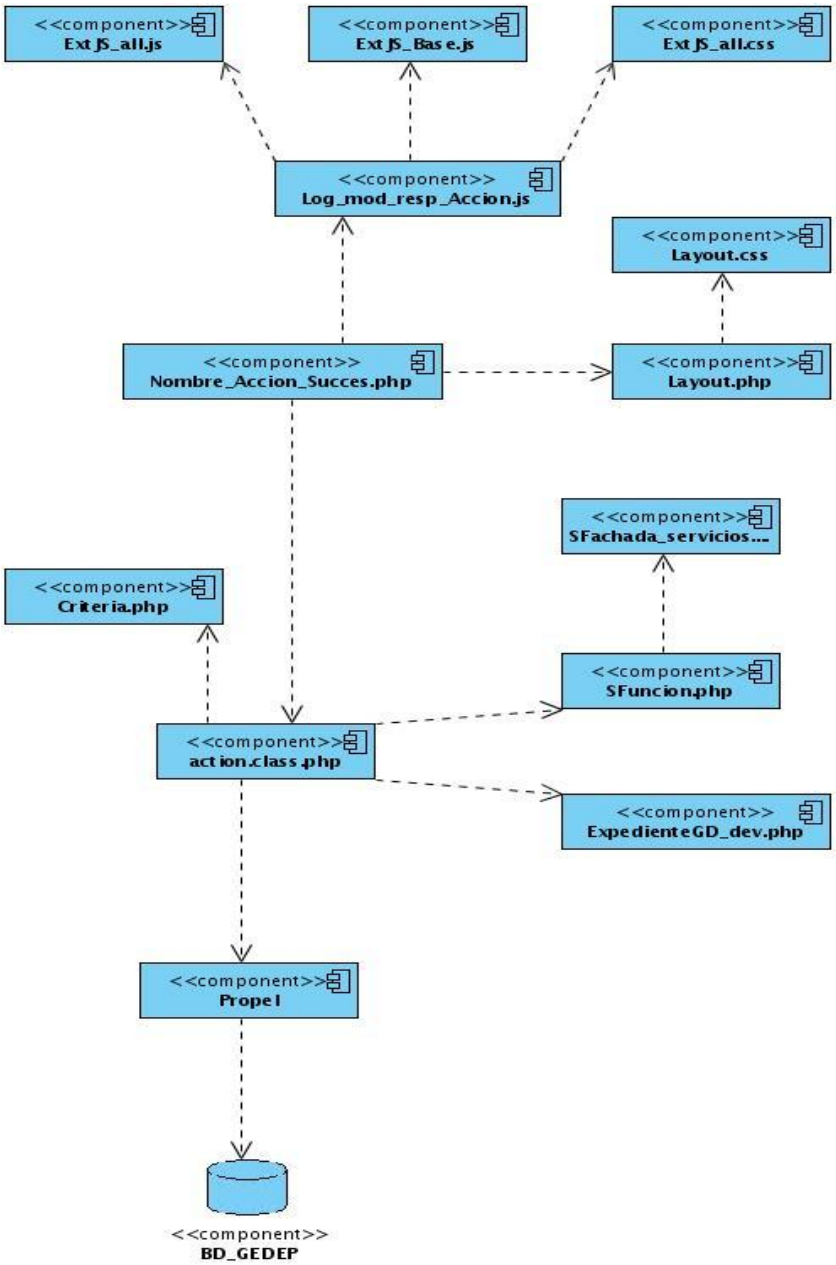


**Figura 19 Descripción en componentes de la vista global de la aplicación.**

| Descripción de los componentes | | |
|---|---|---|
| **Nombre** | **Estereotipo** | **Descripción** |
| ExtJS_all.js, ExtJS_base.js | Fichero | Ficheros con las librerías ExtJS. |
| ExtJS_all.css | Fichero | Ficheros con los estilos css para ExtJS. |
| Layout.css | Fichero | Ficheros con los estilos del layout de la aplicación. |

