**UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS**

**Facultad 9**


**TÍTULO:** Arquitectura de Software del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo.

**TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS**

**AUTOR:** Frank Hector Rios Morales

**TUTOR:** Ing. Daniel Sampedro Bello

**CO-TUTOR:** Ing. David Tavares Cuevas

**Ciudad de la Habana, mayo de 2010. "Año 52 de la Revolución"**

**DEDICATORIA**

Dedico esta tesis a mi mamá por siempre estar a mi lado, por el sacrificio que ha hecho por mi para que yo llegue a donde estoy hoy. Por cuidarme y enseñarme todo lo que se.
Michi eres la mujer más bella del mundo todo lo que soy te lo debo a ti, atribuyo todo mi éxito en la vida a ti, la enseñanza moral, intelectual y física que he recibido de ti, Te quiero con todo mi corazón.

A mi papá que es la persona que más respeto y admiro, eres la persona más maravillosa de la tierra. A ti, que junto a mamá, supiste estar siempre a nuestro lado en los momentos buenos y malos, te quiero mucho puro.

A mi hermano al que quiero mucho y cuido en todo momento, sabes que tú y yo estaremos siempre juntos y unidos, te agradezco que seas mi hermano y amigo. Te quiero mucho y nunca olvidaré todo lo que has hecho por mí en la vida, y los buenos recuerdos que tenemos juntos, pero no te creas cosas que tu enseguida te creces.

A mi hermanita del alma, a ella que es la más grande de los tres hermanos, y la que siempre nos cuida bueno a mí, porque a Fredy no, te agradezco por ser amiga, compañera y hermana. I love you.

A Dariela mi compañera, amiga, novia y dentro de unos años mujer y madre de mis hijos porque quiero tener muchos contigo, gracias por ayudarme tanto en mi vida y en la tesis, gracias por estar siempre a mi lado en los momentos malos y buenos, desde que te conocí mi vida cambio mucho, te quiero con todo mi corazón.

**AGRADECIMIENTOS**

A mis padres Maribel Morales López y Héctor Rios Moya por su apoyo, compresión y por la confianza que han depositado en mí. A ustedes gracias por hacer este sueño realidad.

A mis hermanos Fredy Hector Rios Morales y Yarely Rios Morales ustedes son mi mayor inspiración.

A toda mi familia que me apoyo tanto a mis tíos y tías en especial a Belkis y Marielena, abuelo y abuela a los que quiero mucho a pesar de no pensar igual que yo porque se quedaron congelados en el tiempo, Onelio, Irma y Mima. A los que siguieron más de cerca mis pasos y a los que no también. En especial a Dariela que desde que entro a mi vida la llevó por el buen camino y que me quiere mucho.

A mi suegra Bertica que es la mejor suegra que he tenido y que me ha ayudado mucho en todo lo que ha podido sin ella muchas cosas de mi vida no se hubiesen logrado.

A mis amigos Karen Beatriz Alvarez Fernández, Antonio Antelo, Yuneiry Barroso, Pedro Luis Lantigua (el flaco), Pedro Abigantus, y en especial a Cleyver John Sánchez (el salvaje).

A mi tutor Daniel Sampedro Bello gracias por estar ahí y ayudarme a tomar las decisiones correctas, sin ti el desarrollo de este trabajo no hubiese sido posible. La calidad del resultado de esta tesis se la debo a él. Al tribunal por haberme ayudado. En especial a Nilberto por ser más que mi Jefe de Tribunal, ser fuerte y exigirme tanto. Gracias.

**DECLARACIÓN DE AUTORÍA**

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____ _____

Frank Hector Rios Morales Daniel Sampedro Bello

**DATOS DE CONTACTO**

**Tutor:**

**Nombre y apellidos:** Daniel Sampedro Bello.

**Sexo:** ___M___ **Institución:** Universidad de las Ciencias Informáticas (UCI).

**Dirección de la institución:** Carretera San Antonio de Baños Km 2 ½, Boyero.

**Correo electrónico:** dsampedro@uci.cu.

**Teléfono particular:** 835 8896.

**Título de la especialidad de graduado:** Ingeniero en Ciencias Informáticas.

**Año de graduación:** 2007.

**Institución donde se graduó:** Universidad de las Ciencias Informáticas.

**Cotutor:**

**Nombre y apellidos:** David Tavares Cuevas

**Sexo:** __M____ **Institución:** Universidad de las Ciencias Informáticas (UCI).

**Dirección de la institución:** Carretera San Antonio de los Baños, Km 2 ½, Boyero.

**Correo electrónico:** dtavares@uci.cu.

**Teléfono particular:** 835 8894.

**Título de la especialidad de graduado:** Ingeniero en Ciencias Informáticas

**Año de graduación:** 2007.

**Institución donde se graduó:** Universidad de las Ciencias Informáticas (UCI).

**RESUMEN**

La presente investigación proporciona una Arquitectura de Software del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo para el desarrollo de una aplicación que permita la gestión y el control de la información relacionada con la graficación de columnas litológicas de los pozos en perforación del país y que permita integrarse fácilmente al proyecto productivo SIPP (Sistema de Manejo Integral de Perforación de Pozos) del CEINPET. Para alcanzar el objetivo propuesto, que no es más que un solución arquitectónica que permita posteriormente el desarrollo de un producto que satisfaga las necesidades del cliente, con la calidad y el costo requeridos se realizó un exhaustivo estudio de todos los procesos del CEINPET relacionados con la graficación de dichas columnas, además de un análisis de todos los fundamentos de la ingeniería de software necesarios para la propuesta de arquitectura del sistema a implementar.

En los diferentes capítulos de esta investigación se abordan las principales herramientas, tecnologías y metodología a utilizar para el desarrollo del sistema, se define la línea base de la arquitectura que proporciona un modelo de descripción de la arquitectura a través de las vistas definidas por la metodología RUP (Proceso Unificado de Desarrollo), además se genera como resultado de este trabajo el Documento de Arquitectura de Software del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo.

Finalmente, como principal resultado de la investigación se obtiene una sólida arquitectura del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo, que permite a los trabajadores del CEINPET todo el proceso de graficación de columnas de una forma sencilla y eficiente, ahorrándole a la empresa tiempo y dinero de inversión en otros software propietarios e inapropiados.

**PALABRAS CLAVES**

Arquitectura, Tecnologías, Metodologías, Vistas, Línea Base, Petróleo.

**Abstract**

This research provides an architectural solution for the Software Architecture of the System for Graphing lithological Columns of Oil Wells for the development of an application that enables the management and control of information related to graphing lithologic columns for drilling wells at the country and this application must be easily integrated to the SIPP productive project (Integrated Management System Drilling) of CEINPET. To achieve that objective, the development of a product that meets customer needs with quality and cost required, has been made a comprehensive study of all processes related to graphing columns of the CEINPET, and has been done an analysis of all the basis of software engineering necessary for the proposed system architecture to implement.

The different chapters of this investigation covers the main tools, technologies and methodologies to be used for system development, has been defined the baseline architecture that provides a model for describing the architecture through the views defined by RUP methodology (Rational Unified Process), also has been generated, as a result of this work, the Software Architecture Document of System for Graphing lithological Columns of Oil Wells.

Finally, as the main result of this investigation is obtained a robust architecture for the System for Graphing Lithological Columns of Oil Wells, allowing workers of CEINPET to do the process for graphing columns in a simple and efficient way, saving the company time and money investment and the use of inappropriated and proprietary software.

**Keywords:**

Architecture, Technologies, Methodologies, View, Base Line, Oil.

**INDICE**

**Índice de Figuras**



**Índice de Tablas**

## INTRODUCCIÓN

La Unión Cuba-Petróleo (CUPET) es la unión de empresas, responsable del desarrollo y mantenimiento de la industria del petróleo en Cuba. Junto con otras compañías extranjeras interesadas en este sector del desarrollo petrolífero en Cuba, estas empresas con sus distintas entidades distribuidas en todo el país intervienen en los diferentes procesos por los que debe pasar el crudo.

La industria del petróleo en Cuba, se dividen en tres grandes procesos: 1- Exploración-Producción, 2- Refinación y 3- Comercialización. Las entidades especializadas en Exploración-Producción, como su nombre lo indica, se encargan de la exploración y desarrollo de los campos, la perforación de nuevos pozos, así como la reparación de los que ya se encuentran en producción (Intervención de Pozos).

La DIPP (Dirección de Intervención y Perforación de Pozos), entidad única en el país, que dirige la perforación e intervención de pozos de petróleo radica en La Empresa de Producción y Extracción de Petróleo del CENTRO (EPEPC). El Centro de Investigaciones del Petróleo (CEINPET), es un centro único de su tipo en Cuba, el cual se encarga de dar respuesta de forma integral a toda la actividad petrolera, desde la Exploración hasta la Refinación, en el proceso de investigación – desarrollo – producción.

Dentro del Área de Exploración-Producción, se encuentra comprendido el proceso de Perforación. El CEINPET brinda a todos los pozos en perforación del país el servicio especializado de análisis geológico, realizado por una dotación de Ingenieros Geólogos, especialistas en este tema. Los geólogos registran diariamente las incidencias geológicas del pozo, analizando los cortes o sartas[1] de la perforación, registrando las características geológicas de la formación en el Reporte Diario de Geología, incluyendo en el mismo la columna litológica del pozo (Proyecto y Real). Este Reporte se le entrega por escrito al Supervisor del Pozo y se envía por correo al CEINPET.

En los pozos en perforación existe una dotación de supervisores de pozo, el supervisor de guardia confecciona diariamente el Reporte Operativo del Pozo, donde registra las incidencias (información) geológicas del día, el cual es enviado vía correo electrónico a la DIPP y al CEINPET. En el CEINPET se reciben el Reporte Diario de Geología y el Reporte Operativo de todos los pozos en Perforación y Ensayo; también se reciben por el CEINPET los Reportes de Intervención de los pozos en Intervención.

[1]Se refiere a Tuberías de acero que se unen para formar un tubo desde la barrena de perforación hasta la plataforma de perforación.

