Universidad de las Ciencias Informáticas

Título:

“Diseño de una Arquitectura de Software para el Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado.”

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor:

Noraysi Prat Montesino

Tutor:

Ing. Armando Ortíz Cabrera.

Pensamiento

Figura # 1. Danny Thorpe

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

Danny Thorpe.

Dedicatoria

Dedico esta tesis a mi mamá por no dejarme sola nunca y darme fuerzas y luchar junto conmigo. Por exigirme tanto con sus peleas.

Amita de veras te mereces esto y mucho más por ser mi luz y ejemplo. Te quiero.

A mi papá que es la persona que más respeto y admiro. Hoy quería que supieras que te quiero con todo mi corazón y que estoy muy orgullosa de tenerte como mi papi.

A ti apito te dedico esta tesis.

A mi hermanita del alama. A ella que es mi niña querida y mimada. Es mi amiga, compañera, confidente y hermana. La quiero más que a nada en este mundo.

A ti hembrito te dedico esta tesis.

A mi abuela que me ah querido, cuidado, guiado, aconsejado y amado como mi propia madre. Por eso te quiero tanto y te respecto además.

A ti mima te dedico esta tesis con todo mi amor.

Agradecimientos

A mis padres Noraida Montesino Cruz y Servilio Prat Madera por todo su apoyo, comprensión y por la confianza depositada en mí. A ustedes, gracias por hacer de este sueño una realidad.

A mi hermanita Norielyz Prat Montesino que aún estando lejos se mantuvo siempre junto a mí. Tú eres mi mayor inspiración. Gracias.

A toda mi familia que me apoyó tanto. A mis tíos y tías, primos y primas. A los que siguieron más de cerca mis pasos y a los que no también. Al Cani y a Juanito. A mis abuelos, en especial a mi abuela Zoila Montesino Cruz por ser abuela y madre a la vez.

A ti Carlos Jonet Rodríguez San Miguel por ser mi amigo y compañero. Por creer siempre en mí, aún cuando yo misma no creí. Gracias.

A todos mis compañeros de la primaria y secundaria. A Danay, Yuselys, Yariel y Alfre por mantenerse cerca aún cuando el destino nos ha llevado por caminos diferentes.

A mis compañeros de la Vocacional. En especial a Yaneysi Humaran Barreira, Raimleys Díaz Sendín y Zenen Hernández Martínez, por apoyarme tanto en su momento y por ser más que mis amigos mis hermanitos. También a Dayamí Chávez Ayala y a Daylin García Giniebra que siguieron el mismo camino que yo y me han acompañado y aconsejado durante toda la carrera.

A mis verdaderos amigos de la Universidad. Los que quedan. A Ariagna Rodríguez Donatien por tener la paciencia de soportarme durante estos 5 largos años. A Yadier Castro Piedra por compartir 3 maravillosos años conmigo y ser el pinareño más lindo de todos. A Yaneisy Castilla Ramirez por estar siempre ahí cuando la necesité por más de 2 años. A Maylen Ricardo Évora por dejar esos recuerdos lindos de aquel primer año de carrera y por mantenerse siempre cerca. A todos ustedes muchas gracias por ser los mejores amigos del mundo. Me va a costar mucho aceptar que posiblemente a algunos no los vea más.

A Michel Álvarez Posada, un amigo del barrio, por darme ánimo y aliento. Por preguntar siempre ¿Y cómo va la tesis?

A ti Irisbel Fuentes Benítez por todo el apoyo que me has brindado.

A mis antiguos compañeros de aula del grupo 9103. En especial a Yoandri Quintana Rondón, la copa y a Adonis Gonzales Cárdenas, el pocho. A los nuevos compañeros del grupo 9502. En especial a Yunier Alexander Pimienta por ser más que mi compañero mi tutor personal, mi consultante. Por estar siempre ahí dispuesto a ayudarme. Gracias Pimi.

A mis compañeras de apartamento durante todos estos años. Las que convivimos en el 86, luego en el 105 y ahora en el 148. Gracias por aceptarme como soy.

A mi tutor Armando Ortiz Cabrera. Gracias por estar ahí y ayudarme a tomar las decisiones correctas. Sin ti, el desarrollo de este trabajo no hubiese sido posible.

Al tribunal. En especial a Alexey Díaz Domínguez por ser más que mi Jefe de Tribunal, ser profe y a la vez amigo. Por exigirme tanto y tanto. La calidad del resultado de esta tesis se la debo a él. Gracias.

A todos los profes que me ayudaron a llegar hasta aquí. A mi maestro de primaria Albe y a mi profes de secundaria Edilia y Papito. A todos los profes que durante la carrera Universitaria creyeron en mí y a los que no también. En especial a Vladimir Martell mi profe para la preparación de la PNP. A tí Vladimir muchas, muchas gracias por toda tu dedicación y empeño. Sabes que no hay palabras para describir el aprecio tan grande que te tengo.

Y a todos aquellos que sin ser padres, amigos o profes, estuvieron ahí en un momento dado, apoyándome o tal vez criticándome, a ustedes gracias.

Resumen

El presente trabajo contempla el diseño de una Arquitectura de Software para los Procesos de Facturación y Cobro de la Empresa de Gas Manufacturado, ubicada en Ciudad de La Habana.

En el primer capítulo se aborda toda la teoría relacionada con los procesos de facturación y cobro en la empresa. Se analizan otras soluciones existentes y se estudian las principales herramientas y tecnologías a utilizar en el desarrollo del sistema. En el segundo capítulo se seleccionan las tecnologías y herramientas que fueron necesarias utilizar para desarrollar el nuevo sistema automatizado. El tercer capítulo describe la línea base de la arquitectura, siendo aquí donde se propone la arquitectura con que se modelará la aplicación.

Como resultado de la investigación se obtiene una sólida arquitectura para el Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado (producto: Manugas), que permite facturar a sus clientes por una tarifa escalonada dinámica, facilitándole el pago a los clientes en diferentes Casas Comerciales, así como la consulta y reporte de información del estado de facturación y cobro actualizada en cualquier momento.

Índice

Índice de Figuras

Índice de Tablas

Introducción

Actualmente el país se encuentra inmerso en un proceso de informatización, con el propósito de extender el desarrollo informático hacia todos los sectores de la sociedad. No cabe duda de que un elevado número de las empresas cubanas van camino a la adaptación de los modelos de negocio basados en las tecnologías de la información y las comunicaciones (TIC). Sin embargo, el problema actual radica en que no todas disponen de la capacidad suficiente para adoptar aquellos modelos de negocio que las hagan más abiertas y competitivas. La base tecnológica de la gran mayoría de las empresas de nuestro país enfrenta una etapa de revolución y por otra parte, es necesario un cambio de mentalidad hacia posiciones más flexibles respecto al empleo de las TIC.

El desarrollo de las TIC está provocando una migración en el funcionamiento interno de las organizaciones hacia sistemas electrónicos y digitales. Esto se logra mediante la instalación de redes informáticas y la implementación de aplicaciones y sistemas que mejoran la rapidez de las operaciones inherentes que a su vez están protegidas por mecanismos de mayor seguridad que los tradicionales. La aplicación de sistemas electrónicos modernos de esta naturaleza (aplicaciones básicas y herramientas avanzadas de gestión de los recursos de la empresa así como de la relación con los clientes, etc.), posibilita un control permanente y mucho más efectivo de todas las secciones de la organización así como una evaluación más integral de todo el sistema.

En la actualidad la empresa Cuba Petróleo (CUPET) desarrolla proyectos en el sector de la energía con la Universidad de las Ciencias Informáticas (UCI). Dicha entidad brinda un conjunto de servicios a lo largo de todo el país. Uno de estos servicios lo rectorea la Empresa de Gas Manufacturado ubicada en Ciudad de La Habana, que se dedica a la distribución, control y cobro del servicio de gas manufacturado a domicilio.

La empresa tiene en ejecución un software que le informatiza los procesos, el mismo tiene una serie de limitaciones que provocan que el control de dichos procesos se realice más lento y que existan problemas de eficiencia en los mismos.

Entre las principales limitaciones del sistema (SISMET) se encuentran las que a continuación se mencionan.

No permite:

1. Que las tarifas sean configurables y dinámicas.
2. Facturar más de un mes a cada cliente, es decir, que la facturación no se puede realizar ni bimestral ni trimestral.
3. Efectuar depósito o pagos adelantados para el cobro del gas.
4. Efectuar el cobro de gas a través de correo y banco.
5. Facturar por otro precio que no es el actual.
6. Incluir dentro del proceso de facturación el cobro de servicios post venta al cliente. Por ej. Variaciones en la instalación del servicio de gas.
7. Consultar información actualizada desde las Casas Comerciales.
8. Efectuar pago desde cualquier Casa Comercial.
9. Que la conexión entre las Casas Comerciales por redes sea mayor a los 56 Kbps En muchas de las ocasiones la conexión llega a ser de 24 Kbps

Por tales razones, la Empresa de Gas Manufacturado (conocida por UEB Comercial), solicitó al polo de PetroSoft de la Facultad 9 de la UCI, el desarrollo de un sistema que controle y gestione la información referente a los procesos de Facturación y Cobro del gas suministrado a los núcleos familiares de Ciudad de La Habana.

Las deficiencias señaladas en párrafos anteriores conllevan a la empresa a presentar diferentes fallas en sus procesos fundamentales, de ahí que se identifica desde la dirección de la entidad la necesidad de desarrollar un software que informatice los procesos de manera eficiente y con la calidad requerida. Después de haber realizado un estudio del software que está en ejecución en la empresa se detectó que el mismo necesita interactuar con dispositivos externos como el Terminal Portátil de Lectura (TPL), entidades de tercera y que no presentaba una arquitectura adecuada y robusta que permitiera a los desarrolladores poder realizarle nuevas actualizaciones.

Debido a lo anteriormente planteado se identificó el siguiente **problema científico**: Inexistencia de una adecuada Arquitectura de Software con el uso de herramientas libres, que permita el desarrollo de un producto de software para el Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado.

Es por ello que el presente trabajo tiene como **objeto de estudio**:

- Los Procesos de Facturación y Cobro de la Empresa de Gas Manufacturado.

El **campo de acción** donde se enmarca: La Arquitectura de Software para el Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado.

Como **idea a defender** se tiene: Si se diseña una Arquitectura de Software, que cumpla con los requerimientos y atributos de calidad establecidos para el Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado, apoyada en el uso de herramientas libres, se sentarán las bases necesarias para comprender, desarrollar y producir económicamente la aplicación, logrando una alta satisfacción del cliente.

Para ello se ha definido el siguiente **objetivo general**: Diseñar una Arquitectura de Software para el Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado, que permita el desarrollo de un producto flexible, robusto y escalable, cumpliendo así con los requerimientos no funcionales, apoyándose en el uso de herramientas libres.

Este objetivo puede ser logrado a partir de las siguientes **tareas de la investigación**:

1. Describir los procesos de negocio actuales de la Empresa de Gas Manufacturado a informatizar.
2. Describir las actividades que se identifican en el rol del Arquitecto de software.
3. Definir y caracterizar los estilos arquitectónicos existentes.
4. Definir y caracterizar los patrones de diseño.
5. Caracterizar las herramientas y tecnologías a utilizar en el desarrollo del producto.
6. Determinar las funcionalidades con que debe contar el sistema.
7. Diseñar una propuesta arquitectónica que cumpla con los requerimientos de la aplicación y que garantice el desarrollo paralelo y la reutilización.

Para dar cumplimiento a estas tareas de investigación, se hizo necesario el uso de diferentes métodos de investigación.

Como **métodos teóricos** se emplea:

- Analítico-sintético: Permite a través de la investigación de las teorías y documentos existentes sobre las Arquitecturas de Software usadas en el Proceso de Desarrollo de Software hacer un análisis más profundo y llegar así a su esencia, determinar los rasgos que lo caracterizan y extraer los elementos más importantes que se relacionan con dichas arquitecturas.

- Inductivo-deductivo: Se pone de manifiesto cuando generalizamos el conocimiento adquirido, o sea, que partiendo del estudio de lo particular se llega a una conclusión general.

- Histórico-lógico: Para determinar las tendencias actuales. En este caso se empleará para realizar el estudio de la trayectoria real de determinados elementos que servirán de guía para la construcción de una buena Arquitectura de Software.

- Modelación: Se hace muy importante el uso de éste método para el diseño de la Arquitectura de Software pues la modelación es justamente el proceso mediante el cual creamos modelos, dígase modelos a propuestas, alternativas, estrategias, etc. que son usadas con vistas a investigar la realidad.

Como **métodos empíricos** se usa:

- Observación: Permite conocer los detalles fundamentales sobre el desarrollo de la arquitectura para el Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado.

- Entrevista: Permite conocer exactamente el funcionamiento del negocio y detallar cuáles eran los requisitos necesarios para llevar a cabo los procesos de negocio identificados.

Aportes prácticos esperados del trabajo.

Se espera, por tanto, al finalizar la investigación, la obtención de una sólida arquitectura para el Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado que permita facturar a sus clientes por una tarifa escalonada dinámica, facilitándoles el pago a los mismos en diferentes Casas Comerciales, así como las consultas y reportes de información del estado de la facturación y el cobro actualizada en cualquier momento.

El presente trabajo consta de 3 capítulos estructurados de la siguiente forma:

Capítulo1.Estudio del Arte y Fundamentación Teórica.

En esta sección se describen los principales procesos de negocio que se identifican en la empresa y así mismo se definen a modo general, lo que son las herramientas y

tecnologías a usar para el desarrollo del sistema, de forma tal que en capítulos posteriores se pueda decidir cuál es la más adecuada para el desarrollo del proyecto.

Capítulo2: Tecnologías a utilizar.

Se dispone de un análisis fuerte sobre las principales tecnologías y herramientas, definiendo finalmente cuáles son los apropiados para el desarrollo del sistema.

Capítulo3: Línea base de la Arquitectura.

Basado en los estudios realizados en capítulos anteriores y las necesidades del negocio que exige la empresa, se propone la arquitectura que dará soporte al sistema implementado. Se define la línea base de la arquitectura propuesta.

Capítulo 1: Estado del Arte y Fundamentación Teórica.

1.1 Introducción.

En este capítulo se abordan las temáticas referentes al estado del arte de la Arquitectura de Software, partiendo del estudio de los procesos de Facturación y Cobro que realizan otros sistemas ya existentes. Se tienen en cuenta para ello los principales conceptos de arquitectura, los diferentes estilos y patrones arquitectónicos. De la misma forma, se introducen definiciones y características sobre las principales herramientas y tecnologías a usar en el desarrollo del sistema.

1.2 Empresa de Gas Manufacturado.

A continuación se detallan los principales servicios que brinda la Empresa de Gas Manufacturado y se describen los principales procesos que dicha empresa realiza.

1.2.1 Procesos de Negocio y Servicios que brinda.

La Empresa de Gas Manufacturado, es la encargada de gestionar todos los servicios de gas manufacturado a la población de Ciudad de La Habana que tienen contratos con la misma. De esta forma, la empresa le garantiza a las familias cubanas la instalación del sistema de gas a particulares, reparación de sistemas de gas en caso de averías, cobro no solo por la reparación sino también por el servicio que presta dicha entidad y facturación del mismo, siendo este último, al igual que el proceso de cobro, los procesos principales que actualmente se llevan a cabo en dicha empresa.

Es muy importante tener en cuenta, a la hora de identificar un proceso de negocio, que estos debe generar un valor para el negocio o mitigar los costos del mismo y que cada caso de uso del negocio, representa a un proceso de negocio.

Según lo que se observó en las visitas efectuadas a la Empresa de Gas Manufacturado y las entrevistas realizadas a los trabajadores y desarrolladores del Sistema Metrado (SISMET) se identificó, que muchos de los procesos que allí se realizan presentan algunas que otras ineficiencias que no solo afectan a la empresa sino, también a los clientes.

Dentro de los principales procesos de la empresa se tienen:

- **Proceso de Facturación o Ventas de Gas Manufacturado**: se origina la acción cuando luego de haber realizado los contratos a los clientes, se le otorga su número de cliente y son procesados en el Sistema Automatizado de Dirección (SAD) para introducir sus datos y su venta se registra para el cliente no metrado de forma fija todos los meses y por la lectura mensual al cliente metrado.

- **Proceso de Cobro del servicio de Gas Manufacturado**: visitar a los clientes en sus viviendas o entidades estatales para cobrar las facturaciones realizadas teniendo en cuenta las tarifas establecidas y lecturas de los metros contadores para ingresar a la cuenta de la empresa la venta recaudada.

1.2.2 Arquitectura de la Empresa.

La empresa cuenta con una Casa Matriz Comercial (UEB Comercial), que es la casa rectora donde se efectúan los procesos de Facturación y Cobro a través de SISMET, sistema informático desarrollado para llevar a cabo estos procesos y que actualmente está presentando fallas. Conectada a ella, están otras Casas Comerciales, que son las casas rectoras a nivel de municipio, encargada de llevar el control y cobro de las facturaciones de las lecturas de los metros contadores de cada núcleo familiar. (Ver Anexo #1)

Al finalizar cada mes, las Casas Comerciales rectoras en cada municipio, le envían los datos de las facturas realizadas a la UEB Comercial, para que esta entonces actualice y registre dicha información. La información se trasmite a través del correo, proceso extremadamente lento porque las conexiones entre las casas municipales y la rectora son a través de un modem que en ocasiones ha alcanzado hasta 25kbyte de velocidad. Este envío de información, muchas otras veces, llega en formato duro, dígase a esto tanto disco duro, como memoria, o fax, en ocasiones. También se pueden dar los partes vía teléfono.

Para el trabajo con dicha información se usa un dispositivo o aparato que lleva un lector cobrador, donde recoge y almacena las lecturas de cada metro contador de gas de cada núcleo familiar. A este dispositivo le llaman Terminal Portátil de Lectura (TPL).

1.2.3 Análisis de Soluciones Existentes.

Después de haber estudiado los principales procesos que se realizan en la empresa y analizado la arquitectura de la misma, es necesario entonces hacer un estudio de otros sistemas existentes que implementen dichos procesos, con el objetivo de ver que características similares presentan que puedan ser utilizadas para el desarrollo de la aplicación.

La investigación, arrojó como resultado, tres sistemas a nivel mundial. Estos sistemas se describen a continuación.

1.2.3.1 FYC4

FYC4 es un Sistema de Facturación y Cobros para la facturación automatizada y digital de las tasas de sobrevuelos, que utiliza actualmente la Autoridad Aeronáutica Civil (AAC) de Panamá, en aras de continuar con su proceso de modernización institucional. Dicho sistema puede recibir información directamente de un sistema de tratamiento de datos de vuelos de forma automatizada, facilitando al usuario el ingreso de los mismos y transformando una labor de varias horas a unos cuantos segundos, eliminando la tasa de errores en su transcripción.

Principales Ventajas del FYC4. (52)

- Aumento de la productividad del personal que gestiona el cobro de los clientes.
- Aumento de la productividad del personal que gestiona la facturación de los clientes.
- Disminución del tiempo de acceso a la información de facturación.
- Disminución del tiempo de acceso a la documentación de los clientes utilizando los mecanismos de digitalización automatizados.
- Disminución del margen de error por la recepción automática de la información a facturar en línea desde el centro de control.
- Disminución del margen del error de los registros de vuelos gracias a los procedimientos nativos de detección de errores del aplicativo.
- Acceso a consultas estadísticas para la planificación y gestión de tráfico aéreo y de cobro.

1.2.3.2 CommTrack.

El sistema de facturación CommTrack fue diseñado específicamente para la industria de las telecomunicaciones en 1997. Está idealmente adecuado para proveer pequeñas y medianas, nuevas o ya existentes compañías de telecomunicaciones, con un sistema completo para atender a sus clientes, recibir pagos, procesar récords de datos recibidos de múltiples portadores (por ej: celulares, servicios locales, larga distancia, localizadores, servicios de internet, etc.), producir una factura consolidada e imprimir esa factura en formato para papel o electrónico. (39)

Dicho sistema cuenta con cuatro módulos: atención al cliente, facturación, contabilidad y administración del sistema. El módulo de facturación que implementa CommTrack tiene características muy similares al que actualmente se está desarrollando en el proyecto. CommTrack utiliza el módulo de facturación para compilar, evaluar y producir facturas unificadas de una amplia variedad de tipos de servicios y formatos de datos de la misma forma que se prevé lo haga el sistema que se está implementando. Por otra parte, CommTrack al igual que Manugas, produce reportes y facturas para papel o electrónicos, además de que puede generar reportes administrativos para realizar análisis de costos y ver los impuestos sobre las ventas.

1.2.3.3 E-xtendgfp.

E-xtendgfp es una plataforma para la gestión de facturas y pagos, cuyo objetivo es conseguir que las empresas usuarias obtengan ahorros de costes sustanciales, gracias a la mejora de la estructura de costes necesarios para llevar a cabo sus procesos de facturación, cobros y pagos. (53)

¿Cuáles son las ventajas de integrar en la empresa este nuevo tipo de facturación?

Las ventajas de usar un sistema de facturación electrónica, como E-xtendgfp, viene dada en el ahorro de la re-introducción de datos en diversos sistemas desde el papel, en la eliminación de los consiguientes errores, en la posibilidad de automatizar la integración con sus sistemas contables y de estandarizar y mecanizar determinados tratamientos sobre las facturas, condiciones de pagos y comunicaciones con sus proveedores y clientes relativas a estos procesos. (53)

Los sistemas analizados anteriormente: FYC4, CommTrack y E-xtendgfp son muy utilizados en el mundo. De los mismos solo se tuvieron en cuenta sus principales características para la realización del sistema, es decir se agrupaba lo mejor de cada uno para obtener un mejor resultado. El principal inconveniente de estos sistemas es que todos están realizados con herramientas propietarias, razón por la cual no se trabajó sobre la base de uno de ellos sino que se decidió desarrollar una nueva aplicación que se adecuara a las características propias del negocio de la Empresa de Gas Manufacturado.

1.3 Arquitectura de Software.

Según RUP: La arquitectura de un sistema es la visión común en la que todos los empleados (desarrolladores y otros usuarios) deben estar de acuerdo, o como poco, deben aceptar. La arquitectura nos da una clara perspectiva del sistema completo, necesaria para controlar el desarrollo. (5)

Una definición oficial dada por la IEEE Std 1471-2000: La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución. (6)

En sí, la Arquitectura de Software da una visión de lo que el sistema debe hacer y se apoya para ello en el arquitecto que es quien define las principales herramientas y tecnologías, dígase patrones de diseño, estilos arquitectónicos, lenguajes de implementación, framework, etc. Por consiguiente la Arquitectura de Software sienta las bases para el desarrollo y la construcción de un sistema de software de manera económico y fiable con la realización de la mayor cantidad de requerimientos no funcionales, obteniendo una alta satisfacción del cliente.