### 3.7.4 Vista de despliegue.

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

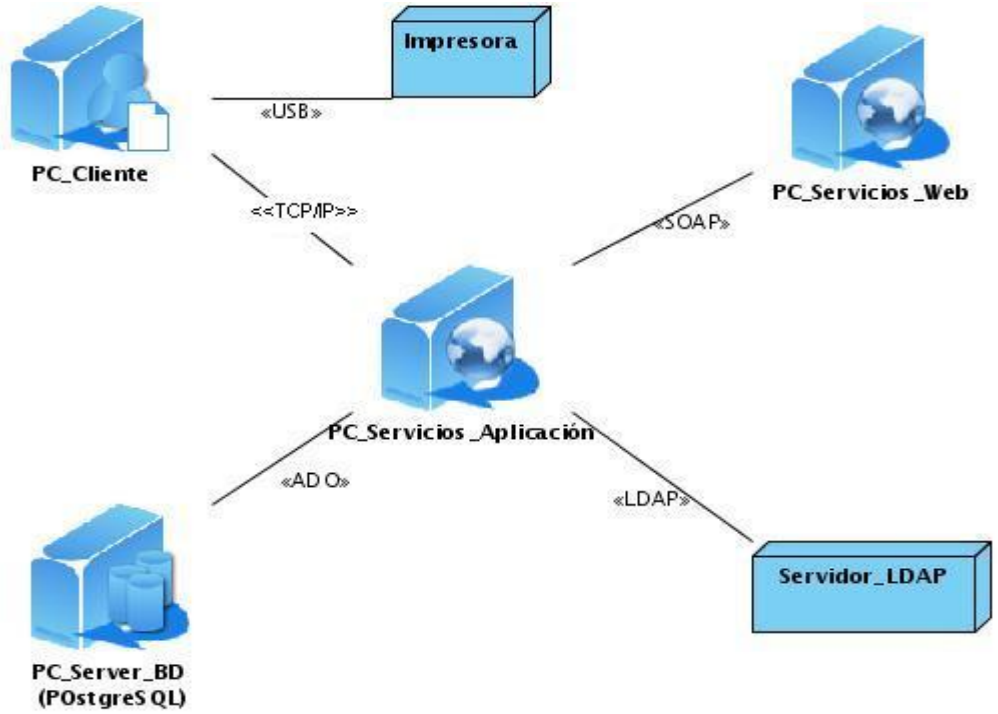


**Figura 20 Descripción de la vista de despliegue de la aplicación.**

| Modelo de despliegue | |
|---|---|
| **Dispositivo** | **Descripción** |
| PC_Cliente | Este ordenador corresponde con el puesto de trabajo de los clientes y del administrador del sistema, estos serán los únicos trabajadores que van a tener contacto directo con la aplicación, desde aquí el Admin_EP o el lider_proyecto pueden mandar a imprimir los EP, en caso que así lo deseen. |
| PC_Servicios_Web | En esta PC se encuentran los servicios web con los que el sistema va a estar intercambiando información constantemente. En este caso se debe conectar con el sistema de gestión de proyectos de la Infraestructura Productiva de la UCI (IP). |
| PC_Servicios_Aplicación | En esta PC se encuentra el servidor de la aplicación, además se publican los servicios web propios de la aplicación ya sean de gestión o públicos. |
| PC_Server_BD | Servidor de la BD con el cliente PostgreSQL de la aplicación. |
| Servidor_LDAP | Este será el nodo al que la aplicación deberá conectarse para gestionar la autenticación de los usuarios. |

## 3.8 Conclusiones.

En este capítulo se definió la arquitectura que utilizará el sistema, se especificaron para ello los estilos y patrones que se deberán implementar y se concretó con el documento de descripción de la arquitectura, exponiendo sus vistas y los diferentes componentes y ficheros que están presentes en ellas.

## CONCLUSIONES

En el presente trabajo de diploma se presenta una propuesta arquitectónica para un Sistema de Gestión Documental para Expedientes de Proyectos, GEDEP, lo cual arroja las siguientes conclusiones:

- El estudio de diferentes sistemas de Gestión Documental a nivel mundial permitió determinar que no existe uno que satisfaga las necesidades de la Dirección de Calidad de Software con respecto a la gestión del expediente de los proyectos productivos de la Universidad de las Ciencias Informáticas.
- Las herramientas, metodología y lenguaje de modelado identificado a partir de la investigación de las tendencias y tecnologías actuales para el desarrollo de software garantizará una modelación estable y organizada de los procesos que formarán parte del proceso de descripción de la arquitectura de GEDEP y cumplen además con el principio de independencia tecnológica por el que se rige el país y la universidad.
- Se realizó un estudio detallado de varios estilos y patrones arquitectónicos existentes en la actualidad para hacer uso de las mejores técnicas de diseño arquitectónico.
- Con la obtención de los requisitos, se lograron identificar y describir los casos de uso arquitectónicamente significativos, validados mediante el prototipo de interfaz gráfica de usuario.
- La descripción de la arquitectura propuesta para el GEDEP facilita el desarrollo paralelo del sistema, el mantenimiento y soporte de la aplicación al tratar cada capa de forma individual, dando así cumplimiento a todos los objetivos que se trazaron para el trabajo de diploma.