Con toda esta información se construye el Parte Diario de Geología, el cual es enviado a las altas esferas de CUPET y el Ministerio de la Industria Básica (MINBAS), debido a su gran importancia, ya que mantiene informado a la Unión CUPET de las incidencias geológicas y de la perforación de todos los pozos en Perforación, Ensayo e Intervención.

Actualmente no existe en el CEINPET ningún sistema que se encargue de graficar la información que envían los diferentes pozos de petróleos en perforación, como son las gráficas de las Columnas Litológicas. Este proceso se confecciona de manera manual por un técnico encargado de esta tarea y utilizando herramientas que no están concebidas para la gestión de este tipo de reportes; además que el flujo de información se realiza a través del correo, propiciando la introducción de errores en la información, tampoco se puede contar con la información actualizada de los pozos en un momento o lugar determinado, ni existe un orden para el almacenamiento de los datos, los cuales se almacenan en una única computadora.

En la Universidad de la Ciencias Informáticas, en la Facultad 9 existe el Polo de Soluciones Informáticas para la Industria del Petróleo (PetroSoft). Dadas la problemática anteriormente descrita, los líderes del CEINPET, optaron por la adquisición de un sistema de software para resolver la misma. Por este motivo, se está desarrollando en el polo el Sistema de Manejo Integral de Perforación de Pozos, el cual posee un Módulo (Subsistemas) que está concebido para automatizar el proceso de Graficar Columnas Litológicas de Pozos de Petróleo.

Este proyecto se encuentra en fase de construcción y posee una arquitectura definida para sus módulos desarrollados en plataforma web, por lo que se identifica la necesidad de una arquitectura de software del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo, lo que constituye el **problema a resolver**.

Luego de haber identificado el problema a resolver se define como **objetivo general** de la investigación: realizar una propuesta de arquitectura de software del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo y como **objetivos específicos**:

1. Evaluar los posibles estilos y patrones arquitectónicos que más se adecuan al proyecto productivo SIPP.
2. Generalizar los elementos encontrados, para poder obtener una propuesta de arquitectura del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo.

Para darle solución a este problema, se debe estudiar el proceso de Graficar Columnas Litológicas en el CEINPET a partir de la obtención de los datos geológicos necesarios, el cual es el **objeto de estudio** de la presente investigación, enmarcada en la arquitectura de software del proyecto productivo SIPP que es nuestro **campo de acción**. Una vez analizado el proceso de graficación de dichas columnas, la descripción de la arquitectura de software del Sistema para Graficar Columnas Litológicas de pozos de Petróleo, permitirá la implementación de un sistema que facilite a los trabajadores de CUPET graficar columnas litológicas a partir de la obtención de las lecturas de pozos, siendo esta la **idea a defender** de la presente investigación.

Con el objetivo de darle solución a las cuestiones relacionadas con la confección de las columnas litológicas, se han trazado un conjunto de tareas y orientaciones concretas que constituirán la guía para resolver los problemas tratados, en función de las necesidades prácticas y cognoscitivas existentes. Las mismas son:

1. Realizar el estado del arte relacionado con la Arquitectura de Software.
2. Identificar los requisitos no funcionales del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo.
3. Describir los componentes que integrarán el Sistema para Graficar Columnas Litológicas de Pozos de Petróleo, así como sus conexiones, configuraciones y restricciones.
4. Definir las herramientas y tecnologías a usar en la implementación del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo.
5. Generalizar los elementos identificados para diseñar la arquitectura de software del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo.

Para un mejor entendimiento de la problemática que surge ante la graficación de columnas litológicas de pozos de petróleo se decide estructurar esta investigación de la siguiente manera:

**Capítulo I**: En este capítulo se describe el estado del arte inicial, se especifican los aspectos más importantes relacionados con la arquitectura de software, además de resumir el estado del Arte, particionado en los temas fundamentales relacionados con la arquitectura de software y su aplicación en el proyecto productivo SIPP. También se definen los principales estilos, patrones, herramientas y tecnologías arquitectónicas utilizados en el desarrollo de software.

**Capítulo II**: En este capítulo se elabora el artefacto: Documento de arquitectura de software del Sistema para Graficar Columnas Litológicas de pozos de petróleo. Se definen: alcance del documento,

herramientas y tecnologías a emplear. Se fundamentan patrones y estilos arquitectónicos utilizados; además se describe detalladamente el modelo de las vistas arquitectónicas adaptadas al proyecto SIPP.

**Capítulo III**: En este capítulo se pone en práctica la propuesta arquitectónica, se documentan los resultados obtenidos durante la descripción y desarrollo de la arquitectura.

# CAPÍTULO 1 FUNDAMENTACIÓN TEÓRICA

## 1.1 Introducción

En este capítulo se describe el estado del arte inicial para esta arquitectura, especificando los aspectos más importantes relacionados con la arquitectura de software, como son las herramientas, tendencias y metodologías de desarrollo para poder enfatizar en cuales de estas serán usadas, resumiendo el estado del arte, particionado en los temas fundamentales relacionados con la arquitectura de software y su aplicación en el proyecto productivo SIPP.

## 1.2 Aspectos fundamentales de la Arquitectura del Software.

La Arquitectura del Software es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación, aporta una visión abstracta de alto nivel para un mejor entendimiento de un sistema y poder definir los módulos principales así como también las responsabilidades que tendrán los mismos, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño.
La arquitectura de software aporta elementos que ayudan a una mejor toma de decisiones, establece la línea base para que los analistas, diseñadores y programadores trabajen en conjunto permitiendo cubrir todas las necesidades. **(Casanovas, 2004)**

En los últimos años se ha presentando toda una sucesión de lenguajes para la descripción de la arquitectura del software. La mayoría de estos lenguajes de primera generación están aún en desarrollo o tienen carácter experimental pero son buenos ejemplos de búsqueda de las estructuras lingüísticas necesarias para describir las propiedades arquitectónicas de los sistemas de software.

## 1.3 Tendencias, Herramientas y Tecnologías

En este epígrafe se abordaran y analizarán las características principales de las metodologías, herramientas CASE, sistemas gestores de base de datos, lenguajes de programación y modelado, además de patrones y estilos arquitectónicos candidatos para la propuesta de una arquitectura de software del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo.

### 1.3.1 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 con la finalidad de garantizar la eficacia y la eficiencia en el proceso de generación de software. Los riesgos a afrontar y los controles a establecer varían en función de las diferentes etapas del ciclo de vida de desarrollo que defina cada metodología en particular. El núcleo de cualquier metodología de desarrollo se encuentra constituido por documentos escritos que detallan cada uno de los puntos expuestos. Existen metodologías ágiles y metodologías pesadas.

#### 1.3.1.1 Microsoft Solutions Framework (MSF)

Microsoft Solutions Framework 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. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

Esta metodología proporciona un sistema de modelos, principios, y pautas para dar soluciones a empresas que diseñan y desarrollan de una manera que se asegure de que todos los elementos de un proyecto, tales como personas, procesos y herramientas, puedan ser manejados con éxito. Además propone una secuencia generalizada de actividades para la construcción de soluciones empresariales.

El MSF combina los mejores principios del modelo en cascada y del modelo en espiral. Combina la claridad que planea el modelo en cascada y las ventajas de los puntos de transición del modelo en espiral.

El Modelo de proceso MSF consta de cinco fases distintas:

- Previsión
- Planeamiento
- Desarrollo
- Estabilización
- Implementación **(Sánchez, 2004)**

#### 1.3.1.2 Proceso Unificado de Desarrollo (RUP)

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, divide en 4 fases de desarrollo con sus respectivos hitos, estas fases son: Inicio, Elaboración, Construcción y Transición.

Cada una de estas etapas es desarrollada mediante un ciclo de iteraciones, que consisten en reproducir el ciclo de vida en cascada a menor escala. Los Objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.

RUP muestra como modelar software visualmente para capturar la estructura y comportamiento de arquitecturas y componentes. Las abstracciones visuales ayudan a comunicar diferentes aspectos del software; comprender los requerimientos, ver como los elementos del sistema se relacionan entre sí, mantener la consistencia entre diseño e implementación y promover una comunicación precisa. El estándar UML (Lenguaje de Modelado Unificado), creado por Rational Software, es el cimiento para un modelado visual exitoso. **(Sánchez, 2004)**

#### 1.3.1.3 Extreme Programming (XP)

Es una de las metodologías de desarrollo de software más exitosas en la actualidad utilizada para proyectos de corto plazo. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo al usuario final, lo que se convierte en un requisito para llegar al éxito del proyecto.

La metodología se basa en:

- **Pruebas Unitarias:** se basa en las pruebas realizadas a los principales procesos, de tal manera que se adelanten en algo hacia el futuro y se realicen pruebas de las fallas que pudieran ocurrir, para de esta manera obtener los posibles errores.
- **Re fabricación:** se basa en la reutilización de código, creando 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.

Extreme Programming propone comenzar en pequeño y añadir funcionalidad con retroalimentación continua. El manejo de cambios se convierte en una parte sustantiva del proceso. El costo del cambio no depende de la etapa o fase en la que se encuentre. No introduce funcionalidades antes que sean necesarias y el usuario o cliente se convierte en miembro del equipo.

Los factores que identifican a este tipo de metodología son:

- La comunicación, entre los usuarios y los desarrolladores.
- La simplicidad, al desarrollar y codificar los módulos del sistema.
- La retroalimentación concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales. **(Sánchez, 2004)**

Luego de haber analizado las carácterísticas de todas estas metodologías se llegó a la conclusión que la metodología a utilizar para la propuesta de arquitectura del Sistema Para Graficar Columnas Litológicas de pozos de petróleo será el Proceso Unificado de Desarrollo (RUP) debido a sus condiciones, como se explicará en capítulos posteriores.

### 1.4 Herramientas

¿Qué son las Herramientas CASE?

Se puede definir a las Herramientas CASE[2] como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software (Investigación Preliminar, Análisis, Diseño, Implementación e Instalación).

CASE es también definido como el Conjunto de métodos, utilidades y técnicas que facilitan el mejoramiento del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases.

Se puede ver al CASE como la unión de las herramientas automáticas de software y las metodologías de desarrollo de software formales. **(Valle, 2005)**

[2]CASE: (Computer-Aided Software Engineering). Ingeniería de Software Asistida por Ordenador.