A todo ello se puede agregar que la Arquitectura de Software es la encargada de la descripción y estudio de las propiedades estructurales de los sistemas de software. Este estudio puede llevarse a cabo desde diferentes niveles de abstracción. Yendo de lo más general a lo más específico, se pueden distinguir varios niveles: (8)

- **Primer nivel**: estilos arquitectónicos.
- **Segundo nivel**: modelos y arquitecturas de referencia.
- **Tercer nivel**: marcos de trabajo.
- **Cuarto nivel**: familias y líneas de productos.
- **Quinto nivel**: instancias arquitectónicas.

Es por todo lo antes mencionado que el diseño de una buena arquitectura es muy importante para que el software salga con la calidad requerida, obteniendo como resultado el éxito del proyecto. Además de eso, la arquitectura permite una muy buena comunicación entre las personas involucradas en el desarrollo del sistema y una temprana documentación de las decisiones de diseño tomadas.

1.4 Patrones.

El término patrón se utiliza inicialmente en el campo de la arquitectura, por Christopher Alexander, a finales de los 70s con la publicación de su libro The Timeless Way of Building, en el cual proponía el aprendizaje y uso de patrones para la construcción de edificios.

Según este autor: Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo siquiera dos veces de la misma forma. (14)

El uso de patrones da la posibilidad de identificar y completar los casos de uso básicos expuestos por el cliente; comprender la arquitectura del sistema a construir, así como su problemática; buscar componentes ya desarrollados que cumplan con los requisitos del tipo de sistema a construir. (14)

Es decir, el uso de los patrones permite obtener de una forma sencilla la arquitectura base que se busca durante la fase de diseño arquitectónico.

1.4.1 Formato de un Patrón.

- **Nombre del patrón**: <Nombre>

- **Problema que resuelve**: < ¿Cuál es el principio fundamental en virtud del cual se asignan las responsabilidades a los objetos?>
- **Solución**: <Asignar una responsabilidad a la clase que tiene la información necesaria para cumplirla>

1.4.2 Clasificación de los Patrones de software.

Todo el mundo acepta que existen diversas clases de patrones: de casos de uso, de análisis, de arquitectura, de diseño, de organización o proceso, de programación y los llamados idiom, entre otros.

En el libro Pattern-Oriented Software Architecture se definen tres tipos de patrones: (15)

- **Patrones arquitectónicos** sobre aspectos fundamentales de la estructura de un sistema software. Especifican un conjunto predefinido de módulos con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes.
- **Patrones de diseño** sobre aspectos relacionados con el diseño de los módulos. Por tanto se centran en aspectos más específicos.
- **Un Idiom** que es un patrón de bajo nivel específico de un lenguaje de programación o entorno de desarrollo. Describe como implementar aspectos particulares de los componentes de un patrón de diseño usando las características y potencialidades de un lenguaje de programación concreto. (54)

1.5 Estilos y Patrones Arquitectónicos.

En este epígrafe se describen los estilos y patrones arquitectónicos y se ponen algunos ejemplos para en capítulos posteriores analizar sus características y determinar cuál usar para la aplicación.

1.5.1 Estilos Arquitectónicos.

Según Mary Shaw y Paul Clements identifican los estilos arquitectónicos como: Un conjunto de reglas de diseño que identifica las clases de componentes y conectores

que se pueden utilizar para componer en sistema o módulo, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo. (9)

Los estilos arquitectónicos son la clave en la reducción del costo durante el desarrollo del software. Reutilizar soluciones efectivas es una de las prácticas fundamentales en el éxito del proceso de ingeniería, y lo mismo pasa en el contexto de la Arquitectura de Software. La reutilización a nivel de arquitectura se consigue con la adopción de estilos arquitectónicos.

A la hora de definir un estilo arquitectónico el campo de aplicación no es relevante, sólo se tiene en cuenta el patrón de organización general, los tipos de componentes presentes habitualmente en el estilo y las interacciones que se establecen entre ellos. Así, ejemplos de tipos de componentes son: clientes, servidores, filtros, niveles, bases de datos, etc., mientras que ejemplos de mecanismos de interacción son: llamadas a procedimientos o métodos, ya sean locales o remotas, paso de mensajes, protocolos de comunicación, etc. (8)

1.5.2 Patrones Arquitectónicos.

Un patrón de Arquitectura de Software describe un problema particular y recurrente del diseño, que surge en un contexto específico, y presenta un esquema genérico y probado de su solución. (11)

Un patrón de Arquitectura de Software es “un esquema genérico probado para solucionar un problema particular recurrente que surge en un cierto contexto. Este esquema se especifica describiendo las componentes, con sus responsabilidades, relaciones, y las formas en que colaboran. (12)

Ambas definiciones parecen expresar la misma idea. En los dos casos, se refieren al concepto de patrón de Arquitectura de Software como un esquema genérico, que dan solución a un problema particular. Sin embargo, la segunda definición está más completa porque aquí el autor especifica que dicho esquema se describe a través de componentes.

1.5.3 Ejemplos de Estilos y Patrones Arquitectónicos.

Seguidamente se muestran ejemplos de los principales estilos y patrones arquitectónicos que reflejan lo que anteriormente se abordaba.

Ejemplos:

- Estilos de Flujo de Datos.
  - Sistemas de filtros y tuberías.
  - Secuencial por lotes.
- Estilos Centrados en Datos.
  - Arquitecturas de Pizarra o Repositorio.
- Estilos de Llamada y Retorno.
  - Modelo-Vista-Controlador (MVC).
  - Arquitecturas en Capas.
  - Arquitecturas Orientadas a Objetos.
  - Arquitecturas Basadas en Componentes.
- Estilos Derivados.
  - C2.
  - GenVoca.
  - REST.
- Estilos de Código Móvil.
  - Arquitectura de Máquinas Virtuales.
- Estilos heterogéneos.
  - Sistemas de control de procesos.
  - Arquitecturas Basadas en Atributos.
- Estilos Peer-to-Peer.
  - Arquitecturas Basadas en Eventos.
  - Arquitecturas Orientadas a Servicios (SOA).
  - Arquitecturas Basadas en Recursos.

Como se ha estado viendo, los estilos se encuentran en el centro de la arquitectura y constituyen buena parte de su sustancia. Los patrones de arquitectura están claramente dentro de la disciplina arquitectónica, solapándose con los estilos, mientras

que los patrones de diseño se encuentran más bien en la periferia, si es que no decididamente afuera. (13)

1.6 Patrones de Diseño.

Los patrones de diseño tratan los problemas de diseño que se repiten y que se presentan en situaciones particulares del diseño, con el fin de proponer soluciones a ellas. Por lo tanto, los patrones de diseño son soluciones exitosas a problemas comunes basadas en la experiencia y que se ha demostrado que funcionan.

El uso de los patrones de diseño no es de carácter obligatorio, solo es aconsejable en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable. De hecho, forzar el uso de los patrones puede ser un error.

En el libro Design Patterns publicado en los años 90´s, escrito por los que comúnmente se conoce como GoF (Gang of Four, "pandilla de los cuatro"), compuesto por Erich Gamma, Richard Helm, Ralph Johnson y John Vlisides, se recopilaron y documentaron 23 patrones de diseño clasificados en tres grandes categorías.

A continuación una lista con los patrones de diseño a objetos más habituales publicados en el libro Design Patterns.

1.6.2 Patrones GoF.

- **Creacionales.**
  - Abstract Factory**.**
  - Builder.
  - Factory Method**.**
  - Prototype**.**
  - Singleton**.**
- **Estructurales.**
  - Adapter.
  - Bridge**.**
  - Composite**.**
  - Decorator**.**

- Facade.
- Flyweight.
- Proxy.

➢ **De Comportamiento**.

- Chain of Responsibility.
- Command.
- Interpreter.
- Iterator.
- Mediator.
- Memento.
- Observer.
- State.
- Strategy.
- Template Method.
- Visitor.

1.6.3 Patrones de Asignación de Responsabilidades.

Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.

Dentro de los patrones GRASP se pueden distinguir cinco patrones principales y cuatro adicionales.

➢ **GRASP principales**:

- Experto.
- Creador.
- Alta Cohesión.
- Bajo Acoplamiento.
- Controlador.

➢ **GRASP adicionales**:

- Fabricación Pura.

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

1.7 Frameworks de Desarrollo.

El término framework, se refiere a una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta. (16)

Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones. (16)

Un framework Web, por tanto, se puede definir como un conjunto de componentes (por ej. clases en java y descriptores y archivos de configuración en XML (Extensible Markup Language, lenguaje de marcas) que componen un diseño reutilizable que facilita y agiliza el desarrollo de sistemas Web. (16)

Un framework es por tanto una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Normalmente integra componentes variados para desarrollo de aplicaciones, pero dependen del lenguaje y ambiente de desarrollo. Por ej. Un framework para Java es Struts, un framework para Windows es .Net, un framework para PHP (PHP Hypertext Pre-processor, Pre-procesador de hipertexto) es Symfony, etc.

La razón fundamental por la que se hace uso de los frameworks de desarrollo es para no tener que comenzar el proyecto de cero y así ganar en tiempo y también en organización.

1.7.2 Características de los Frameworks. (16)

- **Abstracción de URL y sesiones**: no es necesario manipular directamente las URL ni las sesiones, el framework ya se encarga de hacerlo.
- **Acceso a datos**: incluyen las herramientas e interfaces necesarias para integrarse con herramientas de acceso a datos, XML, etc.
- **Controladores**: la mayoría de los frameworks implementa una serie de controladores para gestionar eventos, como una introducción de datos mediante un formulario o el acceso a una página. Estos controladores suelen ser fácilmente adaptables a las necesidades de un proyecto en concreto.
- **Autentificación y control de acceso**: incluyen mecanismos para la identificación de usuarios mediante login y password y permiten restringir el acceso a determinadas páginas a determinados usuarios.

1.7.3 Ejemplos de Frameworks para Aplicaciones Web.

- Frameworks para aplicaciones Web en PHP:
  - CakePHP.
  - Prado.
  - Symfony.
  - Kumbia.
  - ZendFW.

1.8 Lenguajes de Programación.

Si llamamos lenguaje al medio utilizado para el entendimiento y la interacción con otras personas, entonces un lenguaje de programación es aquel que te permite una interacción de un usuario con un computador. Los lenguajes de programación son más bien, aquellas herramientas que permiten crear programas y software. Facilitan así la tarea de programación, ya que disponen de formas adecuadas que permiten ser leídas y escritas por personas, a su vez resultan independientes del modelo de computador a utilizar.

1.8.1 Programación Orientada a Objetos.

La programación orientada a objetos maneja conceptos tales como, la encapsulación, la abstracción, la modularidad, la jerarquía de clases.

- **Encapsular:** hace referencia a reunir y controlar el grupo resultante como un todo y no individualmente.

- **La abstracción**: es un término externo al objeto, que controla la forma en que es visto por los demás.

- **La modularidad:** considera ventajoso dividir un gran sistema en varios módulos, pues en la programación orientada a objetos, un programa grande siempre será más complicado que la suma de varios programas pequeños.

- **La jerarquía de clases:** el código se ordena jerárquicamente por clases. El ejemplo más claro de la jerarquía es la herencia.

Otro aspecto de importancia en la programación orientada a objetos es la forma en que se intenta simular el mundo real a través del significado de los objetos que contienen características y funciones, y no solo se crean funciones sino que también se crean instancias. Todo ello permite que los programas sean más fáciles de escribir, mantener y reutilizar.

1.8.2 Lenguajes de lado del servidor.

Son aquellos lenguajes que son reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato comprensible para él.

- Ejemplos:

  - PHP.
  - Java.
  - ASP.net
  - XML.

1.8.3 Lenguajes de lado del cliente.

Los lenguajes de lado del cliente son aquellos que pueden ser directamente digeridos por el navegador y no necesitan un pre-tratamiento.

- Ejemplos:
  - HTML.
  - Java Script.
  - Applets de Java.
  - CSS.

1.9 Metodologías de Desarrollo.

Se entiende por metodología de desarrollo una colección de documentación formal referente a los procesos, las políticas y los procedimientos que intervienen en el desarrollo del software. (18)

La finalidad de una metodología de desarrollo es garantizar la eficacia (por ej. cumplir los requisitos iniciales) y la eficiencia (por ej. minimizar las pérdidas de tiempo) en el proceso de generación de software. (18)

Una metodología es una guía que se sigue, que va indicando qué hacer y cómo se debe actuar cuando se quiere obtener algún tipo de investigación. En términos informáticos, una metodología de desarrollo de software es un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevos software.

1.9.2 Clasificación de las Metodologías.

Las metodologías se clasifican en robustas y ágiles. Seguidamente se describen estas metodologías y se ponen algunos ejemplos.

1.9.2.1 Metodología Robusta.

Las metodologías robustas son aquellas en las que se hace una fuerte planificación del proyecto durante todo el proceso de desarrollo, encaminadas para proyectos de gran tamaño. Están divididas por etapas en las cuales se genera gran cantidad de

documentación. En este tipo de metodologías se realiza además un profundo desarrollo en el análisis y diseño de los sistemas antes de su construcción.

- Ejemplos:
    - RUP: Rational Unified Process (Proceso Unificado de Desarrollo).
    - MSF: Microsoft Solutions Framework (Marco de Soluciones Microsoft).

1.9.2.2 Metodologías Ágiles.

Las metodologías ágiles por otra parte están encaminadas para proyectos de menos volumen y en los que la documentación no juega el papel más importante como en con en el caso anterior. Son orientadas a las personas y no a los procesos.

- Ejemplos:
    - XP: Extreme Programming (Programación Extrema).
    - Scrum.
    - Family of Crystal Methodologies (Familia de Metodologías Crystal).

1.10 Herramientas CASE.

1.10.1 ¿Qué definimos cómo CASE?

Las herramientas CASE se acoplan con las metodologías para dar una forma de representar sistemas. Una herramienta CASE es una herramienta software que automatiza (al menos en parte) una tarea en particular del ciclo de vida del software.

CASE se define también como: (19)

- Conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases.
- La sigla genérica para una serie de programas y una filosofía de desarrollo de software que ayuda a automatizar el ciclo de vida de desarrollo de los sistemas.

- Una innovación en la organización, un concepto avanzado en la evolución de tecnología con un potencial de efecto profundo en la organización. Se puede ver al CASE como la unión de las herramientas automáticas de software y las metodologías de desarrollo de software formales.

El uso de las herramientas CASE aumenta la calidad del software desarrollado, debido a que una funcionalidad se basa en la comprobación automática de errores. Permite la reutilización de componentes software, como librerías. Acelera el proceso de desarrollo del software y permite además un desarrollo gradual e iterativo.

La principal ventaja de la utilización de una herramienta CASE, es la mejora de la calidad de los desarrollos realizados y, en segundo término, el aumento de la productividad. Para conseguir estos dos objetivos es conveniente contar con una organización y una metodología de trabajo, además de la propia herramienta. (19)

1.10.2 Ejemplos de Herramientas CASE.

- Ejemplos:
  - Visual Paradigm.
  - Rational Rose.

1.11 Sistemas Gestores de Base de Datos.

Un Sistema Gestor de Base de Datos (SGBD) es un conjunto de programas que permiten crear y mantener una base de datos (BD), asegurando su integridad, confidencialidad y seguridad. (20)

Los SGBD son software que le permiten a uno o más usuarios la utilización y/o actualización de los datos almacenaos en una o varias BD desde diferentes lugares y a la vez, de modo que los usuarios no tienen conocimiento de cómo es el modo de almacenamiento de los datos ni tampoco el método de acceso empleado para obtener dichos datos.

1.11.2 Ventajas y Desventajas de usar un Sistema Gestor de Base de Datos.

Al hacer uso de los SGBD se eliminan las inconsistencias en los datos. Esto es algo de mucha importancia, pues cuando los mismos datos se utilizan y actualizan en diferentes procesos, si no se tiene un SGBD se hace realmente difícil esta tarea. Permiten compartir los mismos datos entre diferentes aplicaciones con distintas necesidades al mismo tiempo. Con un SGBD se evita la redundancia, lo que hace que se ahorre en espacio de almacenamiento y se mejora en cuanto a la seguridad de los datos pues normalmente incorporan mecanismo de seguridad.

Las desventajas del uso de los SGBD viene dada por su gran complejidad, ya que son conjuntos de programas muy complejos y es necesario comprenderlos bien para poder entonces sacar lo mejor de ellos. El tamaño es otro de los inconvenientes pues requieren de una gran cantidad de espacio en disco duro y de memoria si se quiere trabajar de forma eficiente. Por otra parte está la vulnerabilidad a los fallos: el hecho de que todo esté centralizado en el SGBD hace que el sistema sea más vulnerable ante los fallos que puedan producirse.

1.11.3 Ejemplos de Sistemas Gestores de Base de Datos.

- Ejemplos:

  - Oracle.
  - DB2.
  - PostgreSQL.
  - MySQL.
  - SQL Server.

1.12 El Rol del Arquitecto de Software.

El Arquitecto es una persona, equipo u organización responsable por la arquitectura del sistema. (17)

Un Arquitecto de software es aquel que está involucrado con uno o varios desarrollos de software, pero desde un punto de vista macro. Es decir, comenzando por los requisitos (funcionales y no funcionales) y hasta llegar a conocer el impacto que tendrá en el hardware del cliente. (18)

Los arquitectos son necesarios para todo proyecto de software. Hoy día son vistos como uno de los integrantes fundamentales de los equipos de desarrollo de software empresarial. Esto se debe a que el Arquitecto de software es el principal tomador de decisiones respecto a la manera en que será construida la aplicación por los programadores del equipo.

De igual forma los arquitectos son los encargados de:

- Diseñar el sistema de software.
- Guiar el equipo de desarrollo en la aplicación del sistema.
- Prever y diagnosticar problemas.
- Encontrar/desarrollar soluciones a estos problemas.
- Determinar los componentes que deben utilizarse para construir el sistema
- Definir y asegurar el cumplimiento de estándares y mejores prácticas para el desarrollo y mantenimiento de aplicaciones.
- Definición y revisión de la arquitectura y diseño de sistemas nuevos teniendo en cuenta las necesidades y restricciones propias del negocio de la organización.

En sí, el principal rol del Arquitecto es asegurar el éxito del proyecto diseñando las bases de la aplicación. Esto incluye la definición tanto de la estructura organizacional como de la estructura física de su despliegue.

En el proyecto de Manugas se definen como actividades para el rol del arquitecto:

- Análisis de la arquitectura.
- Priorizar los Casos de Uso.
- Identificar mecanismos de diseño.
- Estructurar el modelo de implementación.
- Reutilización de elementos de diseño existentes.
- Identificar los elementos de diseño.
- Describir la arquitectura en tiempo de ejecución.

Como artefactos principales que debe generar el rol de arquitecto:

- En la fase de Inicio, flujo de trabajo Requerimientos generar la Plantilla DCS Especificación de los Requisitos no Funcionales.
- En la fase de Elaboración, flujo de trabajo de Análisis y Diseño generar la Plantilla DCS Documento Arquitectura de Software.
- En la fase de Transición, flujo de trabajo Despliegue generar la Plantilla DCS Modelo de Despliegue.

1.13 Conclusiones.

Al terminar este capítulo se abordaron temas relacionados con la Arquitectura de Software, entre ellos los estilos y patrones arquitectónicos. Además se analizaron algunos sistemas de Facturación y Cobro existentes con el objetivo de valorar las ventajas que aportaba cada uno para la realización del sistema. Se tuvo en cuenta también las principales características de las herramientas y tecnologías, para en capítulos posteriores proponer la arquitectura basada en ello.

Capítulo 2: Tecnologías a utilizar.

2.1 Introducción.

Este capítulo es el resultado del estudio realizado sobre las principales herramientas y tecnologías existentes con respecto a la Arquitectura de Software. Más bien se establece una comparación y se procede a seleccionar según las características más acordes al proyecto, con cuál de estas herramientas y tecnologías se modelará la arquitectura del sistema en desarrollo. Específicamente se analizan los estilos y patrones arquitectónicos, las metodologías de desarrollo, las herramientas CASE y lenguaje de modelado, lenguaje de programación, gestor de bases de datos y otras herramientas necesarias para lograr la evolución del proyecto**.**

2.2 Metodologías de Desarrollo.

Siempre que se vaya a desarrollar un software es necesario usar una metodología que guíe el proceso. A continuación se muestran algunas características de las metodologías más utilizadas en la actualidad, seleccionando finalmente cuál será usada para modelar la aplicación Manugas.

2.2.1 Extreme Programing.

La Programación Extrema (XP) surge ideada por Kent Beck, como proceso de creación de software diferente al convencional. Es la más destacada de los procesos ágiles de desarrollo de software. En palabras de Beck: "XP es una metodología ligera, eficiente, con bajo riesgo, flexible, predecible y divertida para desarrollar software". (Ver Anexo #2)

XP se basa en la simplicidad, la comunicación y el reciclado continuo de código. Para algunos no es más que aplicar una pura lógica. Lo que buscan en definitiva es la reducción de costes.

Características de XP. (25)

- **Pruebas Unitarias:** se basa en las pruebas realizadas a los principales procesos, de tal manera que si se adelanta en algo hacia el futuro, pueden hacer pruebas de las fallas que pudieran ocurrir. Es como si se adelantara a obtener los posibles errores.

- **Refabricación:** se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- **Programación en pares:** una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa.

¿Qué es lo que propone XP? (25)

1- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
2- El manejo del cambio se convierte en parte sustantiva del proceso.
3- El costo del cambio no depende de la fase o etapa.
4- No introduce funcionalidades antes que sean necesarias.
5- El cliente o el usuario se convierte en miembro del equipo.

2.2.2 Microsoft Solution Framework.

Microsoft Solution Framework (MSF) se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. Es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. La Metodología MSF se adapta a proyectos de cualquier dimensión y de cualquier tecnología. (55)

Características de MSF. (25)

- **Adaptable:** es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- **Escalable:** puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas o más.
- **Flexible:** es utilizada en el ambiente de desarrollo de cualquier cliente.
- **Tecnología Agnóstica:** porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

MSF se compone de varios modelos que se encargan de planificar las diferentes partes implicadas en el desarrollo de un proyecto. Define las pautas para construir proyectos empresariales a través del lanzamiento de versiones. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles. Acorta el tiempo de entrega y minimiza los riegos para lograr un mejor control del proyecto. Además de ello este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar. (56)

2.2.3 Rational Unified Process (RUP).

RUP es una metodología para la ingeniería de software que va más allá del mero análisis y diseño orientado a objetos para proporcionar una familia de técnicas que soportan el ciclo completo de desarrollo de software. El resultado es un proceso dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental. (5)

RUP es un proceso que define claramente “QUIÉN” está haciendo “QUÉ”, “CUÁNDO”, y “CÓMO” para alcanzar un determinado objetivo. Utiliza para ello el Lenguaje Unificado de Modelado (Unified Modeling Languaje, UML) para preparar todos los esquemas de un sistema de software.