**RECOMENDACIONES**

- Aplicar los métodos de evaluación de la arquitectura para identificar las posibles debilidades.
- Mantener un constante refinamiento de la arquitectura a lo largo del ciclo de desarrollo de GEDEP.
- Investigar otros plugins o formas de integración de Symfony con herramientas para generar reportes.
- Desarrollar GEDEP en pos de solucionar los problemas expuestos a lo largo del trabajo de diploma tomando como base la arquitectura propuesta en el trabajo de diploma.

**BIBLIOGRAFÍA**


1. **Arias, Juan Pablo, y otros.** [En línea] http://pisis.unalmed.edu.co/cursos/material/3004582/1/PresentacionFDD.ppt.
2. **Cleveland, Gary. 1995.** Overview of Document Management Technology. *Overview of Document Management Technology.* Ottawa, Canada : s.n., 1995.
3. **Codina, Lluís. 2003.** http://www.elprofesionaldelainformacion.com. *http://www.elprofesionaldelainformacion.com.* [Online] Mayo 2003. [Cited: Enero 8, 2009.] http://www.elprofesionaldelainformacion.com/contenidos/1993/mayo/qu_es_un_sistema_de_gestin_documental.html.
4. **Dante, Gloria Ponjuán. 2005.** http://www.articlearchives.com. *http://www.articlearchives.com.* [Online] Diciembre 1, 2005. [Cited: Enero 8, 2009.] http://www.articlearchives.com/1016376-1.html.
5. **Fowler, Martin. 2003.** *sitio programaciónextrema.* [En línea] abril de 2003. [Citado el: 20 de marzo de 2009.] http://www.programacionextrema.org/articulos/newMethodology.es.html.
6. **González, Encarna. 2007.** http://www.idg.es. *http://www.idg.es.* [Online] Diciembre 21, 2007. [Cited: Enero 9, 2009.] http://www.idg.es/pcworldtech/Gestion-documental-para-una-optima-administracion-/art187432-actualidad.htm.
7. http://www.gitdoc.com. *http://www.gitdoc.com.* [Online] [Cited: Enero 9, 2009.]
8. **Herrera., Antonia Heredia. 2008.** *Gestión documental y calidad.* Sevilla: s.n., 2008. http://www.gitdoc.com/ctl_arch/consultor.htm.
9. **Kaiser, S. H. 2005.** *Software Paradigms.* 2005.
10. **Pimentel, Luis Alberto and Rivero, Iósev Pérez. 2007.** *ArBaWeb: Arquitectura base sobre la web.* Ciudad de La Habana : s.n., 2007.
11. **Patel, P. 1997.** *The Intranet phenomena.* 1, s.l. : Information Management and Technology, 1997, Vol. 30.
12. **Werner, Félix Gómez-Guillamón. 2005.** La Gestión Documental y la norma ISO 15489:2001 Record Management. *Boletin de la asociación andaluza de bibliotecarios.* 2005, Vol. 78.
13. **Zarzuela, Jorge Ferrer. 2003.** *Metodologías Ágiles.* 2003.
14. *sitio Ejemplos de Java y C/linux.* [En línea] http://www.chuidiang.com/ood/metodologia/scrum.php.
15. *sitio EKAENLinea.com.* [En línea] http://www.ekaenlinea.com/index.php?option=com_content&task=view&id=3518&Itemid=66.