### 1.4.1 Visual Paradigm

Esta herramienta está desarrollada por Visual Paradigm Internacional, una de las principales compañías de herramientas CASE. Su mayor éxito consiste en la capacidad de ejecutarse sobre diferentes sistemas operativos lo que le confiere la característica de ser multiplataforma. Visual Paradigm utiliza UML como lenguaje de modelado ofreciendo soluciones de software que permiten a las organizaciones desarrollar las aplicaciones de calidad más rápido, bien y más barato. Es muy fácil de usar y presenta un ambiente gráfico agradable para el usuario. Su notación es muy parecida a la estándar, permite configurar las líneas de redacción, el modelado de base de datos, el modelado de requerimientos, el modelado del proceso de negocio, la interoperabilidad, la generación de documentación y la generación de código base para diferentes lenguajes de programación como Java, C# y PHP. Igualmente permite la integración con herramientas de desarrollo (IDE´s). **(Visual-Paradigm, 1995)**

**Ventajas**

Visual Paradigm es una herramienta de calidad que soporta aplicaciones web, además brinda facilidades de instalación y actualización, es multilenguaje y compatible entre ediciones.

**Desventajas**

Su principal desventaja es la mala calidad de las imágenes y reportes generados, no permite su uso en proyectos comerciales e incluye marca de agua recordando este hecho. **(Luis Giraldo, 2005)**

### 1.4.2 Rational Rose

Rational Rose es una de las más poderosas herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos. Se utiliza para modelar un sistema antes de proceder a construirlo. Cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases. **(Rational Software Corporation, 2003)**

Rational Rose es la herramienta CASE que comercializan los desarrolladores de UML y que soporta de forma completa la especificación del UML 1.1. Esta herramienta propone la utilización de cuatro tipos de modelos para realizar un diseño del sistema, utilizando una vista estática, otra dinámica, una lógica y otra

física. Permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software. Rational Rose utiliza un proceso de desarrollo iterativo controlado, donde el desarrollo se lleva a cabo en una secuencia de iteraciones. Cada iteración comienza con una primera aproximación del análisis, diseño e implementación para identificar los riesgos del diseño, los cuales se utilizan para conducir la iteración, primero se identifican los riesgos y después se prueba la aplicación para que éstos se hagan mínimos. Cuando la implementación pasa todas las pruebas que se determinan en el proceso, ésta se revisa y se añaden los elementos modificados al modelo de análisis y diseño. Una vez que la actualización del modelo se ha modificado, se realiza la siguiente iteración. **(Lafuente, 2001)**

**Ventajas**

Es la herramienta CASE más usada para el modelado de sistemas con UML y UML2. Es muy completa y estable brindando facilidades de uso para la modificación y creación de nuevos diagramas. **(Iván Ayala Catari, 2007)**

**Desventajas**

Esta herramienta presenta un entorno gráfico no muy amigable para el usuario, tampoco es un software libre. Además no se puede crear el entorno del sistema para los diagramas de casos de usos. **(Iván Ayala Catari, 2007)**

### 1.4.3 MagicDraw UML

Esta es una herramienta CASE para hacer modelado de diagramas UML con soporte de distintos estándares de UML y generación de código para los lenguajes de programación Java, C#, C++. También es soportado por distintos entornos de desarrollo. Además cuenta con muchas extensiones que harán más fácil el desarrollo y darán la capacidad de soportar otras tecnologías.

La edición personal de MagicDraw o MagicDraw UML Personal Edition es un excelente programa para ser utilizado como modelado de UML visual y como herramienta CASE teniendo apoyo de trabajo en equipo.

El programa tiene como objetivo el complemento ideal para analistas de negocios de software y programadores.

Es un instrumento de desarrollo dinámico y que facilita el análisis y el diseño orientado a objetos de sistemas y bases de datos. **(MagicDraw, 2009)**

**Ventajas**

MagicDraw permite la navegación rápida a través de sus modelos, deriva modelos de código fuente existente en solo segundos. Además elimina la preparación de documentos con la generación de informe automático, y amplía capacidades de UML más allá de UML 2,0 en un broche de presión MagicDraw hace esto en minutos sin codificación adicional. **(MagicDraw, 2009)**

**Desventajas**

Su principal desventaja es la utilización de una máquina virtual de Java, precisamente por haber sido desarrollado en Java. Por lo que esto requeriría demasiados recursos de memoria que pondrían muy lento el accionar con la máquina. **(MagicDraw, 2009)**

Luego de hacer un análisis de todas estas herramientas CASE se llegó a la conclusión que la herramienta de modelado a utilizar para la propuesta de arquitectura del Sistema Para Graficar Columnas Litológicas de pozos de petróleo será Visual Paradigm debido a sus potentes funcionalidades, como se apreciará en capítulos posteriores.

## 1.5 Sistemas Gestores de Base Datos (SGBD ó DBMS)

Los Sistemas Gestores de Base de Datos son un tipo de software muy específico, dedicado a servir de interfaz entre la Base de Datos, el usuario y las aplicaciones que la utilizan. Se componen de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. **(Valdés, 2007)**

### 1.5.1 SQL Server

Microsoft SQL SERVER es un sistema de gestión de bases de datos relacionales (SGBD) basado en el lenguaje Transact-SQL, y específicamente en SybaseIQ, capaz de poner a disposición de muchos usuarios grandes cantidades de datos de manera simultánea, así como de tener muchas otras ventajas. Microsoft SQL Server constituye la alternativa de Microsoft a otros potentes Sistemas Gestores de Bases de Datos. **(Gallardo, 2009)**

Potente y Escalable: Microsoft SQL Server ofrece registros espectaculares. Posee una plataforma de desarrollo fácil y abierta: integrada con las mejores tecnologías de Internet como ActiveX, ADC y Microsoft Transaction Server y con las mejores herramientas de gestión y desarrollo para Internet como FrontPage97, Microsoft Office97 y Visual Interdev. Es el único gestor de base de datos que contiene de forma integrada la posibilidad de generar contenido HTML de forma automática. Tiene un mínimo coste de propiedad; la sencillez de la instalación, y la potencia de sus herramientas de gestión y el menor coste de toda la industria para entornos Internet, hacen de Microsoft SQL Server la mejor opción con el menor costo. **(Valera)**

### 1.5.2 Oracle

Oracle es una herramienta cliente/servidor para la gestión de Base de Datos relacional. Esta herramienta hace uso de los recursos del sistema informático en todas las arquitecturas de hardware para garantizar su aprovechamiento al máximo en ambientes cargados de información. Con nuevas funciones de autogestión, Oracle elimina las tareas administrativas lentas y propensas a errores, lo que permite a los administradores de las BD concentrarse en los objetivos estratégicos de la empresa en lugar de prevenir problemas de rendimiento y disponibilidad. **(ORACLE Packs de Gestión de Base de Datos de Oracle)**

**Ventajas**

Este SGBD puede ejecutarse en todas las plataformas y soporta todas las funciones de cualquier servidor. Posee un lenguaje de diseño de bases de datos muy completo (PL/SQL) que permite implementar diseños activos, triggers y procedimientos almacenados, con una integridad referencial declarativa bastante potente. Permite el uso de particiones para la mejora de la eficiencia, de replicación. Este sistema ha añadido tipos de clases, referencias, tablas anidadas, matrices y otras estructuras de

datos complejos.

**Desventajas**

Uno de los mayores inconvenientes de Oracle es su alto precio, las licencias de Personal Oracle son excesivamente caras, así como su aprendizaje ya que hay pocos libros sobre asuntos técnicos distintos de la simple instalación y administración. Puede presentar dificultades de configuración, cuando está mal configurado se torna demasiado lento. La implementación de estructuras de datos complejos provoca la incompatibilidad de los diseños.

### 1.5.3 PostgreSQL

El SGBD relacional orientado a objetos conocido como PostgreSQL está derivado del paquete Postgres escrito en Berkeley, distribuida bajo licencia BSD. PostgreSQL es un gestor de bases de datos de código abierto, que ofrece control de concurrencia multiversión, soportando casi toda la sintaxis SQL (incluyendo subconsultas, transacciones, tipos y funciones definidas por el usuario), contando también con un amplio conjunto de enlaces con lenguajes de programación como pueden ser C, C++, Java, Perl, TCL y Python. Al mismo tiempo, ofrece una potencia adicional sustancial al incorporar los siguientes cuatro conceptos adicionales básicos en una vía en la que los usuarios pueden extender fácilmente el sistema: clases, herencia, tipos y funciones.

**Ventajas**

PostgreSQL ofrece muchas ventajas respecto a otros sistemas de bases de datos: tiene una instalación ilimitada, no hay costo asociado a la licencia de software, por lo que es flexible para investigaciones y desarrollo sin necesidad de costos adicionales de licenciamiento. Postgres cuenta con una comunidad de profesionales que les brindan beneficios de soporte. Ha sido diseñado y creado para tener un mantenimiento y ajuste mucho menor que los productos de los proveedores comerciales, conservando todas las características, estabilidad y rendimiento. Posee estabilidad y confiabilidad legendaria, además su código fuente está disponible para todos sin costo (gratis). Es multiplataforma y diseñado para ambientes de alto volumen. Posee herramientas gráficas de diseño y administración de bases de datos.

## 1.6 Lenguajes de programación

Un lenguaje de programación es un lenguaje artificial usado para controlar el comportamiento de una máquina, especialmente una computadora. Se componen de un conjunto de reglas sintácticas y semánticas que permiten expresar instrucciones que luego serán interpretadas por máquinas.

Existen muchos lenguajes de programación, para desarrollar aplicaciones tanto de escritorio como web, sin embargo para este trabajo de diploma fueron tomados en consideración algunos de ellos: C++, Java, XML, y C#.