Características de RUP.

- **Dirigido por Casos de Uso**: los casos de uso son el resultado o mapeo de los requisitos extraídos del negocio y representan piezas de funcionalidad que brindan un resultado de valor al usuario. Guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes workflows (flujos de trabajo), representan la realización de los casos de uso (cómo se llevan a cabo).
- **Centrado en la Arquitectura**: comprende los aspectos estáticos y dinámicos más importantes del sistema.
- **Iterativo e Incremental**: el trabajo se divide en piezas pequeñas o mini proyectos.

RUP divide el proceso en 4 fases: Inicio, Elaboración, Construcción y Transición, dentro de las cuales se realizan varias iteraciones según el proyecto y en las que se

hace un mayor o menor hincapié en las distintas actividades. Agrupa estas actividades o grupos lógicos en 9 flujos de trabajo, los 6 primeros conocidos como flujos de ingeniería: Modelación de Negocio, Requerimientos, Análisis y Diseño, Implementación, Prueba y Despliegue y los 3 últimos conocidos como flujos de apoyo: Configuración y Administración de Cambios, Administración de Proyectos y Ambiente. (Ver Anexo #3)

Ciclo de vida de RUP.

La vida de un sistema transcurre a través de ciclos de desarrollo, desde su nacimiento hasta su muerte. En cada ciclo se repite el proceso unificado de desarrollo. Cada ciclo consta de 4 fases: Inicio. Elaboración, Construcción y Transición y cada una de estas fases se subdivide en iteraciones. Cada ciclo concluye con una versión del producto. Al finalizar el ciclo, se obtiene una nueva versión del sistema. Cada versión es un producto terminado que incluye requisitos, casos de uso, especificaciones no funcionales y casos de prueba.

Las 4+1 Vistas.

Es conveniente ver el sistema desde diferentes perspectivas para comprender mejor el diseño por lo que la arquitectura se representa mediante varias vistas que se centran en aspectos concretos del sistema, abstrayéndose de los demás. Para RUP, todas las vistas juntas forman el llamado modelo 4+1 de la arquitectura, el cual recibe este nombre porque lo forman la vista lógica, de implementación, de proceso y de despliegue, más la vista de casos de uso que es la que da cohesión a todas. (Ver Anexo #4)

2.2.4 ¿Por qué RUP?

Primero que nada se seleccionó RUP como metodología de desarrollo porque es una metodología robusta que a diferencia de las metodologías ágiles, no requiere de la presencia del cliente como parte del equipo de desarrollo, es decir se puede desarrollar la aplicación con el cliente a distancia, y eso en el caso de Manugas ha sido primordial ya que no se ha contado con la presencia del cliente a tiempo completo. Además, RUP es más apropiado para proyectos grandes, como Manugas, dado que requiere un equipo de trabajo capaz de administrar un proceso complejo en

varias etapas. En proyectos pequeños, es posible que no se pueda cubrir los costos de dedicación del equipo de profesionales necesarios.

Hoy día se ve como cada vez se hace mayor la complejidad solicitada para los sistemas de software, de forma que ya no es posible trabajar secuencialmente: definir primero el problema completo, luego diseñar toda la solución, construir el software y finalmente, testear el producto. Es necesario un enfoque iterativo, que permita una comprensión creciente del problema a través de refinamientos sucesivos, llegando a una solución efectiva luego de múltiples iteraciones acotadas en complejidad y precisamente esto lo permite RUP.

Algunas características de esta metodología y que pueden ayudar a entender mejor el por qué de su selección, son:

- RUP constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.
- No es un sistema con pasos firmemente establecidos, sino que es un conjunto de metodologías adaptables al contexto y necesidades de cada organización, por lo que nos ha permitido adaptarlas a las necesidades que impone el producto que se desarrolla.
- RUP, como metodología se rige por 5 procesos básicos: Adaptar al proceso, establecer prioridades, demostrar valor iterativamente, elevar el nivel de abstracción y el enfoque en la calidad.
- Administra de una forma muy eficiente los requisitos y el control de cambios.

Al escoger RUP como metodología de desarrollo se hace necesario entonces detallar cada una de las vistas que describen la arquitectura. En el capítulo 3 se hace referencia a este aspecto.

2.3 Lenguaje Unificado de Modelado.

El Lenguaje Unificado de Modelado (UML) es un lenguaje estándar para el modelado de software. Específicamente es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra gran cantidad de software.

Permite a los desarrolladores visualizar el producto de su trabajo (Artefactos) en esquemas o diagramas estandarizados, o sea que permite combinar diversos elementos gráficos y crear diagramas. Permite además la modelación de sistemas con tecnología orientada a objetos.

Es un lenguaje de representación visual, que describe lo que supuestamente hará un sistema, pero no dice cómo implementarlo.

UML está compuesto por:

- **Elementos**: abstracciones que constituyen los bloques básicos de construcción.
- **Diagramas**: es la representación de un conjunto de elementos: visualizan un sistema desde diferentes perspectivas.
- **Relaciones:** ligan los elementos.

El UML se ha vuelto el estándar de facto (impuesto por la industria y los usuarios) para el modelado de aplicaciones de software. En los últimos años, su popularidad trascendió al desarrollo de software y, en la actualidad, el UML es utilizado para modelar muchos otros dominios, como por ejemplo el modelado de procesos de negocios.

Por tal motivo el UML se utiliza cada vez más y también porque:

- Acelera el proceso de los requisitos.
- Reduce la pérdida de información entre el proceso de requisitos y el de diseño, y entre el diseño y la implementación.
- Comunicación: lenguaje más claro y natural, mayor nivel de precisión, pero ahorra detalles.
- Compatible con el desarrollo iterativo (por ej., modelo de la espiral).
- Acepta requisitos y diseño de alto nivel en las primeras espirales y diseño y requisitos más detallados en fases posteriores.

Existen otros lenguajes de modelado, pero son específicos, tales como: BPMN, SysML, SDL. BPMN es para modelar software de arquitectura orientada a servicios y hay que combinarlo con otro lenguaje de modelado, porque no es completo, generalmente con UML. SysML se usa para modelar proyectos de ingeniería de sistemas y sistemas de sistemas. SDL está diseñado para la especificación de sistemas complejos, interactivos, orientados a eventos, de tiempo real o que presenten un comportamiento paralelo, y donde módulos o entidades independientes se comuniquen por medio de señales para efectuar su función. En fin es un lenguaje para especificar sistemas de telecomunicaciones.

2.4 Herramienta CASE.

A continuación se muestra la Tabla #1 donde se establece una comparación entre dos de las principales y más usadas herramientas CASE para el desarrollo de software.

Tabla # 1. Comparación entre Herramientas CASE: Rational Rose vs. Visual Paradigm

Características	**Rational Rose**	**Visual Paradigm**
Integraciones IDE	Borland JBuilder (versión 7.0 en adelante), Microsoft Visual Studio (versiones 2003 en adelante)	MS Visio, Plug-in, Visual Studio, IntelliJ IDEA, Eclipse, NetBeans
Multiplataforma	No (se recomienda utilizar Windows 2000, Windows NT y Windows XP)	Si (la escuela paga la licencia)
Orientado a Objetos	Sí	Sí
Generación de documentación	No	Si (automáticamente en varios formatos como Web o .pdf)

Soporte de UML	Sí	Sí
Control de versiones	No	Sí

2.4.1 ¿Por qué Visual Paradigm?

Como se puede ver, las dos herramientas tienen características en común, que las hacen una opción deseable para cualquier proyecto, pero para el caso específico de Manugas, Visual Paradigm es la herramienta que se adecua a las necesidades del negocio.

Primeramente se tuvo en cuenta que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite además dibujar todos los tipos de diagramas de clases. Es multiplataforma. Soporta múltiples usuarios trabajando sobre el mismo proyecto. Genera automáticamente la documentación del proyecto en varios formatos. Permite el control de versiones y la ingeniería inversa e incluye el modelado colaborativo con CVS (Concurrent Versions System, Sistema Concurrente de Versiones) y Subversion, así como la integración con MS Visio.

2.5 Lenguaje de Programación.

Cuando se va a desarrollar una aplicación web, hay que tener en cuenta los lenguajes de lado del servidor y los lenguajes de lado del cliente.

2.5.1 Lenguajes de lado del servidor.

Como lenguaje de lado del servidor se analizaron Java y PHP y se escogió finalmente para el desarrollo de la aplicación PHP. A continuación se muestran las características de cada uno de estos lenguajes y el por qué PHP.

2.5.1.1 Java.

Java es un lenguaje orientado a objetos, eso implica que su concepción es muy próxima a la forma de pensar humana. Posee otras características muy importantes como por ej. Es un lenguaje que es compilado, multiplataforma. Gracias al API de java podemos ampliar el lenguaje para que sea capaz de comunicarse con equipos

mediante red, acceder a BD, crear páginas HTML dinámicas, crear aplicaciones visuales al estilo Windows, etc.

Java reduce un 50% los errores más comunes de programación con lenguajes como C y C++ al eliminar muchas de las características de éstos, entre las que destacan:

- Aritmética de punteros.
- No existen referencias.
- Registros (struct).
- Definición de tipos (typedef).
- Necesidad de liberar memoria (free).

Java es un lenguaje como C/C++ o Smalltalk para realizar cualquier aplicación que se puede programar, las páginas web solamente son una forma de usar interface. Con Java se puede crear aplicaciones desktop para diferentes plataformas con Swing y RCP (Rich Client Platform o Plataforma de Clientes Ricos), se puede crear también applets para el browser.

2.5.1.2 PHP.

PHP es un lenguaje de programación del lado del servidor gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación. (33)

Un lenguaje del lado del servidor es aquel que se ejecuta en el servidor web, justo antes de que se envíe la página a través de Internet al cliente. Las páginas que se ejecutan en el servidor pueden realizar accesos a BD, conexiones en red, y otras tareas para crear la página final que verá el cliente. El cliente solamente recibe una página con el código HTML resultante de la ejecución de la PHP. Como la página resultante contiene únicamente código HTML, es compatible con todos los navegadores. (33)

PHP es una pieza trascendental de los denominados sistemas LAMP, que son usados con frecuencia para equipar servidores web muy potentes y con un bajo coste. LAMP es el acrónimo de Linux (el Sistema Operativo), Apache (el Servidor Web), MySQL (el

Gestor de BD) y PHP; Perl o Peython (lenguajes de programación), y se basa principalmente en estos componentes.

Una de sus características más potentes es su suporte para gran cantidad de BD. Ofrece la integración con varias bibliotecas externas, que permiten que el desarrollador haga casi cualquier cosa desde generar documentos en pdf hasta analizar código XML. PHP también ofrece una solución simple y universal para las paginaciones dinámicas del web de fácil programación. Su diseño elegante lo hace perceptiblemente más fácil de mantener que otros lenguajes. (36)

Es principalmente una tecnología para páginas web y por tal motivo requiere en primer lugar instalar un servidor web. Para su funcionamiento necesita de Apache o IIS con las librerías de PHP. Todo el trabajo lo realiza el servidor y no delega al cliente, por tanto puede ser más ineficiente a medida que las solicitudes aumenten de número. La legibilidad del código puede verse afectada al mezclar sentencias HTML y PHP. Dificulta la modularización y la organización por capas de la aplicación. (36)

2.5.1.3 ¿Por qué PHP?

Para la selección de este lenguaje se tuvo en cuenta que fuese un lenguaje con tecnología orientada a objetos, debido a que la metodología seleccionada así lo exige. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar. Además de ello, el equipo de desarrollo del proyecto Manugas está preparado para trabajar con este lenguaje, de escoger otro se hubiese tenido que comenzar una capacitación, lo cual es desfavorable pues consume tiempo y recursos humanos.

Otra razón por la que se eligió es porque PHP es un lenguaje multiplataforma, y esto da la posibilidad de si en un futuro se migra a software libre, poder entonces trabajar con otro sistema operativo diferente a Windows. Es libre, por lo que se presenta como una alternativa de fácil acceso para todos. Abierto a diferentes arquitecturas y paradigmas de programación y además, tiene comunicación directa con distintas BD.

2.5.2 Lenguajes de lado del cliente.

Los lenguajes de lado del cliente que se usaron para el desarrollo de la aplicación fueron HTML y Java Script. A continuación se dan algunas características que respaldan el por qué la selección de estos lenguajes.

2.5.2.1 ¿Por qué HTML?

El HTML es un lenguaje de marcas hipertextuales, un lenguaje diseñado para estructurar textos para generar páginas web. Gracias a Internet y a los navegadores web, el HTML se ha convertido en el formato más fácil para la creación de páginas web debido a su sencillez. La mayoría de las etiquetas del lenguaje HTML son semánticas. La interpretación de las etiquetas es realizada por el navegador web. El lenguaje HTML es extensible, se le pueden añadir características, etiquetas y funciones adicionales para el diseño de páginas web, generando un producto vistoso, rápido y sencillo. (58)

Como bien se cita en el párrafo anterior, HTML es un lenguaje para generar páginas web de forma sencilla. Es extensible, es decir, se le pueden agregar nuevas características que hacen el trabajo más cómodo para los desarrolladores y se obtiene por tanto, un producto más vistoso, rápido y sencillo, requisitos necesarios para lograr la aceptación del cliente. Es muy flexible, o sea que se adapta a nuevos usos y responde a los cambios en lugar de estancarse.

Por todo lo antes expuesto y teniendo en cuenta que el equipo de desarrollo del proyecto Manugas estaba capacitado para el trabajo con este lenguaje y que el framework de desarrollo con el cual se está modelando la aplicación Symfony define por defecto HTML como lenguaje de lado del cliente, entonces se decidió usa por ser este uno de los más factibles para el desarrollo e implementación del sistema.

2.5.2.2 ¿Por qué Java Script?

Java Script es un lenguaje de programación utilizado para crear pequeños programitas encargados de realizar acciones dentro del ámbito de una página web. Se trata de un lenguaje de programación del lado del cliente, porque es el navegador el que soporta la carga de procesamiento. (57)

Se escogió Java Script por ser un lenguaje sencillo que da la posibilidad a las personas que no tengan una experiencia previa en la programación aprender este

lenguaje con facilidad y utilizarlo en toda su potencia con sólo un poco de práctica. Además es un lenguaje rápido que permite la creación de efectos especiales sobre páginas web, elemento que es sumamente importante para el desarrollo del proyecto pues se necesita la creación de contenidos dinámicos y elementos de la página que tengan movimiento, cambien de color o cualquier otro dinamismo para lograr una mayor aceptación del cliente. Permite ejecutar instrucciones como respuesta a las acciones del usuario, con lo que podemos crear páginas interactivas con programas como calculadoras, agendas, o tablas de cálculo que actualmente no es objetivo de la aplicación pero para una segunda versión entonces se tendrán en cuenta estos elementos.

Gracias a su compatibilidad con la mayoría de los navegadores modernos, Java Script es el lenguaje de programación del lado del cliente más utilizado, razón además que se tuvo en cuenta para la selección del mismo, así como que es un lenguaje que permite la programación tanto de pequeños script como de programas más grandes orientados a objetos que es el caso de la aplicación Manugas.

2.6 Framework de Desarrollo.

A continuación se muestra en la Tabla #2 una comparación entre algunos de los frameworks de desarrollo más usados para aplicaciones Web en PHP.

Tabla # 2. Comparación entre Frameworks de Desarrollo: Cake vs. Symfony vs. ZendFW

Características	Cake	Symfony	ZendFW
Incorporación del patrón MVC.	x	x	x
Independiente del manejador de BD.	x	x	x
Generación de URL amigables.	x	x	x
Generación de código PHP.	x	x	
Manual de referencia.	x	x	x
Soporte para envío de correo electrónico.		x	x
Generación de archivos PDF.	x	x	x

Soporte para PHP4.	x		
Soporte para PHP5.	x	x	x
Mapeo de objetos relacionales (ORM).	x	x	
Licencias libres.	x	x	BSD
Validación de información.	x	x	x

2.6.1 ¿Por qué Symfony?

Por las condiciones del negocio se requiere de un framework de desarrollo que sea independiente del SGBD y así de la posibilidad en un futuro migrar a software libre; que implemente como patrón arquitectónico MVC; que tenga una interfaz amigable; que permita el envío de correo electrónico y genere archivos PDF; con soporte para PHP5 o versiones superiores; que permita la validación de la información y posea una licencia libre.

Como se pudo apreciar en la Tabla #2 Symfony cumple con todas estas características, al igual que CakePHP y ZendFW, pero Symfony a diferencia de estos dos frameworks, tiene la ventaja de que puede ser completamente personalizado para cumplir con los requisitos de las empresas que disponen de sus propias políticas y reglas para la gestión de proyectos y la programación de aplicaciones. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación Web compleja y es lo suficientemente estable como para desarrollar aplicaciones a largo plazo.

Además de ello, una de las principales ventajas de usar un framework completo como lo es Symfony, es que te permite estandarizar tus desarrollos y esto se debe a la estructura de archivos y directorios por defecto de él, pues cualquier programador con ciertos conocimientos de Symfony es capaz de continuar el desarrollo de cualquier proyecto, pudiendo a su vez profundizar en el código, solucionar errores y hasta añadir nuevas características.

2.7 Estilos Arquitectónicos.

Con la influencia de los estilos, los ingenieros claramente pueden obtener conocimiento sobre patrones y herramientas de ingeniería que implican una reducción significativa de costos. Si la arquitectura se desarrolla basada en los intereses de los interesados, las características deseadas de la aplicación emergerán con la selección del estilo que incluya cualidades que se acerquen a solucionar los problemas más importantes identificados por el sistema de los mismos.

2.7.1 Estilos de Flujos de Datos.

Este tipo de estilo es ideal para realizar transformaciones de datos dividiéndolos en pasos sucesivos. Cuenta con dos tipos de componentes. El primer tipo de componente son los elementos que realizan las transformaciones de los datos cuando pasan por ellos, los datos pasan de forma secuencial de uno a otro componente, realizando el procesamiento de los datos de forma secuencial. Estos elementos son independientes unos de otros. Todos tienen una o varias puertas de entrada-salida que sirven para mandar los datos modificados de un módulo a otro. El otro tipo de elemento es el que se encarga de enviar el flujo de los datos desde la salida de un módulo transformador a la entrada de otro, es decir, realiza el rol de comunicador. (40)

2.7.1.1 Sistemas de Filtros y Tuberías.

En este estilo arquitectónico, los componentes (llamados filtros) interactúan con el mundo exterior sólo a través de una interface de entrada y salida. Los datos son transferidos a través de esta interface en forma de secuencias de caracteres, aunque el formato específico de esos conjuntos depende de la aplicación. Estos conjuntos están conectados a los componentes por medio de los conectores (llamados tuberías). (42) (Ver Anexo #5)

Ventajas e Inconvenientes.

Las principales ventajas de este patrón están dadas por su facilidad de mantenimiento y mejora; la facilidad de diagnóstico (rendimiento); permite la reutilización: los filtros son independientes de sus vecinos; permite además entender el sistema global en combinación de componentes y soporta la ejecución concurrente.

Así mismo, hay que resaltar los inconvenientes que presenta. En primer lugar no es posible tener un sistema interactivo: para borrar una línea requeriría que los filtros

compartan un almacén de datos compartidos. Y por otra parte hace uso de espacio ineficiente: cada filtro debe copiar todos los datos a sus puertos de salida.

2.7.2 Centrado en los Datos.

El estilo centrado en datos resulta apropiado para sistemas que se centran en el acceso y actualización de datos en estructuras de almacenamiento que son compartidos por un número indefinido de componentes consumidores. Una de las propiedades más destacable de este estilo arquitectónico es la necesidad de crear persistencia de los datos almacenados. Existen dos tipos de componentes, un componente que realiza el almacenaje y persistencia de los datos y otros componentes que interactúan con dichos datos. En estos sistemas hay que tener en cuenta que se pueden producir problemas en la interacción de los componentes que manejan los datos con el componente que almacena los datos (creación, lectura y actualización simultáneas a un mismo dato) y la coherencia (hay que evitar que un componente pueda tener un dato con un valor desactualizado). Pueden existir módulos de almacenamiento activos o pasivos según quien se encargue de la notificación de la actualización de los datos (si el componente encargado del almacenamiento es el notificador, entonces este es un modelo activo). (40)

2.7.2.1 Arquitectura de Pizarra o Repositorio.

En esta arquitectura se pueden encontrar dos componentes principales: una estructura de datos que representa el estado actual, y un número independiente de componentes, los cuales realizan sus operaciones sobre él. La misma se ha usado en aplicaciones que necesitan interpretación de proceso de señales bastante complejo. También es usada en implementaciones de estilos de procesos en lotes de BD. Y por último en programación organizada como colecciones de servicios alrededor de un repositorio común. (40) (Ver Anexo #6)

Ventajas e Inconvenientes.

Muchas son las ventajas de usar esta arquitectura. En primer lugar brinda la posibilidad de presentar a los módulos de la capa de aplicación una interfaz común que se abstraiga de la diversidad de dispositivos del entorno físico; además de ello facilita el trabajo de documentación, gracias a la capacidad de formatear texto intrínseco al lenguaje; funciona muy bien con los problemas no deterministas (especial

para Inteligencia Artificial); permite que la implementación sea independiente del lenguaje de programación elegido.

Los inconvenientes más notables de esta arquitectura son respecto a los problemas de seguridad, pues la pizarra es accesible por todos y existen problemas de sincronización al chequear la pizarra.

2.7.2 Estilos de Llamada y Retorno.

Este estilo es interesante en los sistemas que se centran en la interacción de un usuario con el propio sistema. Podemos dividir la arquitectura en dos partes, la primera representa la interfaz del usuario con el que este realiza la llamada al sistema, la segunda contiene la lógica de negocio que se realiza tras la correspondiente llamada del usuario. Esta familia de estilos enfatiza la modificabilidad, la escalabilidad, la reusabilidad y la usabilidad del sistema. Al diseñar una arquitectura de este tipo es importante saber qué datos son los que servirán para interactuar con el usuario y cuales servirán solo como lógica de aplicación. En esta familia de arquitecturas se destacan los patrones Modelo-Vista-Controlador y arquitectura en tres capas. (40)

2.7.2.1 Modelo-Vista-Controlador (MVC).

El patrón conocido como Modelo-Vista-Controlador separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes. (Ver Anexo #7)

- El **Modelo** es la representación de los datos de la aplicación y del código. Coordina la lógica de la aplicación, acceso a BD y otras partes no visuales del sistema.

- La **Vista** es la representación visual de los datos. Se encarga de la interacción con el usuario.

- El **Controlador** es el intermediario entre las otras dos partes. Es responsable de desplegar la vista adecuada al usuario.