16. *sitio Maestros del Web.* [En línea] http://www.maestrosdelweb.com/principiantes/los-diferentes-lenguajes-de-programacion-para-la-web/.
17. *sitio CMS-SPAIN.com.* [En línea] DAA Contenidos Digitales, S.L., 2003. [Citado el: 10 de enero de 2009.] http://www.ecm-spain.com/noticia.asp?IdItem=10012.
18. *sitio Alfresco.* [En línea] [Citado el: 10 de enero de 2009.] http://www.alfresco.com/es/products/solutions/ecm/dm/.
19. *sitio Alfresco.* [En línea] [Citado el: 11 de enero de 2009.] http://www.alfresco.com/es/products/dm/.
20. *sitio DocShare.* [En línea] [Citado el: 15 de enero de 2009.] http://www.docshare.es.
21. *sitio eWWW.* [En línea] http://www.qualitrain.com.mx/index.php/Procesos-de-software/Metodologias-agiles-de-desarrollo-tercera-parte.html.
22. *sitio gestion-conocimiento.* [En línea] [Citado el: 14 de enero de 2009.] http://www.gestion-conocimiento.com/contenido/gestiondoc.asp.
23. *sitio monografias.* [En línea] 712 de julio de 2005. [Citado el: 26 de febrero de 2009.] http://www.monografias.com.
24. *sitio proyectos ágiles.* [En línea] [Citado el: 16 de febrero de 2009.] http://www.proyectosagiles.org/que-es-scrum.
25. *sitio Alfresco.* [En línea] [Citado el: 12 de enero de 2009.] http://www.alfresco.com/es/media/coverage/2008/10/alfrescoiberiaroadshow/intecna.pdf. bibliociencias. [En línea] [Citado el: 12 de enero de 2009.] http://www.bibliociencias.cu/gsdl/collect/eventos/index/assoc/HASH01ba.dir/doc.pdf.
26. *sitio GitDoc.* [En línea] [Citado el: 12 de enero de 2009.] http://www.gitdoc.com/ctl_arch/escanea.htm.
27. **Herrington, Jack. 2006.** Acerca de nosotros: IBM. *sitio IBM.* [En línea] 18 de julio de 2006. [Citado el: 18 de febrero de 2009.] http://www.ibm.com/developerworks/opensource/.
28. *sitio HSM.* [En línea] [Citado el: 12 de enero de 2009.] http://www.hsmsoft.com/soluciones/matrix-manager_2/papiro_6/.
29. **Sánchez, Ana Leonor González.** *Implementación de un archivo de gestión.* 2008.
30. *sitio Infodoc.* [En línea] 2008. [Citado el: 13 de enero de 2009.]

http://www.infodoc.es/servicios/sistemasdeinformacion.asp.

31. **Díaz, Fernando Antonio Ávila.** *La Gestión Documental en Cuba: resultado de la política cultural de la revolución.* Ciudad de La Habana, Cuba : s.n., 2008.
32. **Werner, Félix Gómez-Guillamón.** *La gestión documental y la norma ISO 15489:2001 Record Management.* 2005.
33. *sitio OrfeoGPL.* [En línea] [Citado el: 09 de enero de 2009.] http://orfeogpl.org/ata/node/358.
34. *sitio OrfeoGPL.* [En línea] [Citado el: 09 de enero de 2009.] http://orfeogpl.info/wiki/index.php/OrfeoWiki#Algunas_Caracter.C3.ADsticas.
35. **Potencier, Fabien.** *sitio librosweb.es.* [En línea] http://www.librosweb.es/symfony_1_0/capitulo1.html.
36. **Potencier, Fabien y Zaninotto, Francois. 2008.** *Symfony la guía definitiva.* 2008.
37. **practices, Microsoft patterns &. 2008.** *Application Architecture Guide 2.0.* Ustados Unidos de América : s.n., 2008.
38. *sitio Radio Surco.* [En línea] [Citado el: 10 de enero de 2009.] http://www.radiosurco.cu/Avilenas.asp?newsid=8622&A=T.
39. Revista de gestión documental. *sitio Revista de gestión documental.* [En línea] [Citado el: 12 de enero de 2009.] http://www.revistagestiondocumental.com/2009/05/21/la-biblioteca-virtual-miguel-de-cervantes-digitaliza-cuatro-anos-de-historia/.
40. **Ricardo, Ángel Ciudad. 2007.** Acerca de nosotros: monografias. *sitio monografias.* [En línea] 18 de febrero de 2007. [Citado el: 15 de marzo de 2009.] http://www.monografias.com.
41. **Cruz, Magdelivia.** *Servicio de Organización de Archivos de Gestión. Una respuesta oportuna a las necesidades de las organizaciones tuneras.* Las Tunas, Cuba : s.n., 2004.
42. **Visconti, Marcello y Astudillo, Hernán. 2004.** [En línea] 8 de septiembre de 2004. [Citado el: 20 de febrero de 2009.] http://portal.inf.utfsm.cl/.
43. *sitio YerbabuenaECM.* [En línea] [Citado el: 10 de enero de 2009.] http://www.yerbabuena.es/sections/productos/ys-ecm.