### 1.6.1 C++

Aunque en un principio C++ se plantea como una mejora de C ('C con clases'), en la actualidad es un lenguaje independiente, versátil, potente y general. Mantiene las ventajas de C en cuanto a riquezas de operadores y expresiones, flexibilidad, concisión y eficiencia. Además ha eliminado algunas de las dificultades y limitaciones del C original. Este es a la vez un lenguaje procedural (orientado a algoritmos) y orientado a objetos. Como lenguaje procedural se asemeja al C y es compatible con él. Como lenguaje orientado a objeto se basa en una filosofía completamente diferente, que exige del programador un completo cambio de mentalidad. Las características propias de la programación orientada a objetos (Object Oriented Programming, u OOP) de C++ son modificaciones mayores que si cambian radicalmente su naturaleza. **(Paul Bustamante, 2004)**

### 1.6.2 C#

C Sharp es un lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg. Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, y programarla usando C# es muy sencillo e intuitivo, ya que carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el lenguaje nativo de .NET. La sintaxis y estructuración de C# es muy similar al C++, ya que la intención de Microsoft es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo. **(Seco, 2001)**

### 1.6.3 Java

Java es toda una tecnología orientada al desarrollo de software con el cual podemos realizar cualquier tipo de programa. La tecnología Java ha cobrado mucha importancia en el ámbito de Internet gracias a su plataforma Java 2 Enterprise Edition (J2EE). Está desarrollado por la compañía Sun Microsystems con gran dedicación y siempre enfocado a cubrir las necesidades tecnológicas más punteras.

El lenguaje es parecido a C y C++, aunque su modelo de objetos es más sencillo, y fue influenciado también por Smalltalk y Eiffel. Incorpora sincronización y manejo de tareas en el lenguaje mismo (similar a Ada) e incorpora interfaces como un mecanismo alternativo a la herencia múltiple de C++.

Los programas en Java generalmente son compilados a un lenguaje intermedio llamado bytecode, que luego son interpretados por una máquina virtual (JVM). Esta última sirve como una plataforma de abstracción entre la máquina y el lenguaje.

Java es un lenguaje orientado a objetos: esto es lo que facilita abordar la resolución de cualquier tipo de problema. Es sencillo, aunque sin duda potente, su ejecución es segura y fiable. Los programas no acceden directamente a la memoria del ordenador, siendo imposible que un programa escrito en Java pueda acceder a los recursos del ordenador sin que esta operación le sea permitida de forma explícita. De este modo, los datos del usuario quedan a salvo de la existencia de virus escritos en Java. La ejecución segura y controlada del código Java es una característica única, que no puede encontrarse en ninguna otra tecnología. Es totalmente multiplataforma: el entorno necesario para su ejecución es de pequeño tamaño y puede adaptarse incluso al interior de un navegador. **(Gastón, 2007)**

### 1.6.4 Lenguaje Extensible de Marcas (XML)

Lenguaje de marcas extensible *(*XML) es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). El lenguaje XML es un lenguaje de marcas, capaz de describir cualquier tipo de información en forma personalizada, aunque también es un metalenguaje de mercado capaz de describir lenguajes de marcas adecuadas para aplicaciones concretas. **(Ayala)**

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Es una tecnología sencilla que tiene a

su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil. XML es un lenguaje extensible lo que quiere decir que una vez diseñado un lenguaje y puesto en producción, igual es posible extenderlo con la adición de nuevas etiquetas de manera de que los antiguos consumidores de la vieja versión todavía puedan entender el nuevo formato. El analizador es un componente estándar, no es necesario crear un analizador específico para cada lenguaje. Esto posibilita el empleo de uno de los tantos disponibles. De esta manera se evitan bugs y se acelera el desarrollo de la aplicación. Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarlo. Mejora la compatibilidad entre aplicaciones. La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. **(Rosas García)**

Luego de analizar estos lenguajes de programación candidatos para la propuesta de arquitectura del Sistema Para Graficar Columnas Litológicas de pozos de petróleo, se escogió C++ debido a sus potentes funcionalidades, como se apreciará en capítulos posteriores.

## 1.7 Estilos y Patrones

### 1.7.1 Patrones

“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”.
Christopher Alexander

Un patrón se puede definir como una pareja de problema/solución con un nombre. Los patrones son utilizados en el campo de la arquitectura en los años 70 por Christopher Alexander que propuso el aprendizaje y uso de patrones para la construcción de edificios. El estudio y establecimiento de los patrones hoy en día posibilita el aprovechamiento de la experiencia acumulada en el diseño y desarrollo de aplicaciones, permitiendo obtener de una forma sencilla la arquitectura.

Un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores y otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas en distintas circunstancias.

Los patrones pueden dividirse y clasificarse de acuerdo con el nivel de abstracción de las vistas de un sistema, en: patrones de arquitectura, patrones de diseño, patrones de idiomas. **(Jara, 2008)**

### 1.7.2 Patrones de Arquitectura

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. **(Oktaba, 2007)**

Los patrones de arquitectura expresan una organización estructural para un sistema de software. Proveen un conjunto de subsistemas predefinidos e incluyen reglas y lineamientos para conectarlos.

Generalmente los patrones de arquitecturas responden a las preguntas: ¿Cuáles son los subsistemas o componentes del sistema?, ¿cómo deben ser estructurados las interfaces entre componentes y cuáles son las características de comunicación de los componentes?

Algunos de los patrones de Arquitecturas son:

Layers: Descompone una aplicación en un conjunto de capas independientes y ordenadas jerárquicamente. Cada nivel o capa usa los servicios de la capa inmediatamente inferior y ofrece servicios a la capa inmediatamente superior.

Broker: este patrón se usa para organizar sistemas distribuidos con componentes débilmente acoplados que interactúan entre sí invocando servicios remotos. **(Universidad EAFIT, 2005)**

### 1.7.3 Patrones de Diseño

Los patrones de diseño son soluciones simples y elegantes a problemas específicos comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia, que se ha demostrado que funcionan.

Los patrones de diseño no son fáciles de entender, pero una vez entendido su funcionamiento, los diseños serán mucho más flexibles, modulares y reutilizables. Han revolucionado el diseño orientado a objetos y todo buen arquitecto de software debería conocerlos. **(García, 2005)**

Los patrones de diseño más habituales son: Patrones GoF y Patrones GRASP

#### 1.7.3.1 Patrones GoF

Patrones de creación

- Abstract Factory: Proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.
- Builder: Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.
- Factory Method: Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos.
- Prototype: Especifica los tipos de objetos a crear por medio de una instancia prototípica, y crear nuevos objetos copiando este prototipo.
- Singleton: Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella. **(García, 2005)**

Patrones estructurales

- Adapter: Convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permiten que cooperen clases que de otra manera no podrían por tener interfaces incompatibles.
- Bridge: Desvincula una abstracción de su implementación, de manera que ambas puedan variar de forma independiente.
- Composite: Combina objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.
- Decorator: Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.
- Facade: Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar.
- Flyweight: Usa el compartimiento para permitir un gran número de objetos de grano fino de forma eficiente.

- Proxy: Proporciona un sustituto o representante de otro objeto para controlar el acceso a éste. **(García, 2005)**

Patrones de comportamiento

- Chain of Responsibility: Evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición. Crea una cadena con los objetos receptores y pasa la petición a través de la cadena hasta que esta sea tratada por algún objeto.
- Command: Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer las operaciones.
- Interpreter: Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje.
- Iterator: Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.
- Mediator: Define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente.
- Memento: Representa y externaliza el estado interno de un objeto sin violar la encapsulación, de forma que éste puede volver a dicho estado más tarde.
- Observer: Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos.
- State: Permite que un objeto modifique su comportamiento cada vez que cambia su estado interno. Parecerá que cambia la clase del objeto.
- Strategy: Define una familia de algoritmos, encapsula uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan.
- TemplateMethod: Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura.
- Visitor: Representa una operación sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera. **(García, 2005)**

### 1.7.3.2 Patrones de Asignación de Responsabilidades (GRASP)

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

Los primeros cinco patrones GRASP:

**Bajo Acoplamiento**

Debe haber pocas dependencias entre las clases. Si todas las clases dependen de todas ¿cuánto software podemos extraer de un modo independiente y reutilizarlo en otro proyecto? para determinar el nivel de acoplamiento de clases, son muy buenos los diagramas de colaboración de UML. Uno de los principales síntomas de un mal diseño y alto acoplamiento es una herencia muy profunda. Siempre hay que considerar las ventajas de la delegación respecto de la herencia**. (Gutiérrez, 2007)**

**Experto**

La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. Hay que tener en cuenta que esto es aplicable mientras estemos considerando los mismos aspectos del sistema:

- Lógica de negocio.
- Persistencia a la base de datos.
- Interfaz de usuario. **(Gutiérrez, 2007)**

**Alta Cohesión**

Cada elemento de nuestro diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable.

Ejemplos de una baja cohesión son clases que hacen demasiadas cosas. En todas las metodologías se considera la refactorización. Uno de los elementos a refactorizar son las clases saturadas de métodos. Ejemplos de buen diseño se producen cuando se crean los denominados “paquetes de servicio” o clases agrupadas por funcionalidades que son fácilmente reutilizables (bien por uso directo o por herencia). **(Gutiérrez, 2007)**

**Creador**

La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización. **(Gutiérrez, 2007)**

**Controlador**

Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones y seguridad). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. Un error muy común es asignarle demasiada responsabilidad y alto nivel de acoplamiento con el resto de los componentes del sistema. **(Gutiérrez, 2007)**

### 1.8 Patrones de Idioma.

Los patrones de idioma se utilizan en los flujos de implementación, mantenimiento y despliegue. Estos patrones son generalmente reconocidos como estándares de codificación ya que describen como codificar y representar operaciones comunes bien conocidas en un nuevo ambiente, o a través de un grupo, brindando legibilidad y predictibilidad. Son sumamente específicos de un lenguaje, plataforma o ambiente. **(Kiccillof, 2004)**

## 1.9 Estilos Arquitectónicos

Se define estilo arquitectónico como una familia de sistemas de software en términos de su organización estructural. Expresa componentes y las relaciones entre estos, con las restricciones de su aplicación y la composición asociada, así como también las reglas para su construcción. **(Frank Buschmann, 1996)**

Los estilos son útiles para sintetizar estructuras de soluciones, también definen los patrones posibles de las aplicaciones y permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales.