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo independientemente de la representación visual. (43)

Ventajas e Inconvenientes.

Entre las ventajas e inconvenientes de este patrón señaladas en la documentación de Patterns & Practices de Microsoft están las siguientes: (43)

Ventajas:

- **Adaptación al cambio:** los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares o PDA (Personal Digital Assistant, Asistente Digital Personal). Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo. Este patrón sentó las bases para especializaciones ulteriores, tales como Page Controller y Front Controller.

Inconvenientes:

- **Complejidad:** el patrón introduce nuevos niveles de indirección y por lo tanto aumenta ligeramente la complejidad de la solución. También se profundiza la orientación a eventos del código de la interfaz de usuario, que puede llegar a ser difícil de depurar. En rigor, la configuración basada en eventos de dicha interfaz corresponde a un estilo particular (arquitectura basada en eventos) que aquí se examina por separado.

- **Costo de actualizaciones frecuentes:** desacoplar el modelo de la vista no significa que los desarrolladores del modelo puedan ignorar la naturaleza de las vistas. Si el modelo experimenta cambios frecuentes, por ejemplo, podrían desbordar las vistas con una lluvia de requerimientos de actualización. Hace pocos años sucedía que algunas vistas, tales como las pantallas gráficas, involucraban más tiempo para plasmar el dibujo que el que demandaban los nuevos requerimientos de actualización.

2.7.2.2 Arquitecturas en Capas.

El patrón en capas indica que las capas superiores (o las capas más dependientes de la aplicación) tienen dependencias hacia abajo, pero las capas de abajo (más generales) no tienen dependencias hacia arriba. Provocando que las capas de arriba hacia abajo puedan ser intercambiadas (porque no están acopladas) con un esfuerzo menor.

El objetivo primordial de esta arquitectura es la separación de la lógica de negocios de la lógica de diseño. Un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario. (Ver Anexo #8)

- La **Capa de presentación:** es donde se le presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso. Se comunica únicamente con la capa de negocio. También es conocida como “capa de usuario” o “interfaz gráfica”.

- La **Capa de negocio:** es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso lógica de negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse.

- La **Capa de datos:** es donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de BD que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

La arquitectura en capas se puede dividir en una, dos, tres o cuatro capas. En este caso, se analizan las principales ventajas e inconvenientes del uso de la arquitectura en tres capas.

Arquitectura en 3 Capas. Ventajas e Inconvenientes. (40)

El modelo en 3 capas facilita que se pueda descomponer la aplicación en varios niveles de abstracción. De esta forma facilita la evolución del sistema, ya que los cambios solo deben afectar a la capa donde se encuentre la modificación. Si la interfaz accede a la misma función (por ej. varios lugares para identificarse), no se repetirá

código, lo que conlleva a una mayor facilidad de mantenimiento de la aplicación. Si se añade un nuevo formulario no ocasionará verdaderos quebraderos de cabeza, ya que los formularios ya no dependen unos de otros, con lo que el cambiar el workflow es algo trivial. El formulario ya no accede de forma independiente a los datos, ya que no se accede a través de ellos, al modificar los datos, esto implica que no se nos mostrará diferencia alguna.

Los inconvenientes más notables de este modelo en 3 capas son que no todo sistema podrá ser estructurado en capas y aún pudiendo serlo, la separación entre una y otra capa no es trivial. Ya no solo porque para un desarrollador no lo es, sino también porque muchos lenguajes y/o frameworks no están preparados para ello.

2.7.3 Estilos Peer-to-Peer.

Esta familia, también llamada de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente. (43)

2.7.3.1 Arquitecturas Orientadas a Servicios (SOA).

La Arquitectura SOA establece un marco de diseño para la integración de aplicaciones independientes de manera que desde la red pueda accederse a sus funcionalidades, las cuales se ofrecen como servicios. La forma más habitual de implementarla es mediante servicios Web, una tecnología basada en estándares e independiente de la plataforma, con la que SOA puede descomponer aplicaciones monolíticas en un conjunto de servicios e implementar esta funcionalidad en forma modular. (11)

En sí SOA es una estrategia que pretende la construcción de todos los componentes software de una compañía utilizando una metodología de desarrollo orientada a servicios y un soporte tecnológico que lo haga posible.

SOA define las siguientes capas de software:

- Las **aplicaciones básicas** son los sistemas desarrollados bajo cualquier arquitectura o tecnología, geográficamente dispersos y bajo cualquier figura de propiedad.

- La **capa de exposición de funcionalidades** es donde las funcionalidades de la capa aplicativas son expuestas en forma de servicios (servicios Web).

- La **capa de integración de servicios** que facilitan el intercambio de datos entre elementos de la capa aplicativa orientada a procesos empresariales internos o en colaboración.

- La de **composición de procesos** que define el proceso en términos del negocio y sus necesidades, y que varía en función del negocio.

- La **capa de entrega** donde los servicios son desplegados a los usuarios finales.

2.7.4 ¿Por qué Modelo-Vista-Controlador?

La base arquitectónica del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado (Producto: Manugas) se ha modelado haciendo uso del Patrón Arquitectónico MVC2 que es una implementación del MVC modificada.

El mayor cambio del MVC2 con respecto al MVC es que el modelo ya no dispara los eventos a sus vistas, o sea que logra la independencia de las vistas con el modelo, eliminando la eventualización del modelo hacia las vistas, aunque este aún tiene acceso a formularios que se encuentran en el modelo. La comunicación de las vistas con el modelo se realiza solamente por medio del controlador.

Este patrón es el más indicado para el desarrollo del producto Manugas por los beneficios que aporta. Entre las ventajas más notables para el desarrollo de la aplicación están las siguientes:

- Aislamiento entre las diferentes capas. Por ej. Si la vista es una aplicación Web basada en Java Script (JS) y se quiere cambiar el modelo, para que acceda a otra BD, la vista no será afectada por el cambio.

- Permite utilizar los mismos objetos del modelo para diferentes vistas. Por ej. Se puede hacer que la aplicación tenga dos tipos de presentación: una en HTML

para visualizarla en un navegador y otra en XML para exportarla. El controlador podrá decidir qué vista presentar. Esta ventaja se ve muy útil en la parte del módulo de facturación de la aplicación.

- Soporte de múltiples vistas. Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Por ej. Múltiples páginas de una aplicación Web pueden utilizar el mismo modelo de objetos mostrado de maneras diferentes.

- Adaptación al cambio. Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo.

Otra de las razones por las que se escogió dicho patrón arquitectónico es porque el framework de desarrollo que se utiliza en el proyecto es Symfony que está basado en este patrón, específicamente implementa la arquitectura MVC2, con el objetivo de utilizar la separación de las responsabilidades de cada una de las capas lógicas que lo conforman y así lograr facilidades en el desarrollo y la reutilización.

A continuación en la Figura #2 se muestra el flujo de mensajes del patrón MVC según Symfony para Manugas.

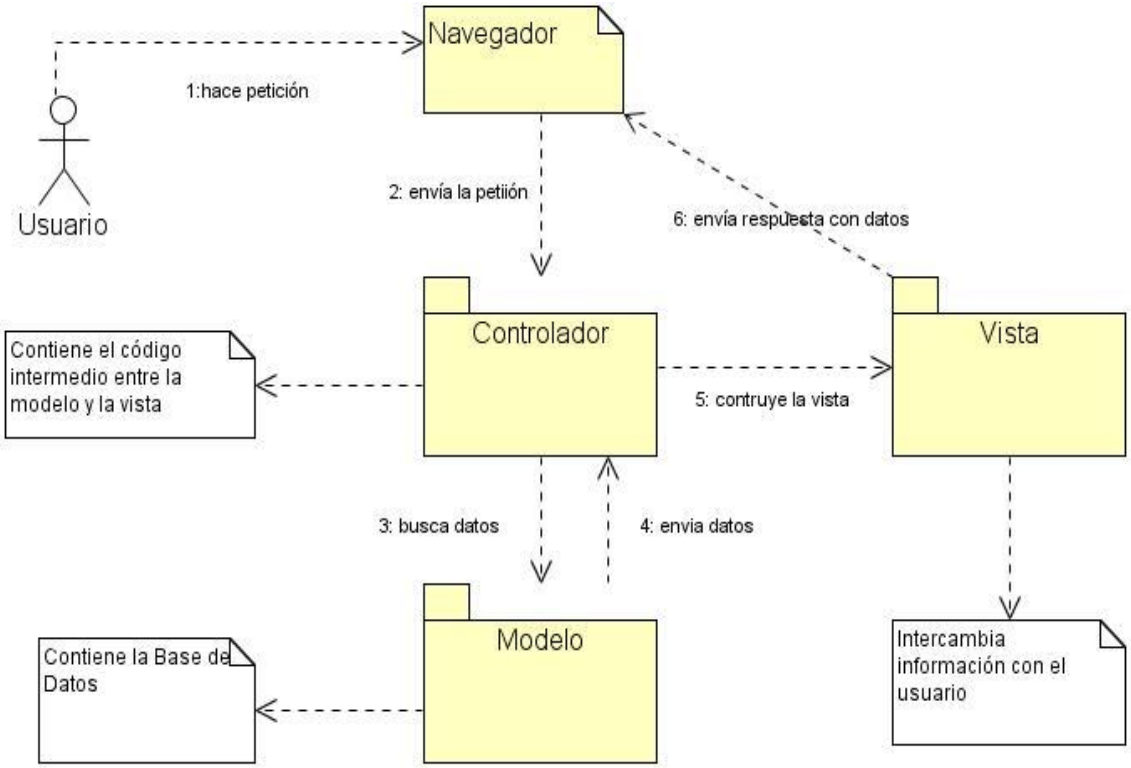


Figura # 2. Flujo de Mensajes del Patrón MVC para Manugas

2.8 Patrones de Diseño.

Para el desarrollo de la aplicación se hizo uso de los siguientes patrones de diseño:

Patrones GRAPS.

- Experto: es un patrón que se usa más que cualquier otro al asignar responsabilidades en un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña, expresa simplemente la intuición de que los objetos hacen cosas relacionadas con la información que poseen.

- Bajo Acoplamiento: el bajo acoplamiento es un principio que debemos recordar durante las decisiones de diseño pues es la meta principal que es preciso tener presente siempre. Es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño.

- Alta Cohesión: una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.

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

- Controlador: este patrón ofrece una guía para tomar decisiones apropiadas que generalmente se aceptan. La misma clase controlador debería utilizarse con todos los eventos sistémicos de un caso de uso, de modo que se puede conservar la información referente al estado del caso de uso.

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

Patrones GoF.

Dentro de los patrones GoF que quedaron definidos en el capítulo anterior, se hará uso de los que a continuación se muestran:

Creacionales.

- Singleton (Instancia única): garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. En el controlador frontal hay una llamada a sfContext: getInstance (). En una acción, el método getContext (), un objeto muy útil que guarda una referencia a todos los objetos del núcleo de Symfony.
- Abstract Factory (Fábrica abstracta): permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando. Cuando el framework necesita por ejemplo crear un nuevo objeto para una petición, busca en la definición de la factoría el nombre de la clase que se debe utilizar para esta tarea.

Estructurales.

- Decorator (Envoltorio): añade funcionalidad a una clase dinámicamente. El archivo layout.php que también se denomina plantilla global, almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el layout, o si se mira desde el otro punto de vista, el layout decora la plantilla.
- Composite (Objeto compuesto): permite tratar objetos compuestos como si de uno simple se tratara. Sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol. Esto simplifica el tratamiento de los objetos creados, ya que al poseer todos ellos una interfaz común, se tratan todos de la misma manera.

De comportamiento.

- Command: permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor real de la misma. Para ello se encapsula la petición como un objeto, con lo que además se facilita la programación de los métodos.

2.9 Sistema Gestor de Base de Datos.

A continuación se muestra la Tabla #3 en donde se establece una pequeña comparación entre algunos de los Gestores de BD.

Tabla # 3. Comparación entre Gestores de Base de Datos: PostgreSQL vs. MySQL vs. SQLServer

Características	**PostgreSQL**	**MySQL**	**SQLServer**
Licencia.	BSD	GPL a partir de la versión 3.23.19.	Propietaria.
Tamaño máximo de la BD.	Ilimitado	Ilimitado	524.272 terabytes
Conexiones simultáneas	32	101	32767(valor predeterminado para las conexiones de usuarios al instalarse SQLServer)
Tamaño de Query	16777216	1048574	-
Multiplataforma	Sí	Sí (es la BD Linux más popular; disponible para casi todas las plataformas hardware)	Sí

2.9.1.1 PostgreSQL.

El SGBD PostgreSQL nace en la universidad de Berkeley - California, en los años 80, como un proyecto académico y actualmente se encuentra en la versión 8.2, siendo permanentemente mantenido por la comunidad Open Source.

Este es un poderoso SGBD diseñado para administrar grandes cantidades de datos. Se ejecuta en la mayoría de los Sistemas Operativos más utilizados en el mundo incluyendo Linux, varias versiones de UNIX y por supuesto Windows. Se ha ganado la reputación de ser confiable y mantener la integridad de los datos al darle solución real a los complejos problemas del mundo empresarial y a la vez mantener la eficiencia al consultar los datos. PostgreSQL es sin dudas "el gestor de BD de código abierto más avanzado del mundo". (31)

Ofrece control de concurrencia multi-versión, soportando casi toda la sintaxis SQL (incluyendo sub-consultas, transacciones, y funciones definidas por el usuario). Soporta un amplio conjunto de enlaces con lenguajes de programación como C, C++, Java, Perl, TCL y Python.

2.9.1.2 MySQL.

MySQL es un SGBD relacionales, multi–hilo y multi-usuario. Una BD relacional almacena datos en tablas separadas en lugar de poner todos los datos en un gran almacén, lo que añade velocidad y flexibilidad. MySQL trabaja en entornos cliente/servidor o incrustados. Además, funciona en diferentes plataformas y fue escrito en C y en C++. (50)

Tiene como una de sus principales ventajas la velocidad en la lectura de datos, pero a costa de eliminar un conjunto de facilidades que presentan otros SGBD como: integridad referencial, bloqueo de registros, procedimientos almacenados.

Es muy popular en aplicaciones web y actúa como un componente de BD para diferentes plataformas trabajando sobre varias de ellas y sobre todas las versiones de Windows. Su mayor desempeño se logra cuando se combina con el lenguaje de programación PHP.

2.9.1.3 SQL Server.

SQL Server es un SGBD relacional basado en el lenguaje Transact-SQL, específicamente en Sybase IQ, capaz de poner a disposición de muchos usuarios grandes cantidades de datos de manera simultánea. Constituye la alternativa de Microsoft a otros potentes SGBD como son Oracle, Sybase ASE, PostgreSQL, Interbase, Firebird o MySQL. (32)

Incluye un paquete de herramientas para la administración de los recursos que el ordenador proporciona y los gestiona para un mejor rendimiento de la BD.

Una buena instalación y configuración de SQL Server, y sobre todo una buena administración de las herramientas que proporciona logrará:

1- Un tiempo óptimo de respuesta para la ejecución de las consultas SQL.
2- Aprovechamiento máximo de la memoria RAM y CPU de la máquina.

CPU es la unidad central de procesamiento sin la cual un equipo no puede funcionar, razón por la que muchas veces se denomina al CPU como el “cerebro” de un ordenador. Su función es realizar todos los cálculos matemáticos que se requieren para que un ordenador funcione correctamente.

2.9.2 ¿Por qué SQL Server 2000?

El tener que adecuarnos a las exigencias del cliente, fue la razón que nos condujo a escoger este SGBD.

SQL Server 2000 administra la BD y distribuye los recursos disponibles del servidor (tales como memoria, operaciones de disco, etc.) entre las múltiples peticiones. Puede proporcionar los servicios de BD necesarios para sistemas extremadamente grandes. Los servidores de gran tamaño pueden tener miles de usuarios conectados a una instancia de SQL Server 2000 al mismo tiempo. Dispone de protección total para estos entornos, con medidas de seguridad que evitan problemas como tener varios usuarios intentando actualizar los mismos datos al mismo tiempo. (51)

Da un gran soporte para estándares web, potentes herramientas para el ajuste, la gestión del sistema, escalabilidad y fiabilidad. Proporciona a los usuarios un entorno completamente integrado con XML, añade una nueva característica de servicio de análisis con apoyo de data mining y mejora la tecnología de repositorios con los servicios de metas.

Otorga a los administradores una herramienta potencialmente robusta, provista de las herramientas suficientes que le permiten mantener un óptimo nivel de seguridad en la utilización de los recursos del sistema y de la BD. Cuenta con características, que lo hace un SGBD fácil de usar y de gran aceptación, entre ellas se tienen, su gran escalabilidad y disponibilidad, así como características de BD corporativas, ya que protege la integridad de los datos y minimiza la carga de trabajo de consultas a las BD por miles de usuarios. Proporciona gran cantidad de documentación para el aprendizaje.

Todo ello permite concluir que SQL Server 2000 es un potente SGBD que aunque tiene el inconveniente de ser un software propietario, controlado por Microsoft, presenta excepcionales características que lo convierten en la mejor elección para la Empresa de Gas Manufacturado teniendo en cuenta los requisitos del negocio. No obstante, el hecho de ser propietario no es impedimento porque Symfony es el framework con el que se está desarrollando el proyecto que implementa como patrón arquitectónico el MVC2. Permite a través de la capa modelo abstraerse de la BD, es decir, que para cuando se desarrolle una segunda versión del software, se puede cambiar a otro SGBD de tipo libre, sin necesidad de cambiar nada en el código.

2.10 Entorno de Desarrollo Integrado.

Un Entorno de Desarrollo Integrado (Integrated Development Environment, IDE) es un entorno de programación que ha sido empaquetado como un programa de aplicación. Consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). (38)

Un IDE es más bien un programa que te ayuda a hacer programas y que a la vez está compuesto por un conjunto de herramientas para un programador, pudiendo dedicarse a un solo lenguaje de programación o incluso pueden integrar varios lenguajes. A

parte de chequear tu sintaxis, los IDE ejecutan su código escrito en una máquina virtual o servidores que se le hayan instalado. Pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse, al que mediante plugins se le puede añadir soporte de lenguajes adicionales. (38)

Eclipse es un IDE de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. (37)

La base para Eclipse es la plataforma de cliente enriquecido (del Inglés Rich Client Platform RCP). Los siguientes componentes constituyen la plataforma de cliente enriquecido: (37)

- Plataforma principal - inicio de Eclipse, ejecución de plugins.
- OSGi - una plataforma para bundling estándar.
- El Standard Widget Toolkit (SWT) - Un widget toolkit portable.
- JFace - manejo de archivos, manejo de texto, editores de texto.
- El Workbench de Eclipse - vistas, editores, perspectivas, asistentes.

2.10.1 ZendStudio6.0.0 para Eclipse.

Este es el IDE que se usa para el desarrollo del proyecto. El mismo abarca todas las capacidades de edición de códigos necesarios para desarrollar modernas aplicaciones empresariales y que está disponible para desarrolladores profesionales. Incluye un poderoso editor de código con soporte para todos los formatos web, Zend Marco de Integración, servicios Web de apoyo y acceso a cientos de plantillas predefinidas. Todas estas características combinadas le proporcionan a los desarrolladores simplicidad y productividad.

2.11 Conclusiones.

En este capítulo se caracterizaron, analizaron y definieron las principales herramientas y tecnologías para darle solución al sistema que se desea desarrollar, teniendo presente para esta selección que dichas herramientas debían ser libres y fáciles de emplear para el desarrollo de aplicaciones Web. En un caso determinado, se seleccionó una herramienta propietaria debido al cliente que así lo exigía.

Capítulo 3: Línea Base de la Arquitectura.

3.1 Introducción.

En este capítulo se define la línea base de la arquitectura. Se tocan diferentes puntos de interés dentro de la arquitectura y uno de ellos es el desarrollo de las 4+1 Vistas de la Arquitectura propuestas por RUP. Además se describe la arquitectura del negocio, la estructura del equipo de desarrollo, la configuración de los puestos de trabajo y se representa mediante un esquema el organigrama de la arquitectura.

3.2 Arquitectura del Negocio.

Durante el desarrollo de un software, aparecen tres arquitecturas, siendo la Arquitectura del negocio una de ella. Esta arquitectura en específico, describe los aspectos importantes de la organización, es decir, incluye a un nivel alto de abstracción la descripción de los procesos y estructuras básicas de la organización, siendo su principal objetivo marcar la base sobre el negocio y las necesidades que de él se generan. Las otras dos son conocidas como la Arquitectura de Software que según los datos que le provee la Arquitectura del negocio define las aplicaciones que darán soporte al negocio y la Arquitectura técnica, que se ve implícita dentro de la Arquitectura de Software y representa los elementos sobre los que se instalará y ejecutará la aplicación, describe así el nivel hardware del sistema.

Este apartado tiene como objetivo dar a conocer la Arquitectura del negocio de la Empresa de Gas Manufacturado y partiendo de ahí dar paso a la nueva arquitectura que dará soporte al sistema que actualmente se está desarrollando para dicha empresa.

La Empresa de Gas Manufacturado actualmente cuenta con un sistema (SISTMET) desarrollado en Visual Basic 6 que no permite ajustarse a la implantación de nuevas tarifas por rango de consumo, sólo da la posibilidad de :

1- Introducir nuevos contratos (nuevos consumidores).

2- Efectuar modificaciones de cualquier tipo a los clientes.

3- La captación de datos por medio manual o electrónico (TPL), tanto de lecturas de los metros contadores como de los cobros efectuados a los clientes.

4- Efectuar ajustes a la facturación de cada uno de los usuarios.

5- Facturar de forma personalizada a los clientes.

6- Emitir los recibos de cobro del gas manufacturado.

7- Visualización e impresión de reportes sobre el proceso de facturación y cobro.

8- Tecleo, procesamiento y cuadre del proceso de cobros recaudados por la UEB Comercial.

Todo este proceso de facturación y cobro se efectúa teniendo en cuenta las dos tarifas vigentes actualmente para el cobro del gas manufacturado a 0.11 centavos/M3 para el cliente metrado y 0.04 centavos/M3 para el cliente no metrado.

Lo que se necesita es un sistema que permita:

1- En primer lugar mantener todas las posibilidades enumeradas anteriormente, lo cual le daría flexibilidad para introducir y captar datos.

2- La facturación dinámica, ya sea mensual, bimestral o trimestral a cada cliente del gas manufacturado, tanto del sector privado como estatal.

3- El pago por adelantado del servicio de gas.

4- La creación y emisión de los recibos de cobro de gas manufacturado en formato de factura de cobro.

5- La facturación a los clientes por tarifas movibles o dinámicas, enmarcadas en rangos de consumo, como se prevé sea establecido el cobro del servicio de gas tanto para el sector privado como estatal.