**ANEXOS**

Anexo 1. CUS Visualizar_expediente.

| **Descripción del caso de uso** | |
|---|---|
| **Nombre del caso de uso** | Visualizar_expediente. |
| **Objetivo** | Visualizar el EP de un proyecto en específico con el fin de poder chequear su estructura y realizarle revisiones de calidad. |
| **Actores** | Trabajador_DCS (inicia) |
| **Resumen** | El CUS inicia cuando un trabajador_DCS necesita visualizar un EP en su totalidad para revisar su estructura o realizar alguna consulta al mismo. |
| **Precondiciones** | Al menos un EP debe estar creado. |
| **Pos condiciones** | |
| **Referencias** | R4 |
| **Curso normal de los eventos.** | |
| **Acción del actor** | **Respuesta del sistema** |
| 1. El actor selecciona la opción "Visualizar Expediente". | 2. El sistema muestra el listado de los expedientes existentes. |
| 3. El actor selecciona el EP deseado. | 4. El sistema muestra un mensaje de confirmación para el cliente. |
| 5. El cliente presiona el botón Aceptar. | 6. El sistema muestra el EP seleccionado. |
| **Flujo alternativo "Cancelar visualizar expediente"** | |
| 1. El usuario presiona el botón Cancelar. | 2. El sistema cancela la operación y regresa al paso 1 del curso normal de los eventos. |
| **Prioridad** | Secundario. |
| **Prototipo** | |

Anexo 2. CUS Realizar_auditorias.

| **Descripción del caso de uso** |
|---|

| | |
|---|---|
| **Nombre del caso de uso** | Realizar_auditorías |
| **Objetivo** | El objetivo del CUS es que un integrante del equipo de auditoría y RTF pueda acceder al EP de un proyecto determinado para realizar los controles de una forma más factible y organizada. |
| **Actores** | Revisor (inicia). |
| **Resumen** | El revisor accede al espacio de un proyecto determinado y escoge ver su EP para realizar las auditorías. |
| **Precondiciones** | Debe existir el proyecto y su EP. |
| **Pos condiciones** | Se guarda en el sistema el resultado final de la auditoría o RTF. |
| **Referencias** | R3 |

| **Curso normal de los eventos.** | |
|---|---|
| **Acción del actor** | **Respuesta del sistema** |
| 1. El actor selecciona la opción “Control de versiones”. | 2. El sistema muestra el listado de documentos a seleccionar por parte del revisor. |
| 3. El actor selecciona el documento desea y presiona el botón Historial de versiones. | 4. El sistema muestra los datos de ese documento, incluyendo además el listado de las versiones por las que ha transcurrido, dándole al usuario la opción de descargar la versión deseada. |
| 5. El usuario presiona el botón Descargar. | 6. El sistema guarda la versión en la dirección indicada para ser sometida a las revisiones. |
| 7. El usuario una vez que tiene los resultados de la auditoría selecciona la opción “Auditoría y RTF”. | 8. El sistema muestra un formulario con los campos a llenar como parte del resultado. |
| 9. El usuario completa los campos respectivos y presiona el botón | 10. El resultado queda registrado en el sistema. |

| | |
|---|---|
| Aceptar. | |
| **Flujo alternativo "Cancelar revisar expediente"** | |
| 9.a) El usuario presiona el botón Cancelar. | 9.b) El sistema cancela la operación y regresa al paso 7. |
| **Prioridad** | Secundario. |
| **Prototipo** | |

Anexo 3. CUS Realizar_rev_tec_formales.