Algunos de los estilos arquitectónicos más conocidos son:

### 1.9.1 Estilos de Flujo de Datos

- Tubería y filtros

Una tubería (*pipeline*) es una popular arquitectura que conecta componentes computacionales (filtros) a través de conectores (*pipes*), de modo que las computaciones se ejecutan a la manera de un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. Debido a su simplicidad y su facilidad para captar una funcionalidad, es una arquitectura mascota cada vez que se trata de demostrar ideas sobre la formalización del espacio de diseño arquitectónico, igual que el tipo de datos stack lo fue en las especificaciones algebraicas o en los tipos de datos abstractos. **(Kiccillof, 2004)**

### 1.9.2 Estilos Centrados en Datos

- Arquitecturas de Pizarra o Repositorio

En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. En base a esta distinción se han definidos dos subcategorías principales del estilo:

Si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, el repositorio puede ser una base de datos tradicional (implícitamente no cliente-servidor).
Si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control. **(Kiccillof, 2004)**

### 1.9.3 Estilos de Llamada y Retorno

- Modelo Vista Controlador(MVC)

Un modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador). Mantiene el conocimiento del sistema. No depende de ninguna vista y de ningún controlador.

La vista maneja la visualización de la información.

El controlador interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado e interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado. **(Nicolás Kiccillof, 2004)**

- Arquitecturas en Capas

El estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. El estilo admite muy naturalmente optimizaciones y refinamientos. Proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas. **(Nicolás Kiccillof, 2004)**

- Arquitecturas Orientadas a Objetos

Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos. En la caracterización clásica de David Garlan y Mary Shaw, los objetos representan una

clase de componentes que ellos llaman managers, debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos. Los componentes del estilo se basan en principios Orientados a Objetos: encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación.

- Arquitecturas Basadas en Componentes

Los componentes en este estilo son en el sentido de Componente basado en Ingeniería de Software (CBSE) y son las unidades de modelado, diseño e implementación. Las interfaces están separadas de las implementaciones y sus interacciones son el centro de incumbencias en el diseño arquitectónico. Los componentes soportan algún régimen de introspección, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución. **(Nicolás Kiccillof, 2004)**

### 1.9.4 Estilos de Código Móvil

- Arquitectura de Máquinas Virtuales

Este estilo se caracteriza por su enorme portabilidad. Las arquitecturas que siguen este estilo tienen una parte del sistema que pertenece al propio entorno nativo de la máquina que lo incluye, la otra parte no. Para realizar una comunicación entre ambas partes se necesita de un intérprete que permita la traducción. Este intérprete es un conector con datos que contiene las instrucciones en el lenguaje no nativo de la máquina y una información de estado de esta parte no nativa. Por tanto, la principal preocupación que hay que tener en cuenta a la hora de diseñar una arquitectura perteneciente a esta familia es como llevar a cabo esta traducción y realizar la integración de la parte del sistema no incluida en la máquina física. **(Juan Pablo López, 2007)**

### 1.9.5 Arquitectura basada en objetos

- Estilos de Llamada/Retorno.
- Arquitecturas basadas en objetos.

- Arquitecturas basadas en mensajes/recursos.
- Arquitecturas basadas en servicios.

### 1.9.6 Estilos heterogéneos

Los estilos naturalmente heterogéneos se clasifican en:

- Sistemas de control de procesos

Desde el punto de vista arquitectónico, mientras casi todos los demás estilos se pueden definir en función de componentes y conectores, los sistemas de control de procesos se caracterizan no sólo por los tipos de componentes, sino por las relaciones que mantienen entre ellos. El objetivo de un sistema de esta clase es mantener ciertos valores dentro de ciertos rangos especificados llamados puntos fijos o valores de calibración.
Shaw y Garlan recomiendan separar los tres elementos del bucle de control (mecanismos para cambiar los valores de variables y algoritmos de control, elementos de datos; esquema del bucle). La ventaja señalada para este estilo radica en su elasticidad ante perturbaciones externas. **(Kiccillof, 2004)**

- Arquitecturas Basadas en Atributos

La arquitectura basada en atributos o ABAS fue propuesta por Klein y Klazman. La intención de estos autores es asociar a la definición del estilo arquitectónico un sistema (framework) de razonamiento (ya sea cuantitativo o cualitativo) basado en modelos de atributos específicos. Su objetivo se funda en la premisa que dicha asociación proporciona las bases para crear una disciplina de diseño arquitectónico, tornando el diseño en un proceso predecible, antes que en una metodología. Con ello se lograría que la arquitectura de software estuviera más cerca de ser una disciplina de ingeniería.
Aportando el beneficio esencial de la ingeniería al diseño arquitectónico.
El modelo de Klein y Kazman en realidad no tipifica como un estilo en estado puro, sino como una asociación entre la idea de estilo con análisis arquitectónico y atributos de calidad. **(Kiccillof, 2004)**

### 1.9.7 Estilos Peer-To-Peer

Los estilos Peer-To-Peer consisten por lo general en una serie de procesos independientes o entidades que se comunican a través de mensajes. Las entidades se pueden enviar mensajes entre sí, pero no se pueden controlar directamente.

- Arquitecturas Basadas en Eventos

Las arquitecturas basadas en eventos se han llamado también de invocación implícita. Los componentes de un estilo de invocación implícita son módulos cuyas interfaces proporcionan tanto una colección de procedimientos (igual que en el estilo de tipos de datos abstractos) como un conjunto de eventos. Los procedimientos se pueden invocar a la manera usual en modelos orientados a objeto, o mediante el sistema de suscripción que se ha descrito.
El estilo se utiliza en ambientes de integración de herramientas, en sistemas de gestión de base de datos para asegurar las restricciones de consistencia (bajo la forma de disparadores). **(Chung, 2002)**

- Arquitecturas Orientadas a Servicios
- Arquitecturas Basadas en Recursos

### 1.9.8 Estilos derivados

- **C2 ó Chiron-2**

Este estilo está constituido por componentes que se comunican a través de buses; la comunicación está basada en eventos. Un componente puede enviar o recibir eventos hacia o desde los buses a los que está conectado. Componentes y buses se pueden componer topológicamente de distintas maneras, siguiendo reglas y restricciones particulares. Cada componente posee dos puntos de conexión, llamados respectivamente top y botón. El esquema no admite ciclos, de modo que un componente no puede recibir una notificación generada por él mismo. **(Proceedings of the 21st International Conference on Software Engineering (ICSE'99), Los Ángeles, 1999)**

- **REST o Estado Representacional de Transferencia**

Rest define recursos identificables y métodos para acceder y manipular el estado de esos recursos. El caso de referencia es nada menos que la World Wide Web, donde los URIs identifican los recursos y HTTP[3] es el protocolo de acceso. El argumento central de Fielding es que HTTP mismo, con su conjunto mínimo de métodos y su semántica simplísima, es suficientemente general para modelar cualquier dominio de aplicación. REST es en parte una reseña de una arquitectura existente y en parte un proyecto para un estilo nuevo. La caracterización de REST constituye una lectura creativa de la lógica dinámica que rige el funcionamiento de la Web (una especie de ingeniería inversa de muy alto nivel), al lado de una propuesta de nuevos rasgos y optimizaciones, o restricciones adicionales. REST se construye expresamente como una articulación compuesta a partir de estilos y sub-estilos preexistentes, con el agregado de restricciones específicas. **(Principled design of the modern Web architecture, 2002)**

Como se puede apreciar hay varios patrones y estilos arquitectónicos usados según sus propias características para resolver un problema u otro, se han seleccionado un grupo de ellos para el desarrollo del Sistema para Graficar Columnas Litológicas de pozos de petróleo los cuales se abordarán en el próximo capítulo de este documento.

## 1.10 Conclusiones Parciales

En el capítulo que concluye se profundizó en las características fundamentales de las tecnologías, metodologías, lenguajes de programación, sistemas gestores de bases de datos, además de estilos y patrones arquitectónicos candidatos para la propuesta de una arquitectura de software del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo. El estudio realizado en este capítulo de las principales tendencias, herramientas y tecnologías usadas en la Ingeniería de software permitirán elegir de forma correcta las que son idóneas para el cumplimiento de los objetivos trazados.

[3]HTTP: HyperText Transfer Protocol, o Protocolo de Transferencia de Hipertexto.

## CAPÍTULO 2 TECNOLOGÍAS A UTILIZAR

### 2.1 Introducción

En este capítulo se abordaran los aspectos relacionados con la arquitectura de software propuesta para el Sistema de Graficar Columnas Litológicas de Pozos de Petróleo además de la descripción, ventajas y desventajas de cada una de las herramientas, tecnologías, lenguaje de programación, metodología de desarrollo, estilos y patrones a emplear para una mejor elaboración del software con el objetivo de obtener un producto con la calidad requerida.

### 2.2 Estilo de Arquitectura Definido: Arquitectura en Capas.

Se llegó a la conclusión de que el estilo arquitectónico que más se adapta a la solución de arquitectura es la Arquitectura en Capas, ya que la utilización de dichas capas proporciona al sistema niveles de abstracción que resuelven muchos problemas de desarrollo, proporcionando reutilización, optimización y refinamiento.

Este estilo asegura que los componentes se estructuren en niveles o capas donde cada nivel invoca sólo al nivel inferior y las interfaces entre capas están claramente definidas. Es definido como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Su objetivo principal es separar la lógica del negocio de la del diseño. **(Mary Garlan, 1994)**

Se utilizará para la solución arquitectónica del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo arquitectura en tres capas, que divide la aplicación en tres capas lógicas distintas:

- Capa de Presentación: esta capa se encarga de gestionar todos aquellos aspectos relacionados con la lógica de presentación de la aplicación como son las validaciones de la información y la gestión de usuarios, pues es la capa que interactúa con el mismo. Se comunica con la capa de Negocio.
- Capa de Lógica del Negocio o Dominio: Se encarga de recibir las peticiones de los usuarios y contiene todo el código que define las reglas de negocio para dar respuestas a las mismas luego de su proceso. Se puede definir como el conjunto de reglas de negocio que abstraen el problema real a tratar.

- Capa de acceso a datos: Contiene los datos necesarios para la aplicación y también puede ofrecer servicios relacionados con la persistencia o recuperación de información, en este caso del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo se encargará de toda la gestión de ficheros y proyectos. En dichos ficheros se realizarán las salvas de las informaciones necesarias para la construcción y visualización de las columnas litológicas. La gestión de datos se realizará mediante la serialización de objetos que contendrán las informaciones necesarias del sistema, garantizando así además la seguridad e integridad de los datos.