6- El enlace Casas Comerciales-UEB Comercial.

7- La realización de cuadres parciales, que permitan saber en cada momento el estado de gestión de la actividad comercial.

8- Un registro histórico detallado de los movimientos que se le realizan al cliente.

9- Efectuar el cobro de gas a través de correo y banco.

10- Incluir dentro del proceso de facturación el cobro de servicios post venta al cliente por ej. variaciones en la instalación del servicio de gas.

11- Efectuar pago desde cualquier Casa Comercial.

Como solución ha dicho problema la base arquitectónica del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado se ha modelado haciendo uso del Patrón Arquitectónico Modelo Vista Controlador a través de una Aplicación Web, desarrollada en PHP 5.0 con tecnología orientada a objetos, haciendo uso del framework Symfony, con UML como lenguaje de modelado y Visual Paradigm Enterprise Edition como herramienta CASE, bajo la metodología RUP, con SQL Server como Gestor de BD y ZendStudio6.0.0 para Eclipse como entorno de desarrollo.

3.2.1 ¿Por qué una Aplicación Web?

Dada la estructura que presenta la Empresa de Gas Manufacturado se hizo necesario el desarrollo de una aplicación Web. Se tuvo en cuenta para ello que la información debía ser accesible desde cualquier lugar dentro de la UEB Comercial y también desde el exterior con las demás Casas Comerciales; o sea esta información debía ser compartida entre todas las partes interesadas, de manera que todas tengan acceso a la información completa (o a aquella parte que les corresponda según su función) en cada momento que se requiera.

A continuación se mencionarán algunas ventajas de usar estas aplicaciones online.

Ventajas: (46)

- Proporcionan movilidad, dado que puedes ejecutarlas desde cualquier ordenador con conexión a internet.
- La información que manejan se accede a través de internet, motivo por el cual son especialmente interesantes para desarrollar aplicaciones multiusuario basadas en la compartición de información.
- El cliente o usuario que utiliza la aplicación no necesita tener un ordenador de grandes prestaciones para trabajar con ella.

- No hay necesidad de actualizaciones.

3.2.2 Restricciones Arquitectónicas.

El Sistema de Facturación y Cobro debe interactuar con un sistema implantado actualmente en la Empresa de Gas Manufacturado que tiene estrecha relación con los procesos de negocio. Se debe mencionar el SISTMET, cuya BD almacena la información de las facturas, lecturas y la información de los contratos. El nuevo sistema se nutrirá de la información que se almacena en la BD de SISTMET.

Otras fuentes de información utilizadas por procesos de negocio son documentos en formato Microsoft Excel, por lo que el sistema propuesto debe de trabajar con este estándar tanto para lectura como para escritura. Se exportará a formato pdf y Excel utilizando librerías existentes de PHP.

Los datos del sistema se manejan a través del protocolo HTTP, el cifrado de datos se gestionará a nivel del servidor web utilizando SSL.

El sistema está diseñado para que desde las Casas Comerciales que estarán conectadas al centro de cálculo se puedan realizar todas las operaciones diarias del mismo. El tecleo de los talones prácticamente se hará todo mediante el uso de los TPL, lo cual reducirá bastante el tecleo manual al igual que las lecturas del mes.

El servidor Web y la BD del sistema se encontrarán en la Casa Matriz Comercial de la Empresa de Gas Manufacturad (UEB Comercial). Teniendo en cuenta que las fuentes de datos son las Casas Comerciales y éstas se encuentran en diferentes municipios de la capital cubana, se puede suponer que la distribución es bastante amplia.

Actualmente existen restricciones para el uso del propio sistema. Una vez que se implante puede no tener conectividad con todas o un grupo de Casas Comerciales, que están distribuidas por diferentes municipios de la capital lo cual obliga a que el sistema incluya las funcionalidades que garantice el trabajo sin conexión.

Estas funcionalidades son:

Cuando la conectividad entre la UEB Comercial y una Casa Comercial sea nula.

- Lectura de ficheros extensiones (**.dbf**, **.xml**, **.xls**).
- Carga y descarga de información al sistema.

- Generar ficheros extensiones (**.pdf**, **.xml**, **.xls**, **.dbf**).

Cuando la conectividad entre la UEB Comercial y una Casa Comercial sea limitada.

El sistema deberá actualizar los cambios en dos momentos del día.

- Antes que empiece la jornada laboral: consiste en actualizar todos los cambios desde la Casa Matriz Comercial hacia las Casas Comerciales.
- Después que culmine la jornada laboral: consiste en actualizar todos los cambios desde las Casas Comerciales hacia la Casa Matriz Comercial.

Cuando la conectividad entre la UEB Comercial y una Casa Comercial sea permanente.

- El sistema trabaja directamente con el servidor central de la Casa Matriz Comercial, actualizando la información de cada Casa Comercial.
- El sistema brinda la opción de actualizar la información a cada Casa Comercial, previendo que la BD del sistema cliente no está actualizada.

No existen otras restricciones que puedan atentar contra el funcionamiento del sistema y el cumplimiento de los objetivos.

3.3 Estructura del Equipo de Desarrollo.

Todo equipo de desarrollo acostumbra a definir una estructura para un mejor funcionamiento y desarrollo de su proyecto. En ésta estructura de tipo formal se ve plasmada la interacción de los integrantes, los diferentes roles que asumen; así como la distribución de los puestos de trabajo por roles; el líder elegido o nombrado y la forma en que se debe dar la toma de decisiones y la distribución de tareas. Así mismo los equipos tienden a desarrollar una estructura informal, dígase estructura informal a la forma en que se da la comunicación y la interacción entre los integrantes de cada equipo.

3.3.1 Herramientas.

A continuación se mencionarán las herramientas tanto de modelación como de desarrollo que se definieron en el proyecto de Manugas.

Herramientas de Modelado.

- Lenguaje de Modelado: UML 1.0
- Herramienta CASE: Visual Paradigm Enterprise Edition.

Herramientas de Desarrollo.

- Lenguajes de Programación: PHP 5.0
- IDE: ZendStudio6.0.0 for Eclipse.
- Gestor de Base de Datos: SQL Server 2000.
- Control de Versiones: Subversion con cliente TortoiseSVN.

3.3.2 Ambiente de Desarrollo.

Para el desarrollo del proyecto se cuenta con 7 PC. En una de ellas se encuentra el Servidor de BD, en el que están almacenados toda la información referente al sistema y en el cual están instalados los mismos programas de las PC clientes y además el Gestor SQL Server 2000. (Ver Tabla #4)

Tabla # 4. Descripción del Servidor de Base de Datos

Servidor de Base Datos	
Descripción	Pentium 4. 3.0 GHz 512MB HDD: 120GB
Software Base	Windows Server 2003
Servicios	
Descripción	**Observaciones**
Zend Studio for Eclipse	IDE
Visual Paradigm Suite v3.1	Herramienta CASE
MS Office 2007	Documentación de Ingeniería
Subversion, (cliente Tortoise 1.4.5)	Cliente SVN para Windows
Internet Explorer	Navegador Windows

PostgreSQL	Para docencia
PHP v5	Lenguaje de Programación
Symfony para PHP v1.0.17	Framework
MS SQL Server	Servidor de Bases de Datos

Las otras 6 PC son clientes (Ver Tabla #5) en las cuales están instalado los programas ZendStudio6.0.0 for Eclipse, Windows XP Service Pack 2, Subversion con cliente TortoiseSVN para el control de versiones, Visual Paradigm Suite v3.1, Symfony para PHP v1.0.17 y además del gestor de BD con el que se está desarrollando la aplicación: SQL Server 2000 se cuenta con PostgreSQL.

Tabla # 5. Descripción de las PC Clientes

PC cliente	
Descripción	Pentium 4. 3.0 GHz 512MB HDD: 120GB
Software Base	Windows XP Service Pack 2
Servicios	
Descripción	**Observaciones**
Zend Studio for Eclipse	IDE
Visual Paradigm Suite v3.1	Herramienta CASE
MS Office 2007	Documentación de Ingeniería
Subversion, *(cliente Tortoise 1.4.5)*	Cliente SVN para Windows
Internet Explorer	Navegador Windows
PHP v5	Lenguaje de Programación
Symfony para PHP v1.0.17	Framework

Apache	Servidor Web

En general el ambiente de desarrollo gira en torno al servidor de control de versiones (Ver Tabla #6) donde está almacenada toda la información del proyecto de forma general. A él tienen acceso todos los integrantes del proyecto con la posibilidad de subir y descargar información con la última actualización. En dicho servidor, al cual se accede mediante la herramienta de control de versiones *(*TortoiseSVN*)* también se encuentra la última versión del cronograma de trabajo, y el fichero .vpp que contiene todos los artefactos modelados en el desarrollo actual.

Tabla # 6. . Descripción del Servidor Local

Servidor Local	
Descripción	Pentium 4. 3.0 GHz 512MB HDD: 120GB
Software Base	Ubuntu 8.04
Servicios	
Descripción	**Observaciones**
Apache	Servidor Web
PostgreSQL	Servidor de Base de datos para las aplicaciones del polo
DotProject	Para la gestión de Proyecto
PostgreSQL	Servidor de Base de datos para las aplicaciones del polo
DotProject	Para la gestión de Proyecto

3.3.3 Estructura del Equipo de Trabajo.

La Figura #3 muestra en forma de árbol jerárquico la estructura del equipo de desarrollo de Manugas. Se definen para ello los roles y responsables por cada rol.

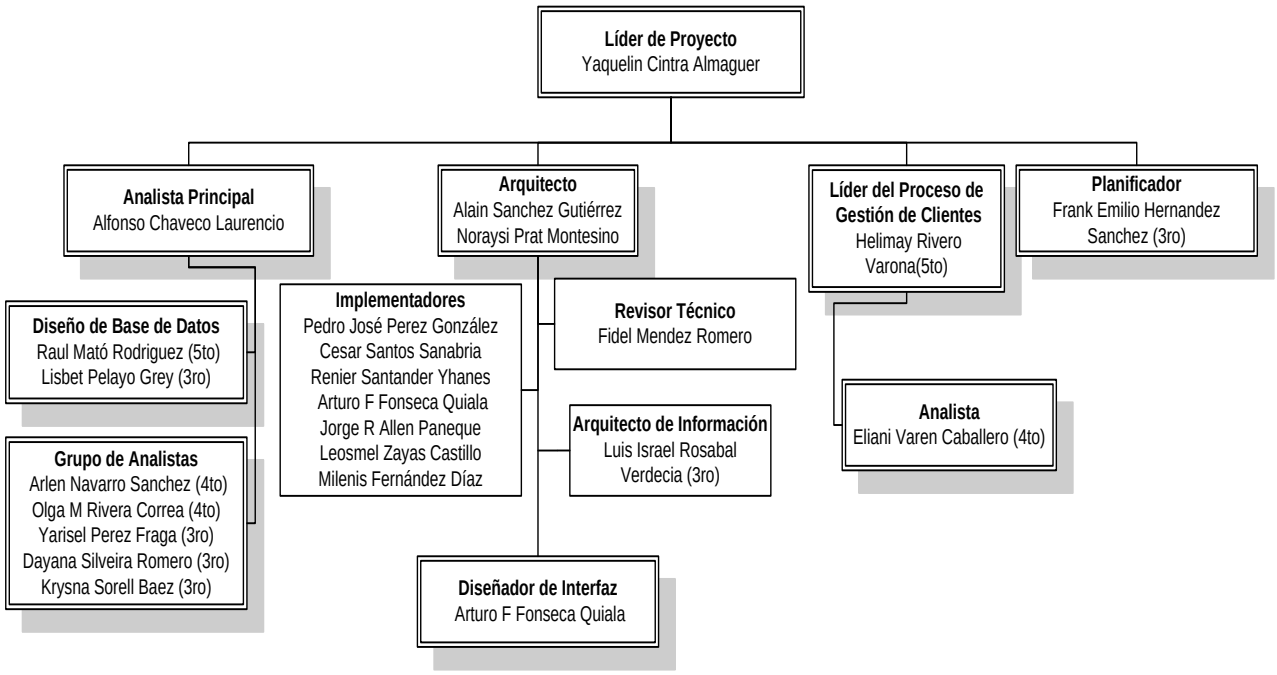


Figura # 3. Estructura del Equipo de Trabajo

3.3.4 Configuración de los Puestos de Trabajos por Roles.

- Arquitecto.
    1. PC, con mouse y teclado.
    2. Visual Paradigm.
    3. Instalación del paquete Office (Incluido el Objeto Visio).
    4. Dreamweaver 8, Apache, Symfony con PHP5.0.
    5. Subversion con cliente TortoiseSVN.

- Analista.
    1. PC, con mouse y teclado.
    2. Visual Paradigm.
    3. Instalación del paquete Office.
    4. Subversion con cliente TortoiseSVN.

- Implementador.
    1. PC, con mouse y teclado
    2. Instalación del paquete Office.
    3. Instalación del IDE ZendStudio6.0.0 for Eclipse.

4. Dreamweaver 8, Apache, Symfony con PHP5.0.
5. Subversion con cliente TortoiseSVN.

- Diseñador de BD.
    1. PC, con mouse y teclado.
    2. Instalación del paquete Office.
    3. SQL Server 2000.
    4. Manager SQL Server 2008 for SQL Server.
    5. Embarcadero ERStudios 6.0.
    6. Subversion con cliente TortoiseSVN.

3.4 Organigrama de la Arquitectura.

Un organigrama es una representación gráfica de la estructura organizativa de una empresa u organización, es decir, representa un modelo abstracto y sistemático, que permite obtener una idea uniforme acerca de la estructura formal de una organización.

El organigrama de la arquitectura refleja en sí, los aspectos internos de cómo está organizado y estructurado el sistema que se quiere desarrollar.

Para el caso específico del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado, se representara la estructura del patrón arquitectónico MVC que propone Symfony y se explica luego el funcionamiento de cada uno de los componentes en que se divide dicho patrón.

Visión general de la arquitectura – Alineamiento de paquetes, módulos y Capas.

Symfony toma lo mejor de la arquitectura MVC y la implementa de forma que el desarrollo de aplicaciones sea rápido y sencillo, además de generar la estructura de los proyectos, así como las clases abstracción a la BD, controlador, layouts, evitando que el desarrollador tenga que escribir código, posibilitando un menor tiempo y una alta calidad y legibilidad de las estructura del código fuente del sistema.

A continuación se muestra en la Figura #4 el organigrama de la arquitectura según Symfony para el patrón MVC.

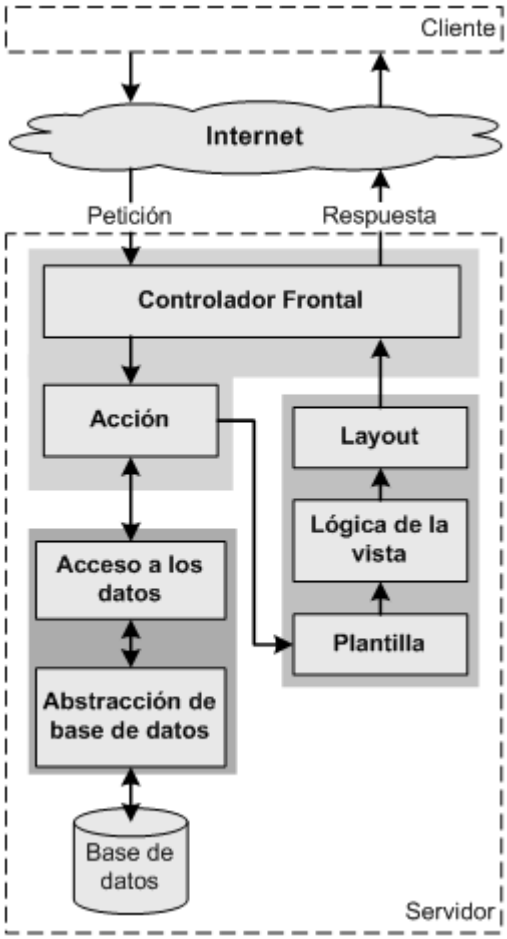


Figura # 4. Organigrama de la Arquitectura

¿Cómo Symfony organiza el trabajo en cada una de las capas?

- **La Modelo.**

La capa del **modelo** es donde se define la lógica de negocio. En esta capa es donde *Symfony* guarda todas las clases y archivos relacionados con el modelo en el directorio lib/model/. (41)

Como se puede apreciar en la Figura #4 la modelo se divide en dos capas: la capa de acceso a datos y la capa de abstracción de BD.

La capa de acceso a datos representa la lógica de negocio de la aplicación, donde se implementan las reglas del negocio y los requerimientos funcionales que debe cumplir el sistema y es independiente de la BD utilizada, lo que da la posibilidad de cambiar de SGBD con solo actualizar la capa de abstracción de la BD.

La capa de abstracción de BD permite evitar que los programadores de aplicaciones web, tengan que generar sentencia SQL, fomenta la reutilización de código, permite realizar cambios de gestores de BD, sin realizar cambios en la lógica de negocio. Para ello se auxilia de la librería de Propel (Implementa el Patrón Objet Relational Mapping) y la librería de Creole, esta última encargada de abstraer el sistema del gestor de BD que se use, facilitado así que el sistema pueda usar varios servidores de BD o soporte la migración de un gestor específico a otro sin modificar código alguno en la aplicación.

- **La vista.**

La **vista** que es lo que utilizan los usuarios para interactuar con la aplicación. En *Symfony* la capa de la vista está formada principalmente por plantillas en PHP. Estas plantillas se guardan en varios directorios llamados templates/ repartidos por todo el proyecto. (41)

La Figura #4 muestra como la vista se separa en el layout y la plantilla.

El layout es un archivo de código interpretado por cualquier navegador web, HTML, CSS, Java Script, común para el uso de toda la aplicación, es decir que es global en toda la aplicación o al menos en un grupo de páginas.

La plantilla es la presentación de los datos de la acción que se está ejecutando. Se encarga de visualizar las variables definidas en el controlador.

- **El controlador.**

El **controlador** representa un bloque de código que realiza llamadas al modelo para obtener los datos y se los pasa a la vista para que los muestre al usuario. O sea, es el intermediario entre la vista y el modelo. En esta capa *Symfony*, realiza las peticiones que se canalizan a través de los controladores frontales (index.php y frontend_dev.php). Estos controladores frontales realmente delegan todo el trabajo en las acciones. (41)

El controlador es un componente que sólo tiene código relativo al MVC, por lo que no es necesario crear uno, ya que Symfony lo genera de forma automática. Entre sus tareas más comunes se encuentran el manejo de las peticiones del usuario, el manejo de la seguridad, cargar la configuración de la aplicación y otras tareas similares.

El controlador se divide, como se muestra en la Figura #4 en un controlador frontal y las acciones.

El controlador frontal es único para cada aplicación. Una de las principales ventajas de utilizar un controlador frontal es que ofrece un punto de entrada único para toda la aplicación. Así, en caso de que sea necesario impedir el acceso a la aplicación, solamente es necesario editar el script correspondiente al controlador frontal. Si la aplicación no dispone de controlador frontal, se debería modificar cada uno de los controladores.

La acción incluye el código específico del controlador de cada página. Representa un requerimiento en específico que el sistema debe realizar, para brindar un determinado resultado a una petición del navegador (Cliente). Pertenece al nivel de control.

El controlador frontal y el layout son comunes para todas las acciones de la aplicación. Se pueden tener varios controladores y varios layouts, pero solamente es obligatorio tener uno de cada aplicación.

3.5 Descripción de la Arquitectura.

Para una mejor comprensión del diseño, es necesario ver el sistema desde diferentes perspectivas, es por eso que la arquitectura se representa mediante varias vistas que se centran en aspectos concretos del sistema, abstrayéndose de los demás. Para RUP, metodología seleccionada para el desarrollo del proyecto, todas las vistas juntas forman el llamado modelo 4+1, siendo este modelo el más aceptado a la hora de establecer las vistas necesarias para describir una arquitectura. Como se mencionaba en el capítulo anterior, RUP define cuatro vistas principales: vista lógica, vista de proceso, vista de despliegue, vista de implementación y una vista rectora que es la vista de casos de uso.

3.5.1 Vista de Casos de Uso.

La vista de casos de uso representa el comportamiento del sistema tal y como es percibido por usuarios, analistas y encargados de pruebas. A través de la misma se muestra un subconjunto del artefacto del modelo de caso de uso y la lista de los casos de uso o escenarios del modelo más significativos con las funcionalidades centrales del sistema.

En el proyecto se identificaron 48 requerimientos, que fueron agrupados en 14 casos de uso del sistema y están distribuidos por módulos como se muestra en la Tabla #7.

Tabla # 7. Casos de Uso por Módulo

Módulo de Facturación.	**Módulo de Cobro.**
Cargar TPL.	Registrar Cobro en las CC.
Leer TPL.	Generar Recibo de Cobro.
Cargar Datos a Facturar.	Buscar cliente.
Realizar Facturación.	Comprobar las cuentas cobradas y por cobrar.
Corregir errores.	Generar Informe Diario por cobrador.
Exportar Datos de Lectura	Generar DC.
	Generar Modelo Liquidación en efectivo por cobrador.
	Imprimir cobros.

Los casos de uso según el impacto que tienen en la arquitectura se pueden clasificar en críticos, secundarios, auxiliares y opcionales.

Los casos de uso críticos son los más importantes para el usuario porque cubren las principales tareas o funciones que el sistema ha de realizar. Definen la arquitectura básica. Por su parte los casos de uso secundarios sirven de apoyo a los casos de uso críticos, involucran funciones secundarias y tienen un impacto más modesto sobre la arquitectura, pero deben implementarse pronto porque responden a requerimientos de interés para los usuarios. Los auxiliares no son claves para la arquitectura y completan casos de uso crítico o secundario. Por último los opcionales responden a funcionalidades que pueden o no estar en la aplicación, pero que no son imprescindibles en las primeras versiones. (5)

Para el proyecto Manugas de los 14 casos de usos del sistema identificados, 5 fueron clasificados como casos de uso críticos y los restantes como casos de uso secundarios. (Ver Tabla #8)

Tabla # 8. Clasificación de los Casos de Uso

Casos de Uso Críticos.	**Casos de Uso Secundarios**.
Cargar TPL.	Exportar Datos de Lectura
Leer TPL.	Registrar Cobro en las CC.
Cargar Datos a Facturar.	Generar Recibo de Cobro.
Realizar Facturación.	Buscar cliente.
Corregir errores.	Comprobar las cuentas cobradas y por cobrar.
	Generar Informe Diario por cobrador.
	Generar DC.
	Generar Modelo Liquidación en efectivo por cobrador.
	Imprimir cobros.

Casos de Uso Arquitectónicamente significativos.