| **Descripción del caso de uso** | |
|---|---|
| **Nombre del caso de uso** | Realizar_rev_tec_formales |
| **Objetivo** | El objetivo del CUS es que un integrante del equipo de auditoría y revisiones pueda acceder al EP de un proyecto determinado para realizar las mismas de una forma más factible y organizada. |
| **Actores** | Revisor (inicia). |
| **Resumen** | El CUS inicia cuando el revisor accede al espacio de un proyecto determinado y escoge ver su EP para realizar las RTF. |
| **Precondiciones** | Debe existir el proyecto y su EP. |
| **Pos condiciones** | |
| **Referencias** | R3 |

| **Curso normal de los eventos.** | |
|---|---|
| **Acción del actor** | **Respuesta del sistema** |
| 1. El actor selecciona la opción "Control de versiones". | 2. El sistema muestra el listado de documentos a seleccionar por parte del revisor. |
| 3. El actor selecciona el documento desea y presiona el botón Historial de versiones. | 4. El sistema muestra los datos de ese documento, incluyendo además el listado de las versiones por las que ha transcurrido, dándole al usuario la |

| | |
|---|---|
| | opción de descargar la versión deseada. |
| 5. El usuario presiona el botón Descargar. | 6. El sistema guarda la versión en la dirección indicada para ser sometida a las revisiones. |
| 7. El usuario una vez que tiene los resultados de la RTF selecciona la opción “Auditoría y RTF”. | 8. El sistema muestra un formulario con los campos a llenar como parte del resultado. |
| 9. El usuario completa los campos respectivos y presiona el botón Aceptar. | 10. El resultado queda registrado en el sistema. |
| **Flujo alternativo “Cancelar revisar expediente”** | |
| 9.a) El usuario presiona el botón Cancelar. | 9.b) El sistema cancela la operación y regresa al paso 7. |
| **Prioridad** | Secundario. |
| **Prototipo** | |

Anexo 4. CUS Exportar_expediente.

| **Descripción del caso de uso** | |
|---|---|
| **Nombre del caso de uso** | Exportar_expediente |
| **Objetivo** | El objetivo del CUS es poder exportar el EP completo o una de sus carpetas con todo su contenido, plantillas o artefactos, hacia un documento en formato PDF. |
| **Actores** | Admin_EP (inicia). |
| **Resumen** | El CUS inicia cuando el Admin_EP selecciona la opción “Exportar expediente”. |
| **Precondiciones** | Debe existir al menos un EP. |

| | |
|---|---|
| **Pos condiciones** | El archivo se exporta hacia una dirección determinada por parte del usuario. |
| **Referencias** | R6. |
| **Curso normal de los eventos.** | |
| **Acción del actor** | **Respuesta del sistema** |
| 1. El actor selecciona los artefactos que desea exportar. | 2. El sistema pide indicar la dirección dónde se encuentra el artefacto seleccionado, ya sea el EP completo, una carpeta, plantilla o artefacto del mismo. |
| 3. El usuario señala la ubicación y presiona el botón Aceptar. | 4. El sistema pide indicar la dirección donde se guardará el resultado de la acción, o sea el documento ya exportado a PDF. |
| 5. El usuario señala la ubicación y presiona el botón Exportar. | 6. El sistema exporta el artefacto a formato PDF. |
| **Flujo alternativo "Cancelar exportar expediente"** | |
| 3.a) El actor presiona el botón Cancelar. | 3.b) El sistema cancela la acción y regresa al paso 1. |
| **Prioridad** | Secundario. |
| **Prototipo** | |

Anexo 5. CUS Controlar_versiones_EP.

| **Descripción del caso de uso** | |
|---|---|
| **Nombre del caso de uso** | Controlar_versiones_EP. |
| **Objetivo** | El objetivo es controlar las diferentes versiones del EP cada vez que se le realiza un cambio a un artefacto del mismo. |
| **Actores** | Trabajador_DCS (inicia). |

| | |
|---|---|
| **Resumen** | El CUS inicia cuando el actor del sistema una vez seleccionado el EP desea consultar las versiones por las que este ha pasado. |
| **Precondiciones** | El actor debe haber seleccionado un EP determinado. |
| **Pos condiciones** | |
| **Referencias** | R5 |

| **Curso normal de los eventos.** | |
|---|---|
| **Acción del actor** | **Respuesta del sistema** |
| 1. El actor selecciona la opción “Control de versiones”. | 2. El sistema muestra el listado de documentos a seleccionar por parte del revisor. |
| 3. El actor selecciona el documento desea y presiona el botón Historial de versiones. | 4. El sistema muestra los datos de ese documento, incluyendo además el listado de las versiones por las que ha transcurrido, dándole al usuario la opción de descargar la versión deseada. |
| 5. El usuario presiona el botón Descargar. | 6. El sistema guarda la versión en la dirección indicada. |
| **Flujo alternativo** | |
| | |
| **Prioridad** | Secundario. |
| **Prototipo** | |