  **Ventajas**

  - Reutilización y desarrollos paralelos en cada capa.
  - Facilita la estandarización.
  - Soporta un diseño basado en niveles de abstracción crecientes.
  - Contención de cambios a una o pocas capas.
  - Flexibilidad y alta escalabilidad.

  **Desventajas**

  - A veces no se logra la contención del cambio y se requiere una cascada de cambios en varias capas.
  - Dificultad de diseñar correctamente la granularidad de las capas. **(Galeon, 2006)**

## 2.3 Framework[4] Definido: QT

Qt es un framework de desarrollo para aplicaciones multiplataforma que simplifica mucho el desarrollo de aplicaciones en C++ de forma nativa, también puede ser utilizado en otros lenguajes, funciona en las principales plataformas y tiene un amplio apoyo. Es una biblioteca para desarrollar interfaces gráficas de usuario y también para el desarrollo de programas sin interfaz gráfica como herramientas de la consola y servidores. Este framework está compuesto por librerías Qt (clases en C++), además se pueden crear formularios visualmente con Qt Designer y presenta un acceso rápido a la documentación a través de Qt

[4] Framework: en el desarrollo de software es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado.

Assistant. QT permite una traducción rápida con su Qt Linguist y simplifica el proceso de construcción de proyectos en las diferentes plataformas soportadas. La Interfaz de Programación de Aplicaciones(API) de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML; cuenta con una API multiplataforma unificada para la manipulación de archivos y otras para el manejo de ficheros, además de estructuras de datos tradicionales. A continuación otras de las características más importantes de este framework que ayudan a la selección del mismo:

- Compatibilidad multiplataforma con un sólo código fuente.
- Rendimiento en C++.
- Disponibilidad del código fuente.
- Excelente documentación.
- Fácilmente internacionalizable.
- Arquitectura lista para plugins.

## 2.4 Patrones Arquitectónicos Seleccionados

Se han seleccionado varios patrones arquitectónicos que definen la estructura del sistema de software a desarrollar, los cuales a su vez se componen de subsistemas con sus responsabilidades, y que tienen una serie de directivas para organizar los componentes con el objetivo de facilitar la tarea del diseño del sistema. Estos patrones garantizarán una mayor comprensión de la arquitectura del sistema y agilizarán el posterior desarrollo e implementación del mismo.

**Facade**

Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar. Para realizar una mejor estructuración de las estructuras de datos necesarias para la implementación del proyecto se utilizará este patrón, en aras de facilitar la utilización de las mismas.

**Singleton**

Se empleará este patrón de diseño GOF ya que garantiza que exista una instancia única para una clase y proporciona un punto de acceso global a ella. A partir de esta instancia se podrán controlar más fácilmente el acceso a clases que tienen responsabilidades bien definidas como la de acceso a ficheros.

**Comando**

Este patrón de diseño GOF permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor real de la misma, lo que será muy importante para la implementación.

**Creador**

El patrón creador es un patrón GRASP que guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se conecte con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento.

**Controlador**

El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado.

**Adaptador**

Este patrón es el encargado de adaptar una interfaz para que pueda ser utilizada que de otro modo no podría utilizarse.

**Puente**

El patrón puente se encarga de desacoplar una abstracción de la implementación. Mediante este patrón se realizarán las implementaciones para lograr el acceso a las interfaces ofrecidas en el sistema.

**Decorador**

El patrón decorador es el encargado de añadir funcionalidades a las clases de forma dinámica. Mediante este patrón se pueden definir un conjunto de funcionalidades que pueden acompañar dinámicamente a otras funcionalidades, brindando la posibilidad de no tener que redefinir estas funcionalidades cada vez que se necesite utilizarlas.

## 2.5 Lenguaje de Modelado seleccionado: UML

El Lenguaje Unificado de Modelado (UML) es un lenguaje para la especificación, visualización, construcción y documentación de los artefactos de sistemas de software, que da la posibilidad a los desarrolladores de visualizar los resultados de su trabajo en esquemas o diagramas estandarizados, proporcionándoles un vocabulario que incluye tres categorías: elementos, relaciones y diagramas. También, UML suministra mecanismos de extensibilidad y permite la modelación de sistemas con tecnología orientada a objetos. Es un lenguaje gráfico con sintaxis y semántica bien definidas.

Este lenguaje reúne las mejores prácticas de ingeniería que han sido probadas con éxito en el modelado de sistemas grandes y complejos.

La representación en UML de un software está formada por las 4+1 vistas de la arquitectura, relacionados entre sí:

- Vista de casos de uso.
- Vista lógica.
- Vista de procesos.
- Vista de implementación.
- Vista de despliegue.

**Ventajas**

UML se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware, y organizaciones del mundo real.

UML ofrece nueve diagramas para modelar sistemas, los cuales se mencionan a continuación:

- Diagramas de casos de usos para modelar los procesos del negocio.
- Diagramas de secuencia para modelar el paso de mensajes entre objetos.
- Diagrama de colaboración para modelar interacciones entre objetos.
- Diagramas de estado para modelar el comportamiento de los objetos en el sistema.
- Diagramas de actividades para modelar el comportamiento de los casos de uso, objetos u operaciones.
- Diagramas de clases para modelar la estructura estática de las clases en el sistema.
- Diagramas de objetos para modelar la estructura estática de los objetos en el sistema.
- Diagramas de componentes para modelar componentes.

- Diagramas de implementación para modelar la distribución del sistema.

UML es la consolidación de muchas de las notaciones y conceptos más usados orientados a objetos. **(Giron, 2007)**

### 2.6 Herramienta CASE Seleccionada: Visual Paradigm

Visual Paradigm es la herramienta CASE escogida para el desarrollo de esta arquitectura de software. Se seleccionó esta herramienta debido a que posee licencia gratuita y comercial, es un producto de mucha calidad, multiidioma, multilenguaje, muy fácil de instalar y actualizar además de bridar compatibilidad entre ediciones. Esta herramienta soporta modelado UML y provee el modelado de procesos de negocios, además de un generador de mapeo de objetos relacionales para los lenguajes de programación Java.NET y PHP. Visual Paradigm UML responde rápidamente y con poca memoria, permitiendo manejar grandes y complicadas estructuras de un proyecto de forma eficiente y que solo requiera de una configuración de escritorio.

Visual Paradigm presenta varias ediciones, cada una destinada a sus propias necesidades: Enterprise, Professional, Community, Standard, Modeler y Personal. También ayuda a los equipos de desarrollo de software a desplegar el proceso de desarrollo de los mismos, logrando maximizar y acelerar tanto las contribuciones individuales como las de equipo, facilita la diagramación visual y el diseño de sus proyectos, además posee alta interoperabilidad.

**Ventajas:**

- Navegación intuitiva entre código y el modelo.
- Poderoso generador de documentación y reportes UML PDF/HTML/MS Word.
- Demanda en tiempo real, modelo incremental de viaje redondo y sincronización de código fuente.
- Superior entorno de modelado visual.
- Soporte completo de notaciones UML.
- Diagramas de diseño automático sofisticado.

- Análisis de texto y soporte de tarjeta CRC.
- Lenguajes en generación de Código e Ingeniería Inversa.
- Creación de modelos UML.
- Modelado de base de datos.
- Mapa de relación de objetos.
- Interoperabilidad.
- Integración IDE.

## 2.7 Lenguaje de Programación Seleccionado: C++

El lenguaje de programación escogido fue C++, con el cual se puede llevar a cabo el desarrollo de cualquier tipo de programa y que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos. Además C++ brinda muchas facilidades para la programación orientada a objetos y para el uso de plantillas o programación genérica. Está considerado como un potente lenguaje, ya que trabaja tanto a alto como a bajo nivel por lo que el código resultante de su compilación es muy eficiente. Además posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel como son la identificación de tipos en tiempo de ejecución (RTTI) y la posibilidad de redefinir operadores.

La posibilidad de orientar la programación a objetos que nos ofrece este lenguaje permite al programador diseñar aplicaciones desde un punto de vista más cercano a la vida real. Además permite la reutilización del código de una manera más lógica y productiva. También es de alta portabilidad y brevedad.

Otra importante característica de C++ es la programación modular ya que una entidad de aplicación en C++ puede estar hecha con varios ficheros de código fuente que son compilados por separado y después unidos. Esta característica permite unir código en C++ con código producido en otros lenguajes.

**Ventajas**

Las principales ventajas que presenta el lenguaje C++ son:

- Difusión: al ser uno de los lenguajes más empleados en la actualidad, posee un gran número de usuarios y existe una gran cantidad de libros, cursos, páginas web, dedicados a él.
- Versatilidad: es un lenguaje de propósito general, por lo que se puede emplear para cualquier tipo de problema.
- Portabilidad: el lenguaje está estandarizado y un mismo código fuente se puede compilar en diversas plataformas.
- Eficiencia; es uno de los lenguajes más rápidos en cuanto a ejecución.
- Herramientas: existen una gran cantidad de compiladores, depuradores, librerías entre otros. **(Mora, 2006)**

## 2.8 Metodología de Desarrollo Definida: Proceso Unificado de Desarrollo (RUP)

El Proceso Unificado de Desarrollo (RUP) es una metodología de desarrollo para proyectos grandes, lo que no quiere decir que no sea también usada en proyectos pequeños, por esto se le denomina una metodología pesada. El ciclo de vida de RUP se divide en cuatro fases de desarrollo (Inicio o Concepción, Elaboración, Construcción, Transición) cada una con sus respectivos hitos, cada fase está compuesta por iteraciones las cuales se divide a su vez en una serie de disciplinas que recuerdan a las definidas en el ciclo de vida clásico o en cascada, mostrando como modelar software visualmente capturando la estructura de comportamientos de arquitecturas y componentes.

Esta metodología utiliza el Lenguaje Unificado de Modelado (Unified Modeling Language, UML) para preparar todos los esquemas de un sistema de software.