Los casos de uso arquitectónicamente significativos son aquellos que representan las partes más críticas de la arquitectura del sistema y demuestran la funcionalidad de este, además de validar la arquitectura propuesta para el mismo. (5)

A partir de aquí, es decir, después que el analista del sistema encuentra los actores y casos de uso y los clasifica, es función del arquitecto priorizar dichos casos de uso, con el objetivo de determinar cuáles son necesarios para el desarrollo en las primeras iteraciones y cuáles pueden dejarse para posteriores iteraciones. Los casos de uso que se tienen en cuenta para ello son los casos de uso críticos para el sistema pues son estos los que definen la arquitectura básica.

Algunos de los criterios a tener en cuenta para priorizar un caso de uso son: casos de uso con dificultad en desarrollo, casos de uso imprescindibles para la puesta en marcha del sistema, organización del desarrollo incremental, disponibilidad del equipo de desarrollo.

A partir de este análisis, se ha elaborado una tabla que refleja dichos aspectos para cada uno de los casos de uso críticos (Ver Tabla #9).

Tabla # 9. Priorización de los Casos de Uso

CUS/ Aspectos	Casos de uso con dificultad en desarrollo.	Casos de uso imprescindibles para la puesta en marcha del sistema.	Organización del desarrollo incremental.	Disponibilidad del equipo de desarrollo.
Cargar TPL.	x	x	x	x
Leer TPL.	x	x	x	x
Cargar Datos a Facturar.	x	x	x	x
Corregir errores.		x	x	x
Realizar Facturación.	x	x	x	x

Teniendo en cuenta el criterio de selección que se definió en la tabla anterior se llevó a una escala de hasta 4 puntos. Cada cruz es 1 punto. Los casos de uso que oscilen entre 3 y 4 puntos entonces serán considerados como arquitectónicamente significativos. Se hace necesario este análisis porque con certeza podemos asegurar que todos los casos de uso identificados como arquitectónicamente significativos son en su concepto casos de uso críticos, pero no todos los casos de uso críticos que se identifiquen en el sistema pueden decirse que son arquitectónicamente significativos antes de someterlos a este proceso de selección.

Además de lo ya mencionado, para Manugas los casos de uso significativos han sido priorizados en base al soporte que brindan a las metas del negocio. La Figura #5 muestra la vista de la arquitectura del modelo de casos de uso para Manugas.

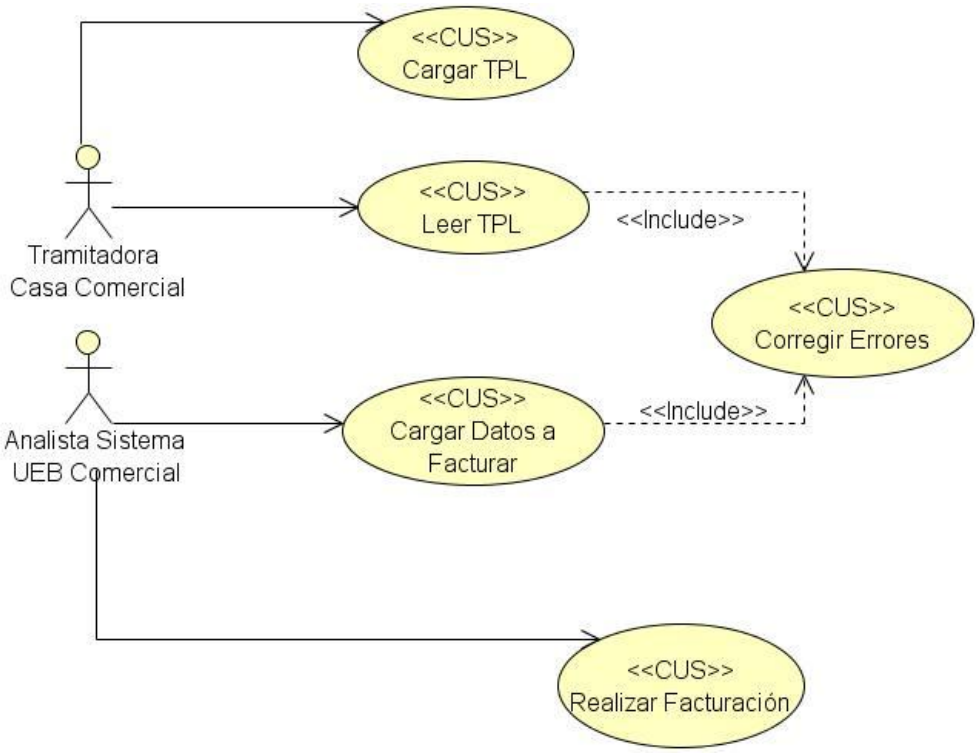


Figura # 5. Vista de la Arquitectura del modelo de casos de uso para Manugas

A continuación se muestra la vista general de casos de uso por módulos y la relación entre ellos. (Ver Figura #6)

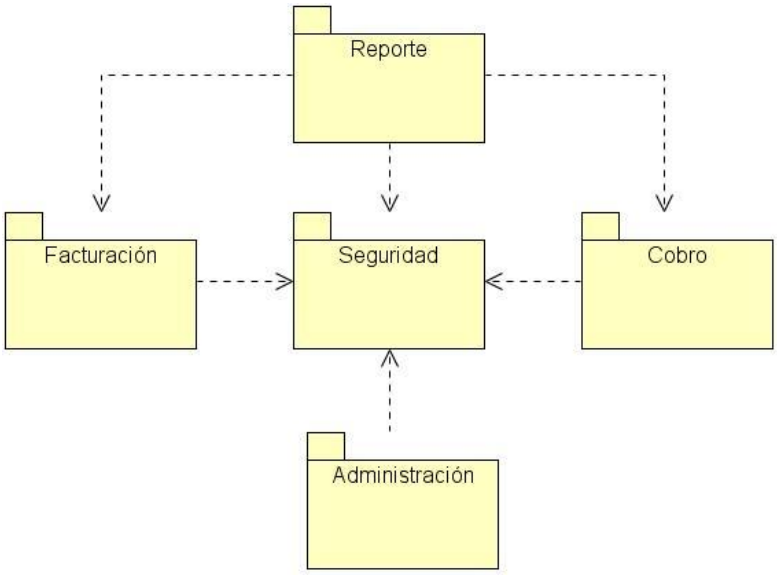


Figura # 6. Vista General de Casos de Uso por Módulos

Módulo de Seguridad: Consiste en que el sistema debe brindarle al cliente la mayor seguridad posible. En dependencia del cargo que ocupe en la empresa son los

permisos que puede tener en el sistema, ya sea el Analista del Sistema, la cajera u otro tipo de persona que tenga relación con el mismo.

Módulo de Reporte: Consiste en generar todos los reportes que se van a generar en la empresa.

Módulo de Administración: Consiste en gestionar usuarios en el sistema, darles los permisos de administración.

Módulo de Facturación: Consiste en lograr que el nuevo sistema emita las facturas de una manera más organizada, segura y cómoda posible para los usuarios del sistema en cuestión.

Módulo de Cobro: Consiste en que dada la factura emitida por el sistema de manera escalonada.

3.5.2 Vista de Procesos.

La vista de procesos representa a los hilos y procesos que forman mecanismos de sincronización y concurrencia en el sistema. Se hace mayor énfasis en las clases activas. La vista de procesos es útil para realizar análisis de integridad y tomar decisiones de integración con otros sistemas.

El sistema de Facturación y Cobro que se está desarrollando actualmente para la Empresa de Gas Manufacturado se basa en la arquitectura MVC sobre la plataforma Web Apache, donde se propone la separación del sistema en distintos componentes de la interfaz de usuario (vistas), el modelo de negocio y la lógica de control. La concurrencia de utilización del servidor Web y la utilización de la BD se maneja a través de los propios servidores.

El sistema no cuenta por tanto con tareas que requieran ejecutarse periódicamente sin la intervención del cliente. Por lo anterior no se requiere de una Vista de Procesos.

3.5.3 Vista Lógica.

En la descripción de la arquitectura, la vista lógica describe las clases más importantes que formarán parte del ciclo de desarrollo. Se describe los paquetes más abstractos del sistema y las relaciones que entre ellos existen ya sea de dependencia o de uso.

En la Figura #7 se muestran los elementos del modelo arquitectónicamente significantes para el proyecto. De esta forma queda representada la vista lógica.

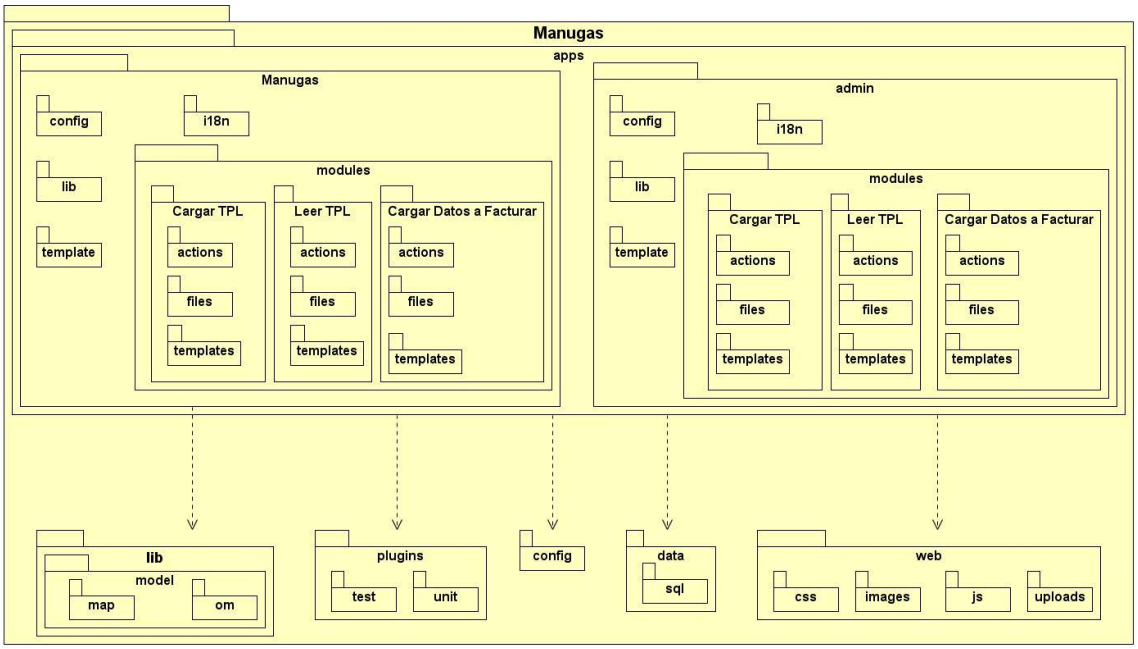


Figura # 7. Vista Lógica

A continuación se describen algunas de las carpetas más importantes y las funcionalidades que recogen.

- **Carpeta apps/:** contiene un directorio por cada aplicación del proyecto. Puede contener una o varias aplicaciones. En este caso se cuenta con dos aplicaciones.

  Manugas: es la parte que tiene que ver con implementación de los procesos de negocio de la empresa, o sea para la parte pública.

  Admin: es la parte administrativa de la aplicación, o sea para la parte de gestión.

Cada aplicación contiene uno o más módulos. Cada módulo tiene su propio subdirectorio dentro del directorio modules y el nombre del directorio es el que se elige durante la creación del módulo.

- **Carpeta modules/:** almacena los módulos que definen las características de la aplicación. Cada módulo representa un caso de uso de sistema.

Cada módulo tiene su actions y sus templates que representan a la clase controladora y a las vistas respectivamente.

actions/: normalmente contiene un único archivo llamado (actions.class.php) y que corresponde a la clase que almacena todas las acciones del módulo. También es posible crear un archivo diferente para cada acción del módulo.

templates/: contiene las plantillas correspondientes a las acciones del módulo. Cuando se crea un nuevo módulo, automáticamente se crea la plantilla llamada (indexSuccess.php).

- **Carpeta lib/:** almacena las clases y librerías externas. Se suele guardar todo el código común a todas las aplicaciones del proyecto.

La model/ que contiene esta carpeta es el subdirectorio que guarda el modelo de objetos del proyecto, donde está implementada toda la lógica de negocio del sistema.

- **Carpeta data/:** en este directorio se almacenan los archivos relacionados con los datos, como por ej. El esquema de una BD, el archivo que contiene las instrucciones SQL para crear las tablas e incluso un archivo de BD de SQLite.
- **Carpeta plugins/:** almacena los plugins instalados en la aplicación.
- **Carpeta web/:** la raíz del servidor web. Los únicos archivos accesibles desde Internet son los que se encuentran en este directorio.

3.5.4 Vista de Implementación.

Esta vista describe la descomposición del software en capas y subsistemas de implementación. Provee una vista de la trazabilidad de los elementos de diseño de la vista lógica ahora para la implementación.

A continuación se muestra la vista de implementación para el proyecto de Manugas. (Ver Figura #8)

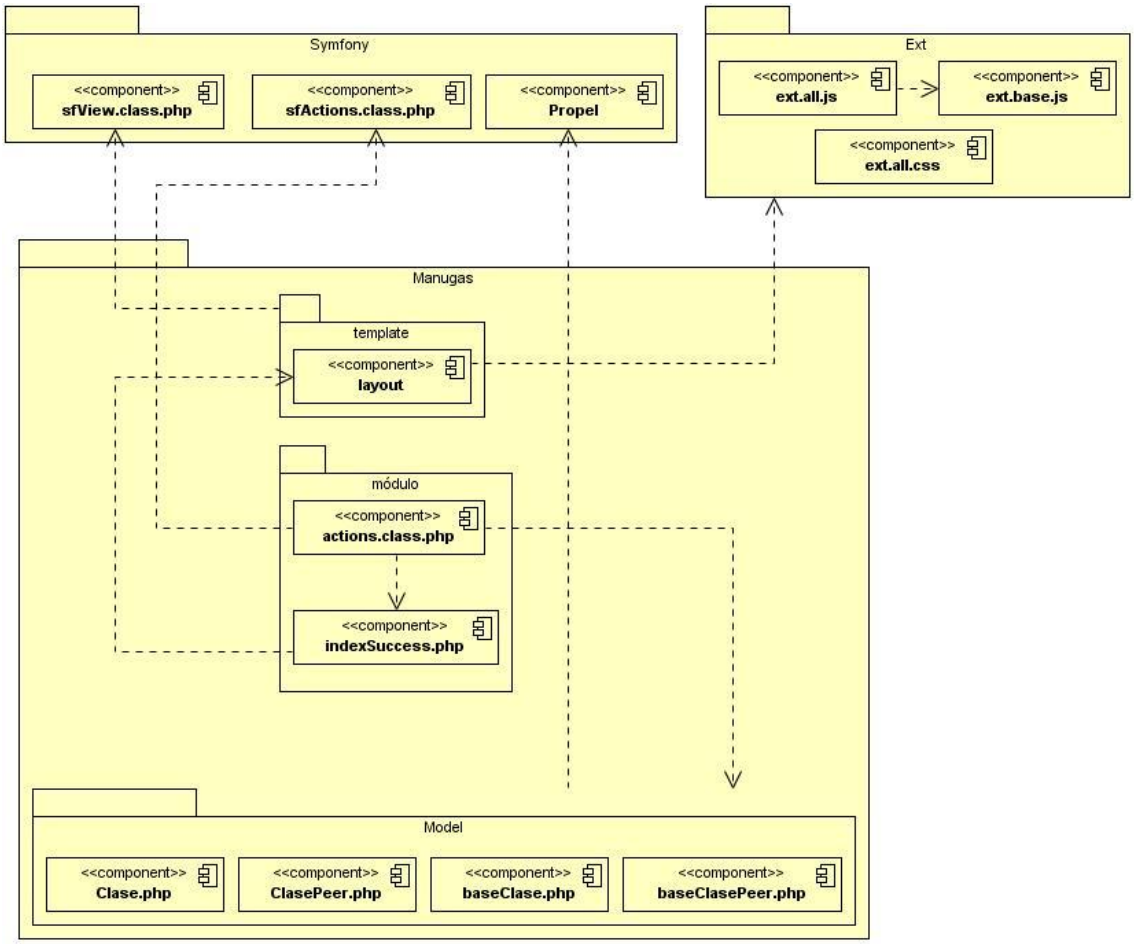


Figura # 8. Vista de Implementación

Describiendo la figura que anteriormente se mostraba se tiene una carpeta raíz: Manugas que contiene toda la información del proyecto. Dentro de ella está la carpeta Template, Módulo y Model.

- **Carpeta Template.**

  En esta carpeta es donde se encuentra el componente layout.php que es la plantilla que se le aplica a todas las páginas. La misma hace uso de los ficheros definidos dentro de la carpeta Ext. JS para conformar la vista y de la componente sfView.class.php para mostrar dicha vista.

- **Carpeta Módulo.**

  Está formada por un componente actions.class.php (que representa al controlador) y un componente indexSuccess.php (que representa a la vista). El controlador se relaciona con la carpeta Model para acceder a los datos

correspondientes de la BD y hace uso además del componente sfAction.class.php que trae Symfony por defecto. La vista por su parte hace uso del componente layout.php que está en la carpeta Template para darle forma a la página.

- **Carpeta Model.**

  La carpeta Model contiene las clases encargadas del manejo de información con las tablas de la BD. Por cada tabla Symfony crea cuatro clases en la BD. Dicha carpeta utiliza el componente Propel para la abstracción a la BD.

  Para el desarrollo del proyecto se consideró que cada caso de uso representaría un módulo. Hasta el momento hay implementados 3 módulos en el proyecto: CargarTPL, LeerTPL, CorregirErrores y se está trabajando sobre un cuarto módulo que es CargarDatosFacturar.

Además se tiene la **carpeta Symfony** que contiene las librerías a usar para el desarrollo de la aplicación teniendo en cuenta el patrón MVC: sfView.class.php para mostrar las vistas, sfAction.class.php para la controladora y Propel para el acceso a la modelo.

Un **paquete Ext. JS** que es una librería Java Script ligera y de alto rendimiento, compatible con la mayoría de los navegadores que permite crear páginas e interfaces web dinámicas usada para trabajar la parte de la presentación de las vistas.

3.5.5 Vista de Despliegue.

La vista de despliegue de un sistema contiene los nodos que forman la topología hardware sobre la que se ejecuta el sistema, la distribución, entrega e instalación de las partes que constituyen el sistema físico. (48)

Con otras palabras la vista de despliegue propone la distribución física del sistema y como estarán distribuidos los componentes de la aplicación en ellos. Cabe destacar que existe una traza directa del modelo de implementación, puesto que cada componente físico debe estar almacenado en un nodo, esto incluye también la asignación de tareas provenientes de la vista de procesos en los nodos.

Diagrama de despliegue.

Los diagramas de despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. (49)

Para realizar el despliegue de la aplicación se requiere tanto en la UEB Comercial como en las Casas Comerciales de un Servidor Web donde se pondrá en marcha el producto Manugas; un Servidor de BD que almacena toda la información que genera el sistema referente a las lecturas y cobros de gas manufacturado a clientes; y varias PC Clientes que deben estar equipadas adicionalmente por un Dispositivo Portátil de Lectura (TPL) y una Impresora.

La figura que se muestra a continuación representa el diagrama de despliegue que se propone para la Empresa de Gas Manufacturado. (Ver Figura #9)

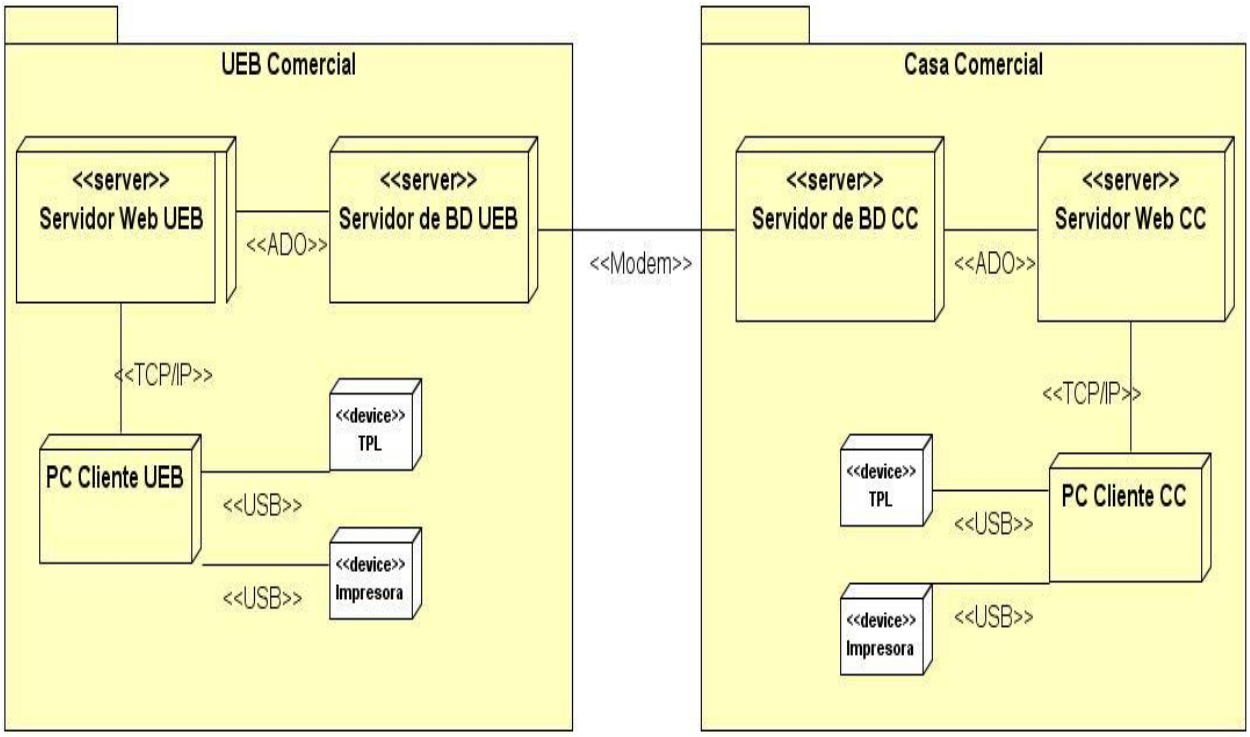


Figura # 9. Diagrama de Despliegue

Descripción de los nodos físicos.

Se propone para el despliegue de la aplicación tener en cuenta los Requerimientos de Hardware que a continuación se describen:

Mínimo

Servidor Web: Hardware de la estación de trabajo del servidor Web, donde se ejecutará el sistema

- Procesador Pentium III o superior 1x2 cache. 2.8 GHz.
- Memoria RAM 256 MBytes.
- Almacenamiento en discos de 5 GBytes.

Servidor de Base de Datos: Hardware de la estación de trabajo servidor BD.