Anexo 6. CUS Buscar_expediente.

| **Descripción del caso de uso** | |
|---|---|
| **Nombre del caso de uso** | Buscar_expediente |
| **Objetivo** | Buscar un EP según los criterios de búsqueda definidos. |
| **Actores** | Trabajador_DCS (inicia). |

| | |
|---|---|
| **Resumen** | El CUS inicia cuando un Trabajador_DSC necesita buscar un EP. |
| **Precondiciones** | Al menos un EP debe estar creado. |
| **Pos condiciones** | El resultado de la búsqueda se mostrará en un listado dando la posibilidad de seleccionar uno de los expedientes hallados y visualizarlo posteriormente. |
| **Referencias** | R7 |

| **Curso normal de los eventos.** | |
|---|---|
| **Acción del actor** | **Respuesta del sistema** |
| 1. El actor selecciona la opción “Buscar expediente”. | 2. El sistema muestra los diferentes criterios de búsqueda. |
| 3. El actor completa la información de los criterios de búsqueda y presiona el botón Buscar. | 4. El sistema muestra un listado con el resultado de la búsqueda y activa el botón Visualizar expediente. |
| **Prioridad** | Secundario. |
| **Prototipo** | |

**GLOSARIO DE TÉRMINOS**

**Actor:** Alguien o algo, fuera del sistema o negocio que interactúa con el sistema o negocio.

**Arquitectura:** Vista del sistema software 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.

**Artefactos:** Una parte de la información que es producida, modificada, o usada por un proceso, define un área de responsabilidad, y está sujeta al control de versión. Un artefacto puede ser un modelo, un elemento del modelo, o un documento. Un documento puede adjuntar otros documentos.

**ASP (Active Server Pages):** es una tecnología del lado servidor de Microsoft para páginas web generadas dinámicamente, que ha sido comercializada como un anexo a Internet Information Server (IIS).

**Caso de uso:** Descripción del comportamiento del sistema en término de secuencia de acciones.

**Cliente:** es un ordenador que accede a recursos y servicios brindados por otro llamado Servidor, generalmente en forma remota. Una persona u organización, interna o externa a la organización productora que toma responsabilidad financiera por el sistema. El cliente es el último destinatario del producto desarrollado y sus artefactos.

**Cliente-Servidor:** esta arquitectura consiste básicamente en que un programa (cliente informático) realiza peticiones a otro programa (servidor) que les da respuesta.

**Data Access Objects (DAO):** es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo.

**GEDEP:** Sistema de Gestión Documental basado en el Expediente de Proyecto.

**GoF (Gang of Four):** Grupo de los Cuatro, se refiere al grupo de los autores de este tipo de patrones de diseño.

**Hardware (soporte físico):** se utiliza generalmente para describir los artefactos físicos de una tecnología. Es el conjunto de elementos físicos que componen una computadora Disco Duro, CD-ROM, etc.

**IDE (Integrated Development Environment):** un entorno de desarrollo integrado, es un programa compuesto por un conjunto de herramientas para un programador. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios.

**Layout:** En Symfony se conoce como Layout el enmarcado que se le da a las páginas.

**Lenguaje de Modelado Unificado (UML):** es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software.

**Metodología:** En un proyecto de desarrollo de software la metodología define Quién debe hacer Qué, Cuándo y Cómo debe hacerlo.

**Servidor:** una aplicación informática o programa que realiza algunas tareas en beneficio de otras aplicaciones llamadas clientes.

**Servicios:** es un conjunto de actividades que buscan responder a una o más necesidades de un cliente.

**Servicios Web:** Un servicio web (en inglés web service) es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones, a través de la web. Los servicios web pueden ser utilizados por distintas aplicaciones de software, desarrollados en diferentes lenguajes de programación y ejecutados sobre cualquier plataforma, para intercambiar datos en redes de ordenadores como Internet.

**Sistema Operativo:** es un conjunto de programas destinados a permitir la comunicación del usuario con un computador y gestionar sus recursos de una forma eficaz.

**Software (soporte lógico):** los componentes intangibles de una computadora, es decir, al conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica.

**XML (sigla en inglés de eXtensible Markup Language):** lenguaje de marcado extensible es un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

1