RUP se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y ser iterativo e incremental. Realiza un levantamiento exhaustivo de requerimientos, busca detectar defectos en las fases iníciales con el fin de realizar un proyecto eficiente con calidad y rentable, intenta reducir el número de cambios que ocurran en el transcurso del software y realiza el análisis y diseño, tan completo como sea posible con un diseño genérico anticipándose a futuras necesidades. El cliente interactúa con el equipo de desarrollo mediante reuniones.

**Ventajas**

- Mitigación temprana de posibles riesgos.
- Progreso visible desde las primeras etapas.

- Temprana retroalimentación que se ajusta a las necesidades reales.
- Gestión de la complejidad.
- El conocimiento adquirido en una iteración puede aplicarse de iteración a iteración.

RUP en su modelado simplifica la realidad, proporciona los planos del sistema e incluye elementos que tienen gran influencia, omitiendo aquellos menores que no son relevantes para el nivel de abstracción dado.

Los modelos ayudan a visualizar como es o como queremos que sea el sistema, permiten especificar la estructura o comportamiento del sistema proporcionando plantillas que guiarán y documentarán las decisiones que se adopten.

El equipo de trabajo se encuentra familiarizado con esta metodología lo que agiliza el proceso de desarrollo, aspecto que se tuvo en cuenta, además de todo lo planteado anteriormente, para la elección de esta metodología de desarrollo.

## 2.9 Conclusiones Parciales

En este capítulo se abordaron las principales tecnologías que serán utilizadas como propuesta de arquitectura de software para el Sistema de Graficar Columnas Litológicas de Pozos de Petróleo. Para cumplir este objetivo se definieron herramientas, framework, metodologías, tecnologías, lenguaje de modelado y de programación, además de estilos y patrones arquitectónicos a usar. Para esta selección se tomaron en cuenta las principales características de cada elección, haciendo énfasis en sus ventajas y desventajas para alcanzar las necesidades de una arquitectura que garantice luego un software con la calidad, el tiempo y el costo requerido.

# CAPÍTULO 3 LÍNEA BASE DE LA ARQUITECTURA

## 3.1 Introducción

En este capítulo se desarrollará la propuesta arquitectónica para dar solución al problema dado a conocer durante el diseño teórico de la investigación, para alcanzar dicho objetivo tomaremos como punto de partida las descripciones de herramientas y tecnologías seleccionadas y descritas en epígrafes anteriores de este documento. Se obtendrá de esta manera una mejor visión del sistema, ya que se especificarán aspectos arquitectónicamente significativos y otros artefactos de la metodología de desarrollo seleccionada (Proceso Unificado de Desarrollo (RUP)), como son las 4+1 vistas de la arquitectura y el Documento de Descripción de la Arquitectura.

## 3.2 Arquitectura del Negocio

La arquitectura del negocio describe aspectos relevantes referentes a la organización, y es una de las tres arquitecturas que aparecen durante el desarrollo de un software. Esta posee un nivel alto de abstracción, además de describir procesos y estructuras básicas del negocio.

Se realizará por tanto un análisis de la arquitectura del negocio del CEINPET para el proceso de gráficas de columnas litológicas y partiendo de este punto dar lugar a la nueva arquitectura del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo para la empresa CUPET.

Hoy se realiza en el CEINPET el Parte Diario de Geología de todos los pozos en perforación, el cual incluye las columnas litológicas generadas que deben permitir la funcionalidad de ser modificadas en cualquier momento que el geólogo que se encuentre trabajando estime conveniente. En este parte se lleva además el registro de una serie de datos importantes generados a partir de las muestras periódicas que se recogen en cada uno de los pozos. Para llevar a cabo todo este trabajo, la entidad hace uso de software propietario, que no proporciona estabilidad y estandarización en las prácticas laborales**.**

Se hace necesario entonces diseñar e implementar un componente que permita graficar la columna litológica de los pozos. Este sistema debe cumplir las siguientes funcionalidades:

- Gestionar proyectos existentes.
- Imprimir un proyecto existente.
- Exportar ficheros referentes a las litologías en los formatos PDF, TXT y XML.
- Importar ficheros referentes a las curvas en los formatos PDF, TXT y XML.

- Crear una nueva litología y agregarla a la biblioteca de símbolos.
- Construir la columna Curvas.
- Construir la columna Litologías.
- Construir la columna Descripciones.
- Gestionar usuarios.
- Autenticar usuarios.

Para dar solución a dicha problemática la base arquitectónica del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo se ha modelado haciendo uso del Patrón Arquitectura en capas a través de una Aplicación Desktop desarrollada en C++ con framework QT y tecnología orientada a objetos. Se utiliza UML como lenguaje de modelado y Visual Paradigm Enterprise Edition como herramienta CASE, además como metodología de desarrollo Proceso Unificado de Desarrollo (RUP) y QTCreator como entorno de desarrollo.

### 3.2.1 ¿Por qué una Aplicación Desktop?

Una aplicación desktop sin dudas ofrece una gama de posibilidades para el desarrollo del Sistema para Graficar Columnas Litológicas, ya que luego de un análisis detallado de las características del Departamento de Sedimentología del CEINPET, y del proceso de Graficar Columnas Litológicas a automatizar, se llegó a la conclusión que se necesita una aplicación con gran capacidad de procesamiento y graficación visual, debido a la necesidad de graficar elementos como las curvas litológicas, además se espera un eficiente tiempo de respuesta, ya que la aplicación necesita de varios recursos de las computadoras que no son utilizados generalmente en los sistemas de cómputo de los pozos petroleros del país, así como que aporte la seguridad requerida y la menor probabilidad de fallos, pues se manejarán datos de importancia para la economía de dicha empresa y por tanto la del país. Además las aplicaciones desktop brindan facilidades específicas y prestaciones para gestionar usuarios, importar y exportar documentos y gestionar todo el procesamiento de ficheros, tareas necesarias para el desarrollo del sistema requerido.

Por tanto, se ha escogido una aplicación desktop, ya que ofrece varias ventajas:

- Mayor capacidad gráfica visual
- Menor tiempo de respuesta (aplicación más rápida)

- Mayor personalización. **(GNU Consultores, 2010).**

### 3.2.2 Restricciones Arquitectónicas

El Sistema para Graficar Columnas Litológicas del Pozos de Petróleo debe utilizar determinadas fuentes de información para realizar gran parte de sus funcionalidades y procesos de negocios como son la utilización de documentos, por lo que el sistema debe trabajar con estándares tanto de lectura como escritura, y poder exportar los formato PDF, TXT y XML.

El sistema está diseñado para que desde cualquier pozo petrolero del país se puedan realizar todas las operaciones diarias que se necesiten. Para todo esto es necesario la utilización de periféricos, tarjetas de video, espacio libre en disco y un CPU Pentium III o superior, además una impresora para imprimir los diferentes documentos.

## 3.3 Estructura del Equipo de Desarrollo

Se realizó una estructura de tipo formal para estructurar el equipo de desarrollo, y de esta manera plasmar los diferentes roles asignados entre el equipo, la distribución de sus puestos de trabajos, la interacción entre ellos, así como la distribución de tareas y responsabilidades.

### 3.3.1 Herramientas

Se mencionarán en este epígrafe las herramientas definidas para el desarrollo del sistema.

**Herramientas de Modelado:**

- Lenguaje de Modelado: Lenguaje Unificado de Modelado (UML).
- Herramientas CASE: Visual Paradigm.

**Herramientas de Desarrollo:**

- Lenguaje de Programación C++, framework QT.
- Entorno Integrado de Desarrollo (IDE) QT Creator.

### 3.3.2 Estructura del Equipo de Trabajo

El equipo de trabajo está distribuido de la siguiente forma:

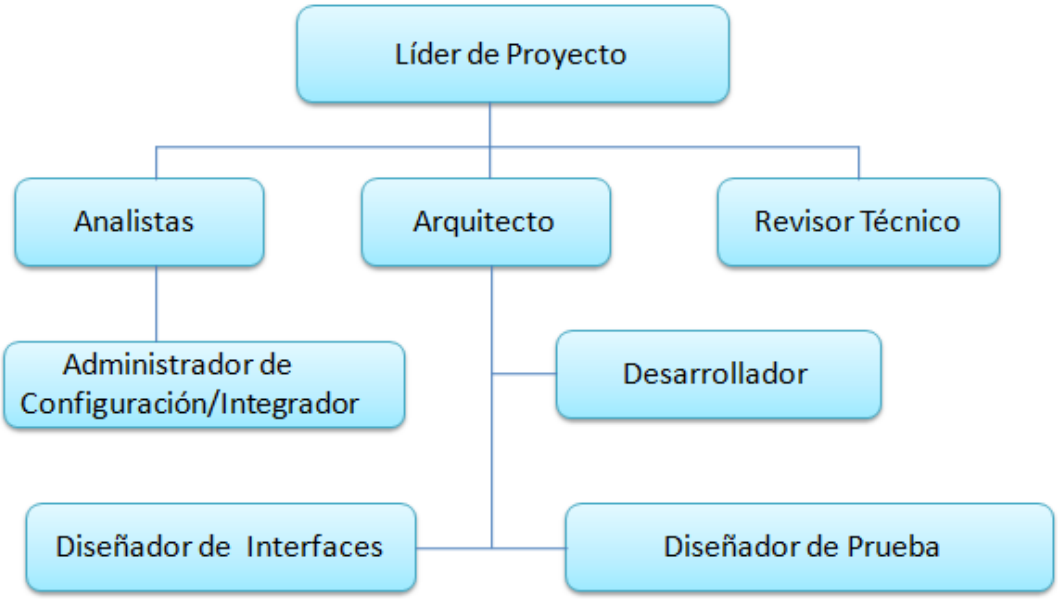


Figure 1 Estructura del Equipo de trabajo.

### 3.3.3 Configuración de los Puestos de Trabajos por Roles.

Líder del Proyecto

1. PC, con mouse y teclado.
2. Instalación de Visual Paradigm.
3. Instalación del paquete Office.

Analista

1. PC, con mouse y teclado.
2. Instalación de Visual Paradigm.
3. Instalación del paquete Office.

Arquitecto de software

1. PC, con mouse y teclado.

2. Instalación del Visual Paradigm.
3. Instalación del paquete Office.
4. Instalación del Visual Paradigm.

Diseñador

1. PC, con mouse y teclado.
2. Instalación del Visual Paradigm.

Implementador

1. PC, con mouse y teclado
2. Instalación del IDE QT Creator, framework QT y C++.

### 3.4 Organigrama de la Arquitectura

El organigrama de la arquitectura del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo refleja los aspectos internos en cuanto a estructura y organización que tendrá dicho sistema, lo que nos permite obtener una idea uniforme acerca de la estructura formal del sistema.