- Procesador Pentium III o superior 1x2 cache. 2.8 GHz.
- Memoria RAM 512 MBytes.
- Almacenamiento en discos de 40 GBytes.

Cliente Web: Hardware de la estación de trabajo del cliente.

- Procesador Pentium III o superior o Clientes Ligeros.
- Memoria RAM 128 MBytes.

Recomendable

Servidor Web: Hardware de la estación de trabajo del servidor Web, donde se ejecutará el sistema.

- Procesador Pentium III o superior 2x2 cache. >= 3.00 GHz.
- Memoria RAM >= 512 MBytes.
- Almacenamiento en discos de 5 GBytes.

Servidor de Base de Datos: Hardware de la estación de trabajo servidor BD.

- Procesador Pentium D 2x2 cache. 3.00 GHz.´

- Memoria RAM >=1 GBytes.
- Almacenamiento en discos de 40 a 120 GBytes.

Cliente Web: Hardware de la estación de trabajo del cliente.

- Procesador Pentium III o superior o Clientes Ligeros.
- Memoria RAM >= 256 MBytes.

3.6 Requerimientos no Funcionales.

Los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como la características que hacen al producto atractivo, usable, rápido o confiable. Los mismos están vinculados con los requisitos funcionales, es decir una vez se conozca lo que el sistema debe hacer podemos determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande deber ser.

Categorías de los RNF.

- Requerimientos de Software.
- Requerimientos de Hardware.
- Restricciones en el diseño y la implementación
  - Estándares requeridos.
  - Lenguajes de programación a ser usados para la implementación.
  - Uso obligatorio de ciertas herramientas de desarrollo.
  - Restricciones en la arquitectura o el diseño.
  - Bibliotecas de clases.
- Requerimientos de apariencia o interfaz externa.

Nota: Es importante destacar que no se trata del diseño de la interfaz en detalle sino que especifican cómo se pretende que sea la interfaz externa del producto.

- Requerimientos de seguridad.
  - Confidencialidad.
  - Integridad.
  - Disponibilidad.
- Requerimientos de usabilidad.

Para el desempeño del Proyecto de Manugas y el desarrollo de la actual aplicación que se está implementando se tuvieron en cuenta los siguientes requisitos no funcionales:

3.6.1 Usabilidad.

- La aplicación debe ser concebida para ser utilizada por personas que tengan conocimientos básicos sobre informática y en el trabajo con los procesos de Facturación y Cobro.
- El sistema debe estar disponible las veinticuatro horas del día, sin ninguna interrupción.
- El sistema debe ser accesible desde todos los puntos donde exista una máquina conectada a la red.
- El sistema debe diferenciar las interfaces gráficas y opciones para los usuarios que accedan al sistema con diferentes roles.

3.6.2 Apariencia o Interfaz Externa.

- Diseño sencillo y fácil de usar, permitiendo que no sea necesario mucho entrenamiento para utilizar el sistema.

3.6.3 Confiabilidad.

- Ante fallas técnicas del sistema, corregirlas en un periodo menor a un mes.

Si este sistema presentara algún desperfecto, este será corregido en un período menor a un mes para que no afecte a sus usuarios. La manera de garantizar la mayor aceptación por parte del cliente es el soporte con que pueda contar el software dándole seguimiento a través de un procedimiento de soporte del software por los desarrolladores.

- El sistema debe tener la capacidad de identificar con certeza a los diversos usuarios o entidades que interactúan con él.
- El sistema debe tener la capacidad de darle seguridad al usuario, de que las informaciones solo serán vistas por quien esté capacitado para esto.

3.6.4 Rendimiento.

- El sistema debe responder en un tiempo relativamente rápido a las peticiones del usuario (menos de 1 segundos).

  Teniendo en cuenta que el sistema no es de tiempo real y no necesita de respuestas de milisegundos se plantea este como un tiempo máximo estimado el cual debe ser mucho menor, pero si se encuentra en estos límites no debe afectar el funcionamiento del sistema.

3.6.5 Soporte.

- El software se desarrolló usando PHP y herramientas de software libre, por tal motivo podrá operar en cualquier sistemas operativo con la excepción del SGBD SQL Server 2000 que solo corre sobre el Sistema Operativo Windows.

3.6.6 Requerimiento de ayuda y documentación.

- Proporcionar gran documentación de ayuda al usuario.

  Por ejemplo: Manual de usuario y otros documentos de ayuda donde se explicaría como trabajar con el sistema, que se haría ante situaciones extremas, cómo solucionar los posibles fallos, etc.

3.6.7 Restricciones de diseño e implementación.

- El sistema debe usar frameworks Symfony 1.011

- El sistema se desarrollará en lenguaje PHP 5.0 Orientado a Objeto.

3.6.8 Requerimientos de Hardware.

Mínimo

Servidor Web: Hardware de la estación de trabajo del servidor Web, donde se ejecutará el sistema

- Procesador Pentium III o superior 1x2 cache. 2.8 GHz.
- Memoria RAM 256 MBytes.
- Almacenamiento en discos de 5 GBytes.

Servidor de Base de Datos: Hardware de la estación de trabajo servidor BD.

- Procesador Pentium III o superior 1x2 cache. 2.8 GHz.
- Memoria RAM 512 MBytes.
- Almacenamiento en discos de 40 GBytes.

Cliente Web: Hardware de la estación de trabajo del cliente.

- Procesador Pentium III o superior o Clientes Ligeros.
- Memoria RAM 128 MBytes.

Recomendable

Servidor Web: Hardware de la estación de trabajo del servidor Web, donde se ejecutará el sistema

- Procesador Pentium III o superior 2x2 cache. >= 3.00 GHz.
- Memoria RAM >= 512 MBytes.
- Almacenamiento en discos de 5 GBytes.

Servidor de Base de Datos: Hardware de la estación de trabajo servidor BD.

- Procesador Pentium D 2x2 cache. 3.00 GHz.´

- Memoria RAM >=1 GBytes.
- Almacenamiento en discos de 40 a 120 GBytes.

Cliente Web: Hardware de la estación de trabajo del cliente.

- Procesador Pentium III o superior o Clientes Ligeros.
- Memoria RAM >= 256 MBytes.

3.6.9 Requerimientos de Software.

PC Cliente: Software instalado en la estación de trabajo del cliente.

- Sistema Operativo Windows 2000 o Superior.
- El Navegador Web compatible con HTML 2.0 y CSS, podrá ser Netscape 3 (o superior), Internet Explorer 4.2 (o superior) y compatibles.

PC Servidor Web: Software instalado en el Servidor Web.

- Servidor de Web Apache 2.2.1

PC Servidor de BD: Software instalado en el Servidor de Base de datos.

- Servidor de Bases de Datos SQL-Server 2000.

3.6.10 Seguridad.

La seguridad se refiere al mantenimiento de información sensible fuera del alcance de personas no autorizadas, ya sea porque quieran usarla con fines propios o porque quieran alterar su integridad.

La integridad de los datos se analiza a partir de 3 criterios fundamentales: la confidencialidad, la integridad y la disponibilidad.

- **Confidencialidad:** la información manejada por el sistema debe estar protegida de acceso no autorizado y divulgación.

  Crear cuentas de usuarios que contengan roles y permisos sobre el sistema.

- **Integridad:** hace referencia a la incorruptibilidad de los datos a lo largo del tiempo, es decir, que éstos no se alteren o se pierdan y puedan ser usados con posterioridad.
- **Disponibilidad:** se garantiza el acceso a un servicio o a los recursos, es decir, que la información esté disponible cuando se necesite.

3.7 Conclusiones.

En este capítulo se definió la línea base de la arquitectura. Se tuvo en cuenta para ello las herramientas de modelado, la estructura del equipo de desarrollo, las 4+1 vistas arquitectónicas, definiendo en la vista rectora de casos de uso los módulos en los que se divide el proyecto y se tuvo en cuenta además los requisitos no funcionales que son tan importantes para la arquitectura.

Conclusiones Generales

Al concluir esta investigación, se logró dar cumplimiento al objetivo trazado con el diseño de una arquitectura flexible, robusta y escalable, además de fácil de implementar y muy bien documentada para el Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado. Como resultado de la misma se generaron los artefactos correspondientes al rol Arquitecto de Software que se desglosan en la Especificación de Requerimientos, el Documento de Arquitectura de Software y el Modelo de Despliegue.

Recomendaciones

Se recomienda sobre la base de la Arquitectura diseñada:

- Certificar la arquitectura a través de uno de los métodos de evaluación (ATAM).
- Proponer como línea base de Arquitectura para proyectos similares que implementan el resultado de esta investigación.

Glosario de Términos

ACC: Autoridad Aeronáutica Civil.

API: La Interfaz de Programación de Aplicaciones, en inglés Application Programming Interface.

Applets: Es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web.

BD: Base de Datos.

Browser: Un navegador, navegador red o navegador web es un programa que permite visualizar la información que contiene una página web, ya esté está alojada en un servidor dentro de la WWW (World Wide Web, Red Global Mundial) o en uno local.

BSD: Distribución de Software Berkeley, en inglés Berkeley Software Distribution.

Casas Comerciales: Casas a nivel de municipio que gestionan los procesos de facturación y cobro a domicilio.

Casa Matriz Comercial (UEB comercial): Es la Casa Rectora en donde está instalado el Sistema: SISMET que será reemplazado por el actual sistema en desarrollo: Manugas para gestionar los procesos de facturación y cobro con los datos que le provee las Casas Comerciales.

CASE: Ingeniería de Software Asistida por Ordenador, en inglés Computer Aided Software Engineering.

Cliente metrado: Es el que tiene firmado el contrato con la Empresa de Gas Manufacturado. Tiene en su casa el metro contador.

Cliente no metrado: Es el que tiene firmado el contrato con la Empresa de Gas Manufacturado. No cuenta con un metro contador en su casa.

CommTrack: Sistema de Facturación para la industria de las Telecomunicaciones.

CSS: Hojas de Estilo en Cascada, en ingles Cascading Style Sheets.

CUPET: Unión Cuba Petróleo.

CUS: Casos de Uso del Sistema.

CVS: Sistema Concurrente de Versiones, en inglés Concurrent Versions System.

Data mining: Minería de datos.

Estándar de Facto: Son documentos o herramientas que son seguidas y usadas por la industria como si hubieran sido aprobadas por organismos oficiales, pero que en realidad no lo son. En sí se trata de una norma generalmente aceptada y ampliamente utilizada por iniciativa propia de un gran número de interesados.

E-xtendgfp: Plataforma para la gestión de facturas y pagos.

GoF: Gang of Four, "pandilla de los cuatro".

GPL: General Public License, Licencia Pública General

GRAPS: Patrones de Asignación de Responsabilidades.

GUI: Constructor de Interfaz Gráfica.

HTML: Lenguaje de Marcas de Hipertexto, en inglés HyperText Markup Language.

HTTP: Protocolo de transferencia de hipertexto, en inglés HyperText Transfer Protocol.

IDE: Entorno de Desarrollo Integrado, en inglés Integrated Development Environment

IEEE: Instituto de Ingenieros Eléctricos y Electrónicos, en inglés The Institute of Electrical and Electronics Engineers.

FYC4: Sistema de Facturación y Cobros para la facturación automatizada y digital de las tasas de sobrevuelos.

IIS: Internet Information Server es una serie de servicios para los ordenadores que funcionan con Windows

Login: Es el momento de autentificación al ingresar a un servicio o sistema.

LAMP: Se refiere a un conjunto de subsistemas de software necesarios para alcanzar una solución global, en este caso configurar sitios web o Servidores dinámicos con un esfuerzo reducido.

Mero: Simple, sencillo, elemental.

Microsoft Visio: Es un software de dibujo vectorial para Microsoft Windows.

Módulos: Es un componente bueno auto controlado de un sistema, el cual posee una interfaz bien definida hacia otros componentes.

Monolítica: Muy compacto, con una unión tan fuerte entre sus distintas partes como si fuera de una sola pieza.

MSF: Marco de Solución Microsoft, en inglés Microsoft Solution Framework.

MVC: Modelo Vista Controlador.

PDA: Asistente Digital Personal, en inglés Personal Digital Assistant.

PetroSoft: Soluciones Informáticas para la Industria del Petróleo.

PHP: Pre-procesador de Hipertexto, en inglés Hypertext Pre-processor.

Plugins: Es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy especifica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.

Query: Consulta.

RAM: Memoria de Acceso Aleatorio, Random Access Memory.

RCP: Plataforma de clientes ricos, en inglés Rich Client Platform.

RNF: Requerimientos no Funcionales.

RUP: El Proceso Unificado de Software, en inglés Rational Unified Process.

SAD: Sistema Automatizado de Dirección.

SGBD: Sistemas Gestores de Base de Datos.

SISMET: Sistema Metrado.

SOA: Arquitectura Orientada a Servicios.

Solapándose: Cubriendo una cosa a otra en su totalidad o en parte.

Struts: Es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC.

Subversion: Es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS.

SSL: Protocolo de Capa de Conexión Segura, en inglés Secure Sockets Layer.

Testear: Someter a una persona o una cosa a un test.

TIC: Tecnologías de la Información y la Comunicación.

TPL: Dispositivo Portátil de Lectura.

Trazabilidad: Es un conjunto de acciones, medidas y procedimientos técnicos que permite identificar y registrar cada producto desde su nacimiento hasta su muerte.

UCI: Universidad de las Ciencias Informáticas.

UML: Lenguaje Unificado de Modelado, en inglés Unified Modeling Languaje.

URL: Localizador Uniforme de Recurso, en inglés Uniform Resource Locator.

Workflow: Flujos de trabajo.

XML: Lenguaje de Marcas, en inglés Extensible Markup Language.

XP: Programación Extrema, en inglés Extreming Programing.

Referencias Bibliográficas

1. **Higazy Rivero, Ing. Roberto.** Sociedad de Ingenieros de Bolivia S.I.B. [En línea] [Citado el: 3 de febrero de 2009.] http://www.sibsc.com/articulos/instalaciones_de_gas.htm...

2. **Harrington, James.** *"Business Process Improveement".* 1991.

3. **Barros, Oscar.** *"Reingeniería de Procesos de negocio".* Chile : Dolmen, 1994.

4. Teleformacion.uci.cu. [En línea] [Citado el: 3 de febrero de 2009.] http://teleformacion.uci.cu/course/view.php?id=102...

5. Jacobson, Ivar. Booch, Grady. Rumbaugh, James. *El Proceso Unificado de Desarrollo. Vol I, Vol II.* La Habana, Cuba. : Editorial Félix Varela., 2004.

6. **Casanovas, Josep.** DesarrolloWeb. [En línea] [Citado el: 6 de febrero de 2009.] http://www.desarrolloweb.com/articulos/1622.php..

7. **Clements, Paul, Bass, len y Kazman, Rick.** *Arquitectura de Software en la práctica (2 ª edición).* s.l. : Addison-Wesley Profesional, 2003. 0321154959.

8. **Canal Velasco, D. Carlos.** *"Un Lenguaje para la Especificación y Validación de Arquitecturas de Software".* 2000.

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

10. **Klein, Mark y Kazman, Rick.** *"Attribute-based architectural styles".* 1999.

11. Microsoft Download. [En línea] [Citado el: 23 de febrero de 2009.] http://download.microsoft.com/download/c/2/c/c2ce8a3a-b412-ba18-7e050aef3364/070717-Read_world_SOA.pdf.

12. **Reynoso, Carlos y Kicillof, Nicolás.** *"Estilos y Patrones en la Estrategia de Arquitectura de Microsoft".* BUENOS AIRES : s.n., 2004.

13. **Alexander, Christophe.** *"The Timeless Way of Building".* 1979.

14. **Oktaba, Hanna.** *"Introducción a Patrones".* Mexico : s.n.

15. **Buschmann, Frank.** *"Pattern-Oriented Software Architecture, Volume 1: A System of Patterns".* 1996.

16. U-Cursos. [En línea] [Citado el: 16 de febrero de 2009.] https://www.u-cursos.cl/ingenieria/2008/1/CC31B/1/material_docente/objeto/174674..

17. **Vásquez, Ivis Rosa, Carrillo, Pablo Andres y Bibiana García, Sorey.** slideshare. [En línea] [Citado el: 20 de febrero de 2009.] http://www.slideshare.net/soreygarcia/el-roldel-arquitecto-exposicion..

18. **Pizarro Mendoza, Pablo.** Arquitectura-de-software. [En línea] [Citado el: 18 de febrero de 2009.] http://arquitectura-de-software.blogspot.com/2006/12/qu-es-un-arquitecto-de-software.html...

19. **Gutiérrez, Javier J.** Departamento de Lenguajes y Sistemas Informáticos. [En línea] [Citado el: 27 de febrero de 2009.] http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf.

20. CakePHP. [En línea] [Citado el: 27 de febrero de 2009.] http://cakephp.org/.

21. Dokeos Latinoamérica. [En línea] [Citado el: 26 de febrero de 2009.] http://dokeoslatinoamerica.wordpress.com/2009/01/14/algunos-frameworks-para-php-mas-usados/.

22. Wikipedia. [En línea] [Citado el: 26 de febrero de 2009.] http://es.wikipedia.org/wiki/Kumbia.

23. **Leopoldo, Carlos.** Blog personal: CarlosLeopoldo. [En línea] [Citado el: 27 de febrero de 2009.] http://www.carlosleopoldo.com/post/zend-framework-una-introduccion/.

24. Marble Station. [En línea] [Citado el: 28 de febrero de 2009.] http://www.marblestation.com/blog/?p=644.

25. **Mendoza Sanchez, María A.** informatízate. [En línea] [Citado el: 2 de marzo de 2009.] httpwww.willydev.netInsiteCreationv1.0descargascualmetodologia.pdf.

26. *Herramientas Case.* s.l. : INSTITUTO NACIONAL DE ESTADISTICA E INFORMATICA.

27. **León, Eduardo.** Blog personal. [En línea] [Citado el: 4 de marzo de 2009.] http://slion2000.blogspot.com/2007/04/visual-paradigm-una-herramienta-de-lo.html.

28. Wikipedia. [En línea] [Citado el: 3 de marzo de 2009.] http://es.wikipedia.org/wiki/Rational_Software.

29. Garbage Collector. [En línea] [Citado el: 6 de marzo de 2009.] http://www.error500.net/garbagecollector/archives/categorias/bases_de_datos/sistema_gestor_de_base_de_datos_sgbd.php.

30. ORACLE. [En línea] [Citado el: 6 de marzo de 2009.] http://www.oracle.com/global/es/products/database/db_manageability.html. .

31. HTTP-PERU. [En línea] [Citado el: 8 de marzo de 2009.] http://www.http-peru.com/postgresql.php.

32. Wikipedia. [En línea] [Citado el: 8 de marzo de 2009.] http://es.wikipedia.org/wiki/SQL_Server.

33. desarrolloweb. [En línea] [Citado el: 14 de marzo de 2009.] http://www.desarrolloweb.com/articulos/392.php.

34. **Serrano Castañeda, Jairo Enrique.** savio.unitecnologica.edu.co. [En línea] [Citado el: 2009 de abril de 11.] http://www.jsnat.com.

35. **Marqués Andrés, María Mercedes.** Uji.es. [En línea] [Citado el: 2009 de abril de 11.] http://www3.uji.es/~mmarques/f47/apun/node7.html.

35. maestrosdelweb. [En línea] [Citado el: 20 de abril de 2009.] http://www.maestrosdelweb.com/principiantes/los-diferentes-lenguajes-de-programacion-para-la-web/.

36. wikipedia. [En línea] [Citado el: 25 de abril de 2009.] http://es.wikipedia.org/wiki/Ambiente_integrado_de_desarrollo.

37. wikipedia. [En línea] [Citado el: 25 de abril de 2009.] http://es.wikipedia.org/wiki/Eclipse_(software).

38. wikipedia. [En línea] [Citado el: 25 de abril de 2009.] http://es.wikipedia.org/wiki/Ambiente_integrado_de_desarrollo.

39. Sistema de Facturación CommTrack. [En línea] [Citado el: 5 de mayo de 2009.] http://www.techmarksoftware.com/media/tsi_pdf/cbs4_espanol.pdf.

40. Modelos avanzados de comunicación de recursos. [En línea] [Citado el: 7 de mayo de 2009.] https://forge.morfeo-project.org/wiki/index.php/D.3.2.2_Modelos_avanzados_de_comunicaci%C3%B3n_de_recursos\.

41. El tutorial Jobeet.

42. **Georgas, John C., Dashofy, Eric M. y Taylor, Richard N**. ACM: Association for Computing Machinery. [En línea] [Citado el: 5 de mayo de 2009.] http://www.acm.org/crossroads/espanol/xrds12-4/arqcentric.html.

43. **Reynoso, Carlos y Kiccillof, Nicolás. WillyDev**. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. [En línea] [Citado el: 7 de mayo de 2009.] http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/estiloypatron.pdf.

44. **Baptiste, Juan Luis**. Introducción a QtJambi. Colombia : Campus Party Colombia, 2008.

45. **BRENDEL, JENS-CHRISTOPH y SCHÖCHLIN, MARC**. Mejores Copias de Seguridad con el Sistema de Backup Bacula. [En línea] [Citado el: 10 de mayo de 2009.] http://www.linux-magazine.es/issue/10/Bacula.pdf.

46. **JMPereda.** ¿Qué es una Aplicación Web? [En línea] [Citado el: 10 de mayo de 2009.] http://jmpereda.wordpress.com/2007/08/24/definiendo-la-plantilla/.

47. Las 4+1 Vistas. [En línea] [Citado el: 10 de mayo de 2009.] http://synergix.wordpress.com/2008/07/31/las-4-mas-1-vistas/.

48. **Amaya Flores, Elvira, Estrada Espinosa, Carlos y Manrique Ramirez, Joel.** Resumen del libro "Lenguaje Unificado de Modelado" Disponible en: http://azul.bnct.ipn.mx/Entidades_vinculadas/Resumen_uml.doc. s.l. : ADISON WESLEY, 2003.

50. MySQL. [En línea] [Citado el: 15 de mayo de 2009.] http://www.mysql.com/common/pages/download_access_denied.html.

51. SQL Server. [En línea] [Citado el: 15 de mayo de 2009.] http://www.uaem.mx/posgrado/mcruz/cursos/miic/sql.pdf.

52. Autoridad Aeronautica Civil. *Implementa sistema de facturación y cobro FYC4.* [En línea] [Citado el: 12 de mayo de 2009.] http://www.aeronautica.gob.pa/index.php?option=com_content&task=view&id=522&Itemid=2.

53. Bode, Carlos. "Más de 350 empresas ya usan en España nuestro sistema de facturación electrónica". [En línea] [Citado el: 12 de mayo de 2009.] https://www.bancoherrero.com/empresa/es/SERVICIOS_PARA_LA_EMPRESA/E-BUSINESS/E-FACTURA/NOTICIAS_DE_E-FACTURA/NOTICIAS_E-FACTURA_1/.

54. Monografias.com. Evolución y Orientaciones de Patrones. [En línea] [Citado el: 15 de mayo de 2009.] http://www.monografias.com/trabajos27/evolucion-patrones/evolucion-patrones.shtml#tipos.

55. Pesentación de Metodologías MSF(Microsoft Solutions Framework). [En línea] [Citado el: 15 de mayo de 2009.] http://www.egattaca.com/econtent/library/documents/DocNewsNo50DocumentNo6.PDF.

56. Proyecto Implementación Sistema Integrado de Gestión Empresarial. [En línea] [Citado el: 15 de mayo de 2009.] http://equiposige.blogspot.es/img/VisionYAlcance.doc.

57. Anuncios Google. Lenguajes del lado servidor o cliente. [En línea] [Citado el: 20 de mayo de 2009.] http://www.adelat.org/media/docum/nuke_publico/lenguajes_del_lado_servidor_o_cliente.html.

58. Hooping.net. Introducción al lenguaje html. [En línea] [Citado el: 20 de mayo de 2009.] http://www.hooping.net/faq-html.aspx.

Bibliografía Consultada

1. **CAMACHO, ERIKA y NÚÑEZ, GABRIEL.** *Arquitecturas de Software.* [En línea] Abril de 2004. http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf .

2. **MORENO, ANA M.** *Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el momento Arquitectónico.*

3. PostgreSQL Práctico. [En línea] 2001. http://www.sobl.org/traducciones/practical-postgres/node19.html .

4. **Lasso, A.** *Arquitectura de Software.* [En línea] http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art110.asp .

5. **RODRÍGUEZ, MARÍA L., y otros.** *Hacia la satisfación de requisitos de la calidad de los sistemas colaborativos mediante un diseño arquitectónico.*

6. **PARRA D.J.** *Hacia una Arquitectura Empresarial basada en Servicios.* [En línea] http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art143.asp.

7. **Bastarrica M.C.** *Atributos de Calidad y Arquitectura del Software. Volumne 4.* 2005.

8. **Booch, G.** *The Unified Modeling Language User Guide.* s.l. : Addison Wesley, 1999.

9. **Bosch J.** *Design & Use of Software Architecture.* s.l. : Addison Wesley, 2000.

10. **Perry, Dewayne E. y Wolf, Alexander L.** *Foundations for the Study of Software Architecture. Vol 17.* New York : ACM SIGSOFT, 1992. 0163-5948..

11. **Garlan, David y Shaw, Mary.** *An Introduction to Software Architecture. Vol 1.* New Jersey : V.Ambriola and G.Tortora, 1994. 15213-3890.

12. **Montalvo, David Peg.** *Seguridad en la transmisión de Datos.* Santiago de Compostela : s.n., 2005.

13. PostgreSQL. *PostgreSQL Global Development Group.* [En línea] tinysofa, 1996. http://www.postgresql.org/..

14. Grupo Soluciones Innova S.A. G.S.I Grupo Soluciones Innova. [En línea] http://www.rational.com.ar/herramientas/roseenterprise.html.

15. Company Headquarters. Visual Paradigm. [En línea] 2002. http://www.visual-paradigm.com/.

16. **Potencier, Fabien y Zaninotto, François.** *Symfony,la guía definitiva. Vol 1.0, Vol 1.1* .

17. **Medvidovic, y otros.** *Domains of Concern in Software Architectures and Architecture Description Languages.* California : s.n., 1997. 92697-3425.

18. **Zarzuela, Jorge Ferrer.** *Metodologías Ágiles.* [En línea] http://libresoft.dat.escet.urjc.es/html/downloads/ferrer-20030312.pdf] .

19. **Higazy Rivero, Ing. Roberto.** Sociedad de Ingenieros de Bolivia S.I.B. [En línea] [Citado el: 3 de febrero de 2009.] http://www.sibsc.com/articulos/instalaciones_de_gas.htm...

20. **Harrington, James.** *"Business Process Improveement".* 1991.

21. **Barros, Oscar.** *"Reingeniería de Procesos de negocio".* Chile : Dolmen, 1994.

22. Teleformacion.uci.cu. [En línea] [Citado el: 3 de febrero de 2009.] http://teleformacion.uci.cu/course/view.php?id=102...

23. Jacobson, Ivar. Booch, Grady. Rumbaugh, James. *El Proceso Unificado de Desarrollo. Vol I, Vol II.* La Habana, Cuba. : Editorial Félix Varela., 2004.

24. **Casanovas, Josep.** DesarrollloWeb. [En línea] [Citado el: 6 de febrero de 2009.] http://www.desarrolloweb.com/articulos/1622.php..

25. **Clements, Paul, Bass, len y Kazman, Rick.** *Arquitectura de Software en la práctica (2 ª edición).* s.l. : Addison-Wesley Profesional, 2003. 0321154959.

26. **Canal Velasco, D. Carlos.** *"Un Lenguaje para la Especificación y Validación de Arquitecturas de Software".* 2000.

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

28. **Klein, Mark y Kazman, Rick.** *"Attribute-based architectural styles".* 1999.

29. Microsoft Download. [En línea] [Citado el: 23 de febrero de 2009.] http://download.microsoft.com/download/c/2/c/c2ce8a3a-b412-ba18-7e050aef3364/070717-Read_world_SOA.pdf.

30. **Reynoso, Carlos y Kicillof, Nicolás.** *"Estilos y Patrones en la Estrategia de Arquitectura de Microsoft".* BUENOS AIRES : s.n., 2004.

31. **Alexander, Christophe.** *"The Timeless Way of Building".* 1979.

32. **Oktaba, Hanna.** *"Introducción a Patrones".* Mexico : s.n.

33. **Buschmann, Frank.** *"Pattern-Oriented Software Architecture, Volume 1: A System of Patterns".* 1996.

34. U-Cursos. [En línea] [Citado el: 16 de febrero de 2009.] https://www.u-cursos.cl/ingenieria/2008/1/CC31B/1/material_docente/objeto/174674..

35. **Vásquez, Ivis Rosa, Carrillo, Pablo Andres y Bibiana García, Sorey.** slideshare. [En línea] [Citado el: 20 de febrero de 2009.] http://www.slideshare.net/soreygarcia/el-roldel-arquitecto-exposicion..

36. **Pizarro Mendoza, Pablo.** Arquitectura-de-software. [En línea] [Citado el: 18 de febrero de 2009.] http://arquitectura-de-software.blogspot.com/2006/12/qu-es-un-arquitecto-de-software.html...

37. **Gutiérrez, Javier J.** Departamento de Lenguajes y Sistemas Informáticos. [En línea] [Citado el: 27 de febrero de 2009.] http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf.

38. CakePHP. [En línea] [Citado el: 27 de febrero de 2009.] http://cakephp.org/.

39. Dokeos Latinoamérica. [En línea] [Citado el: 26 de febrero de 2009.] http://dokeoslatinoamerica.wordpress.com/2009/01/14/algunos-frameworks-para-php-mas-usados/.

40. Wikipedia. [En línea] [Citado el: 26 de febrero de 2009.] http://es.wikipedia.org/wiki/Kumbia.

41. **Leopoldo, Carlos.** Blog personal: CarlosLeopoldo. [En línea] [Citado el: 27 de febrero de 2009.] http://www.carlosleopoldo.com/post/zend-framework-una-introduccion/.

42. Marble Station. [En línea] [Citado el: 28 de febrero de 2009.] http://www.marblestation.com/blog/?p=644.

43. **Mendoza Sanchez, María A.** informatízate. [En línea] [Citado el: 2 de marzo de 2009.] httpwww.willydev.netInsiteCreationv1.0descargascualmetodologia.pdf.

44. *Herramientas Case.* s.l. : INSTITUTO NACIONAL DE ESTADISTICA E INFORMATICA.

45. **León, Eduardo.** Blog personal. [En línea] [Citado el: 4 de marzo de 2009.] http://slion2000.blogspot.com/2007/04/visual-paradigm-una-herramienta-de-lo.html.

46. Wikipedia. [En línea] [Citado el: 3 de marzo de 2009.] http://es.wikipedia.org/wiki/Rational_Software.

47. Garbage Collector. [En línea] [Citado el: 6 de marzo de 2009.] http://www.error500.net/garbagecollector/archives/categorias/bases_de_datos/sistema_gestor_de_base_de_datos_sgbd.php.

48. ORACLE. [En línea] [Citado el: 6 de marzo de 2009.] http://www.oracle.com/global/es/products/database/db_manageability.html. .

49. HTTP-PERU. [En línea] [Citado el: 8 de marzo de 2009.] http://www.http-peru.com/postgresql.php.

50. Wikipedia. [En línea] [Citado el: 8 de marzo de 2009.] http://es.wikipedia.org/wiki/SQL_Server.

51. desarrolloweb. [En línea] [Citado el: 14 de marzo de 2009.] http://www.desarrolloweb.com/articulos/392.php.

52. **Serrano Castañeda, Jairo Enrique.** savio.unitecnologica.edu.co. [En línea] [Citado el: 2009 de abril de 11.] http://www.jsnat.com.

53. **Marqués Andrés, María Mercedes.** Uji.es. [En línea] [Citado el: 2009 de abril de 11.] http://www3.uji.es/~mmarques/f47/apun/node7.html.

54. maestrosdelweb. [En línea] [Citado el: 20 de abril de 2009.] http://www.maestrosdelweb.com/principiantes/los-diferentes-lenguajes-de-programacion-para-la-web/.

55. wikipedia. [En línea] [Citado el: 25 de abril de 2009.] http://es.wikipedia.org/wiki/Ambiente_integrado_de_desarrollo.

56. wikipedia. [En línea] [Citado el: 25 de abril de 2009.] http://es.wikipedia.org/wiki/Eclipse_(software).

57. wikipedia. [En línea] [Citado el: 25 de abril de 2009.] http://es.wikipedia.org/wiki/Ambiente_integrado_de_desarrollo.

58. Sistema de Facturación CommTrack. [En línea] [Citado el: 5 de mayo de 2009.] http://www.techmarksoftware.com/media/tsi_pdf/cbs4_espanol.pdf.

59. Modelos avanzados de comunicación de recursos. [En línea] [Citado el: 7 de mayo de 2009.] https://forge.morfeo-project.org/wiki/index.php/D.3.2.2_Modelos_avanzados_de_comunicaci%C3%B3n_de_recursos\.

60. El tutorial Jobeet.

61. **Georgas, John C., Dashofy, Eric M. y Taylor, Richard N**. ACM: Association for Computing Machinery. [En línea] [Citado el: 5 de mayo de 2009.] http://www.acm.org/crossroads/espanol/xrds12-4/arqcentric.html.

62. **Reynoso, Carlos y Kiccillof, Nicolás. WillyDev**. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. [En línea] [Citado el: 7 de mayo de 2009.] http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/estiloypatron.pdf.

63. **Baptiste, Juan Luis**. Introducción a QtJambi. Colombia : Campus Party Colombia, 2008.

64. **BRENDEL, JENS-CHRISTOPH y SCHÖCHLIN, MARC**. Mejores Copias de Seguridad con el Sistema de Backup Bacula. [En línea] [Citado el: 10 de mayo de 2009.] http://www.linux-magazine.es/issue/10/Bacula.pdf.

65. **JMPereda.** ¿Qué es una Aplicación Web? [En línea] [Citado el: 10 de mayo de 2009.] http://jmpereda.wordpress.com/2007/08/24/definiendo-la-plantilla/.

66. Las 4+1 Vistas. [En línea] [Citado el: 10 de mayo de 2009.] http://synergix.wordpress.com/2008/07/31/las-4-mas-1-vistas/.

67. **Amaya Flores, Elvira, Estrada Espinosa, Carlos y Manrique Ramirez, Joel.** Resumen del libro "Lenguaje Unificado de Modelado" Disponible en: http://azul.bnct.ipn.mx/Entidades_vinculadas/Resumen_uml.doc. s.l. : ADISON WESLEY, 2003.

68. MySQL. [En línea] [Citado el: 15 de mayo de 2009.] http://www.mysql.com/common/pages/download_access_denied.html.

69. SQL Server. [En línea] [Citado el: 15 de mayo de 2009.] http://www.uaem.mx/posgrado/mcruz/cursos/miic/sql.pdf.

70. Autoridad Aeronautica Civil. *Implementa sistema de facturación y cobro FYC4.* [En línea] [Citado el: 12 de mayo de 2009.] http://www.aeronautica.gob.pa/index.php?option=com_content&task=view&id=522&Itemid=2.

71. Bode, Carlos. "Más de 350 empresas ya usan en España nuestro sistema de facturación electrónica". [En línea] [Citado el: 12 de mayo de 2009.] https://www.bancoherrero.com/empresa/es/SERVICIOS_PARA_LA_EMPRESA/E-BUSINESS/E-FACTURA/NOTICIAS_DE_E-FACTURA/NOTICIAS_E-FACTURA_1/.

72. Monografias.com. Evolución y Orientaciones de Patrones. [En línea] [Citado el: 15 de mayo de 2009.] http://www.monografias.com/trabajos27/evolucion-patrones/evolucion-patrones.shtml#tipos.

73. Pesentación de Metodologías MSF(Microsoft Solutions Framework). [En línea] [Citado el: 15 de mayo de 2009.] http://www.egattaca.com/econtent/library/documents/DocNewsNo50DocumentNo6.PDF.

74. Proyecto Implementación Sistema Integrado de Gestión Empresarial. [En línea] [Citado el: 15 de mayo de 2009.] http://equiposige.blogspot.es/img/VisionYAlcance.doc.

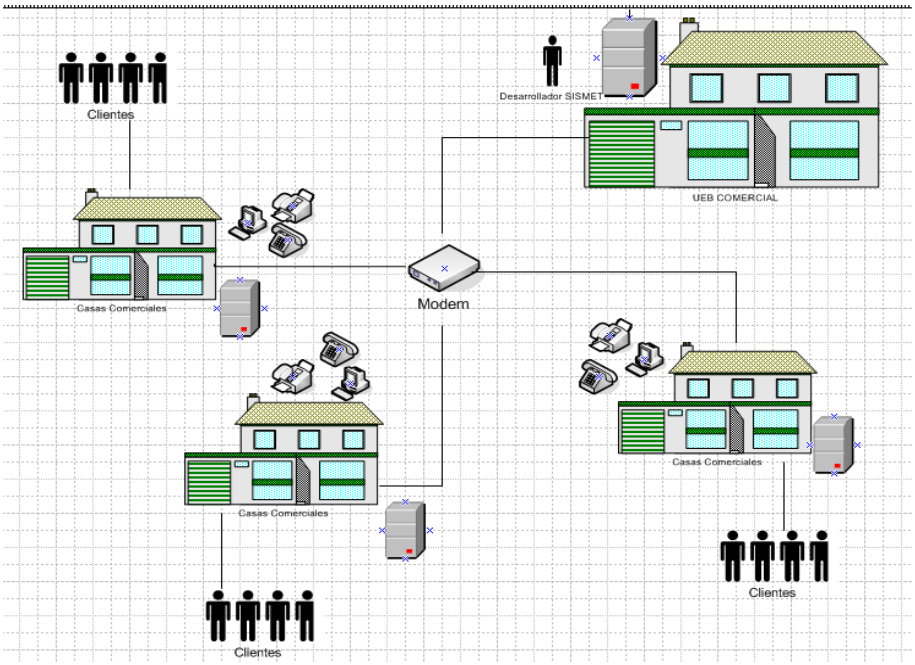
75. Anuncios Google. Lenguajes del lado servidor o cliente. [En línea] [Citado el: 20 de mayo de 2009.]

http://www.adelat.org/media/docum/nuke_publico/lenguajes_del_lado_servidor_o_cliente.html.

76. Hooping.net. Introducción al lenguaje html. [En línea] [Citado el: 20 de mayo de 2009.] http://www.hooping.net/faq-html.aspx.

Anexos.

Anexo # 1. Organigrama de la Empresa de Gas Manufacturado.

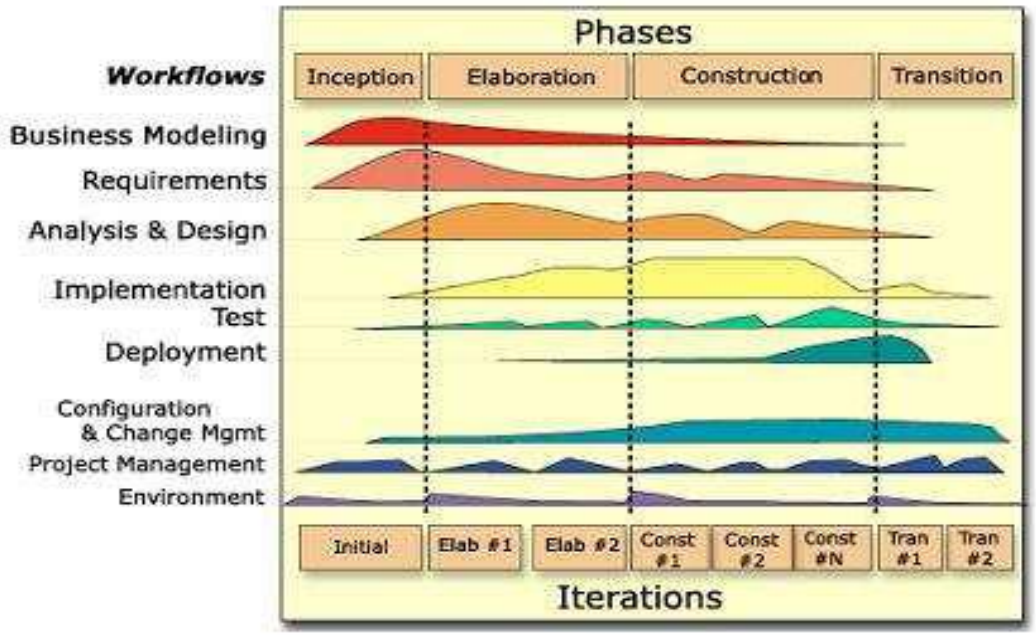


Anexo #2. Metodología XP.

La figura muestra cada una de las fases del desarrollo siguiendo esta metodología, Planificación, Diseño, Codificación y Pruebas. (44)

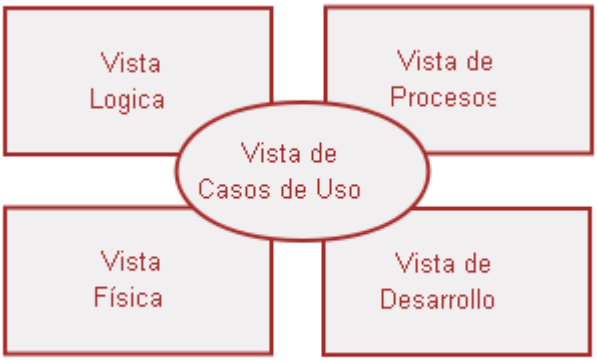
Flujos de Trabajo de la Metodología XP

Anexo #3. Flujo de Trabajo de RUP (45)

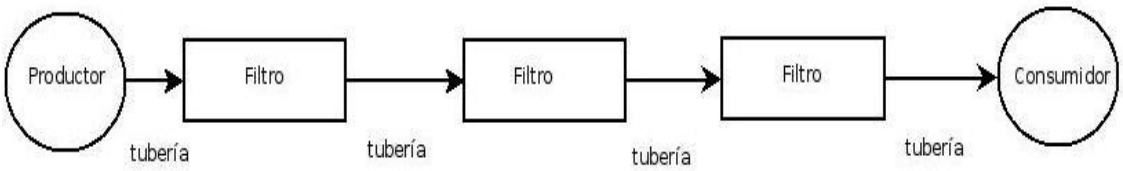


Flujos de Trabajo, Fases y esfuerzo por Iteración

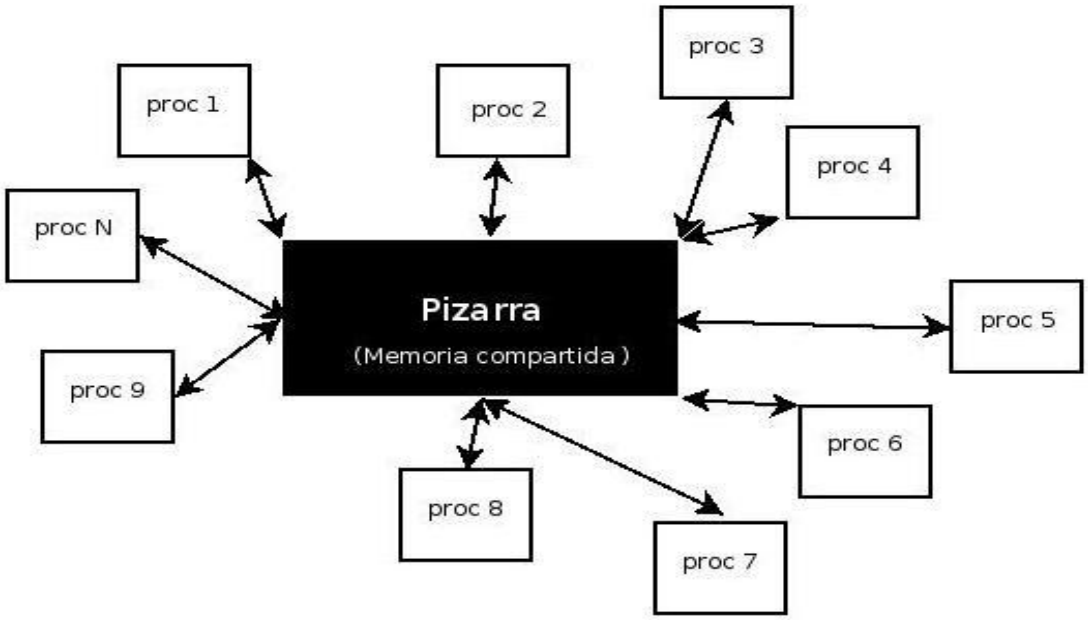
Anexo #4. Vistas Arquitectónicas. Modelo 4+1. (46)



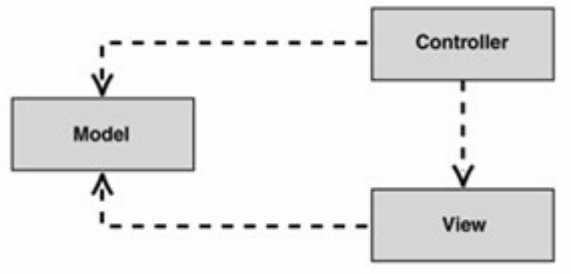
Anexo #5. Arquitectura Tubería/Filtro. (40)



Anexo #6. Arquitectura de Pizarra. (40)



Anexo #7. Arquitectura Modelo Vista Controlador. (43)



Anexo #8. Arquitectura en Capas. (40)