Se propone la arquitectura del Sistema para Graficar Columnas Litológicas de pozos de Petróleo que se encarga de automatizar el proceso de Graficar Columnas Litológicas, asegurando que se realice este proceso de forma eficiente y rápida. Para este sistema es de vital importancia la gestión de la información con el más alto grado de seguridad y calidad pues se manejarán datos de suma importancia, que si fallaran pondrían en riesgo la economía de la empresa. Por tanto deben ser eliminados cualquier tipo de fallos de la aplicación y esta última debe responder eficientemente ante todas las funciones necesarias.

A partir de los elementos y necesidades mencionadas se usará Arquitectura en Capas, específicamente Arquitectura en 3 Capas. Se usa este estilo debido a que es un modelo que simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo las dependencias de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores. Además, nos ayuda a identificar que puede reutilizarse, y proporciona una estructura que ayuda a tomar decisiones.

La Arquitectura en Capas define cómo organizar el modelo de diseño en capas que pueden estar físicamente distribuidas, por lo que los componentes de una capa solo pueden hacer referencias a componentes de capas inmediatas inferiores. También brinda los principales estilos de arquitectura estratificados de las aplicaciones distribuidas contemporáneas:

- Arquitectura de dos niveles.
- Arquitectura de tres niveles.
- Arquitectura de N niveles.

En el desarrollo específico del Sistema para Graficar Columnas Litológicas de Pozos de Petróleo se utilizará el estilo Arquitectura de tres niveles, ya que aporta centralización del control, escalabilidad y fácil mantenimiento, además que existen tecnologías suficientemente desarrolladas, diseñadas para este paradigma, que aseguran la seguridad en las transacciones, una interfaz amigable, y la facilidad de empleo, permitiendo la división de un problema complejo en una secuencia de pasos crecientes que lo hace más entendible y organizado.

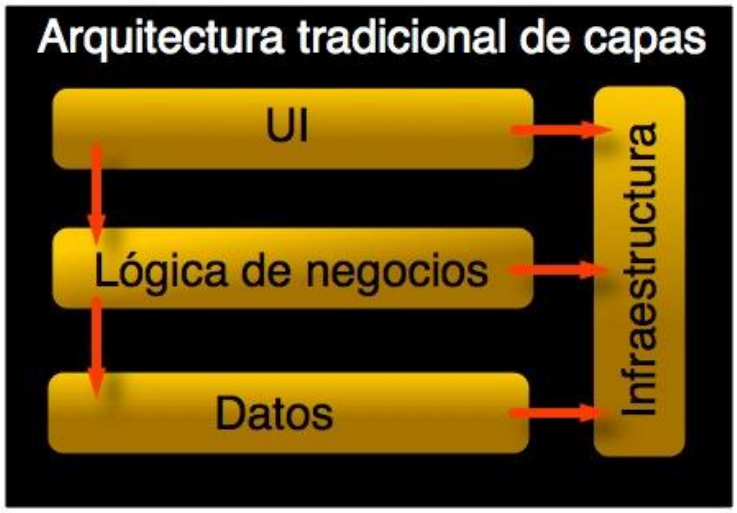


Figure 2 Arquitectura en Capas.

### 3.5 Descripción de la arquitectura

Se realizó la descripción de la Arquitectura ya que proporciona una comprensión arquitectónica global del sistema utilizando las vistas arquitectónicas definidas por la metodología RUP. Estas vistas se centran en aspectos concretos del sistema y son definidas como: vista lógica, vista de procesos, vista de despliegue, vista de implementación y la vista de casos de uso, las cuales permiten ver el sistema desde diferentes

perspectivas, por tanto, esta descripción mejora la comprensión del sistema, la organización del desarrollo, y fomenta la reutilización y toma de decisiones por los miembros del equipo de desarrollo.


Figure 3 4+1 Vistas Arquitectónicas.

### 3.5.1 Vista de Casos de Uso

La vista de casos de uso muestra los actores y la lista de los casos de uso o escenarios del modelo más significativos con las funcionalidades centrales del sistema, así como los requisitos más significativos, tanto funcionales como no funcionales, que tienen más impacto en la arquitectura y con los que se comprueba la funcionalidad de la misma para los elementos fundamentales, así como su comportamiento estable. Además es de gran importancia esta vista ya que une y coordina las cuatro vistas restantes de la arquitectura.

Para el desarrollo del Sistema de Graficar Columnas Litológicas de Pozos de Petróleo se definieron 20 requerimientos que fueron agrupados en 11 Casos de Uso del sistema de acuerdo a su impacto en la arquitectura, de la siguiente manera:

Tabla1 Clasificación de los Casos de Uso del Sistema

| Casos de Uso del Sistema | | | |
|---|---|---|---|
| **Críticos** | **Secundarios** | **Auxiliares** | **Opcionales** |
| Importar Ficheros | Gestionar Usuario | Exportar Ficheros | Imprimir proyecto |
| Construir Columna Curvas | Autenticar Usuario | Crear Litología | Modificar Contraseña |
| Construir Columna Litologías | | | |
| Construir Columna Descripciones | | | |
| Gestionar Proyecto | | | |

Los casos de uso identificados se pueden clasificar en críticos, secundarios, auxiliares y opcionales según su prioridad para la arquitectura.

- Los casos de uso críticos contienen las principales funcionalidades que el sistema debe cumplir y que son de vital importancia para los usuarios, por lo que definen la arquitectura básica.
- Los casos de uso secundarios sirven de apoyo a los casos de uso críticos, contienen aquellas funcionalidades de menor impacto sobre la arquitectura, aunque también deben implementarse tempranamente pues responden a requerimientos de interés para los usuarios.
- Los casos de uso auxiliares no son claves para la arquitectura y son usados sobre todo para completar casos de usos críticos o secundarios.
- Los casos de uso opcionales responden a funcionalidades cuya existencia es optativa para la aplicación y no son imprescindibles en las primeras versiones.

Otra clasificación importante para los casos de uso, son aquellos que se declaran arquitectónicamente significativos, ya que describen funcionalidades imprescindibles para el sistema a través de las cuales se valida la arquitectura propuesta para el mismo. Es responsabilidad del arquitecto priorizar dichos casos de uso atendiendo a su dificultad de desarrollo, importancia para la puesta en marcha del sistema, organización del desarrollo incremental, disponibilidad del equipo de desarrollo. Para el sistema se

clasificaron 5 casos de uso arquitectónicamente significativos atendiendo a los criterios mencionados anteriormente.

Casos de uso arquitectónicamente significativos:

- Importar Ficheros
- Construir Columna Curvas
- Construir Columna Litologías
- Construir Columna Descripciones
- Gestionar Proyecto

Para un mejor entendimiento de la distribución de los casos de usos definidos, actores del sistema y las relaciones entre ellos se presenta a continuación el Diagrama de Casos de Usos del Sistema:

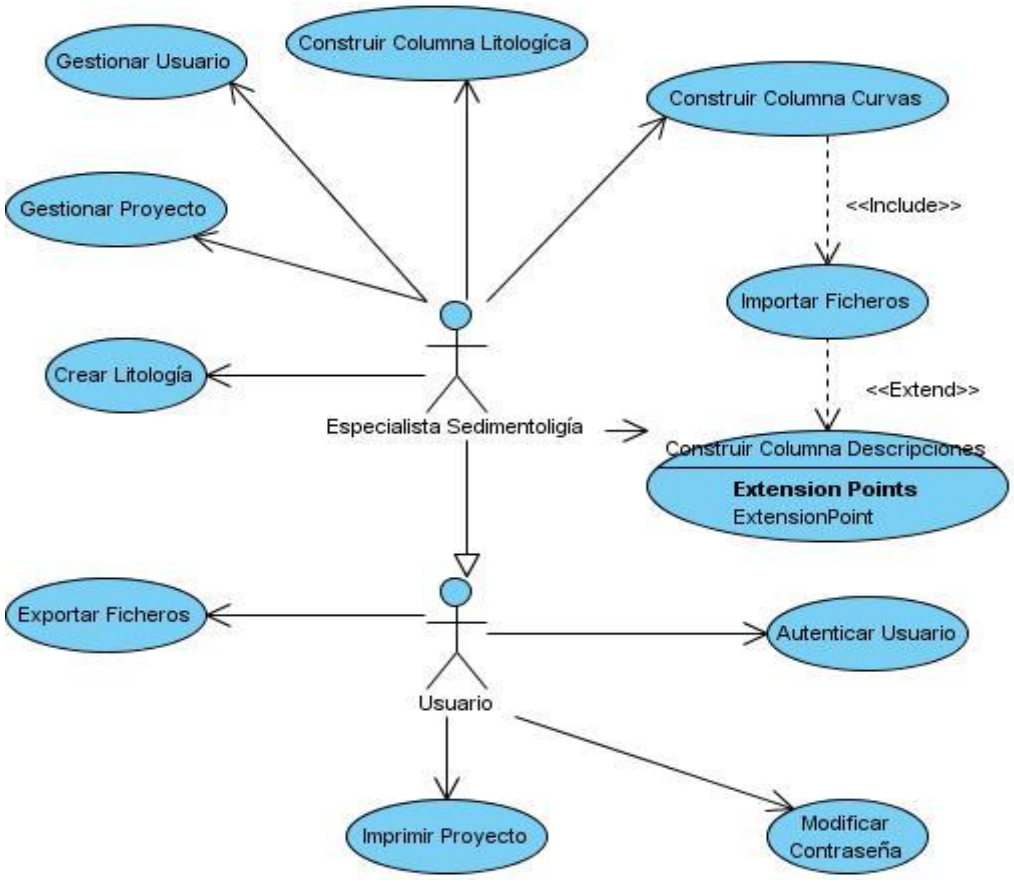


Figure 4 Diagrama de Casos de Uso del Sistema.

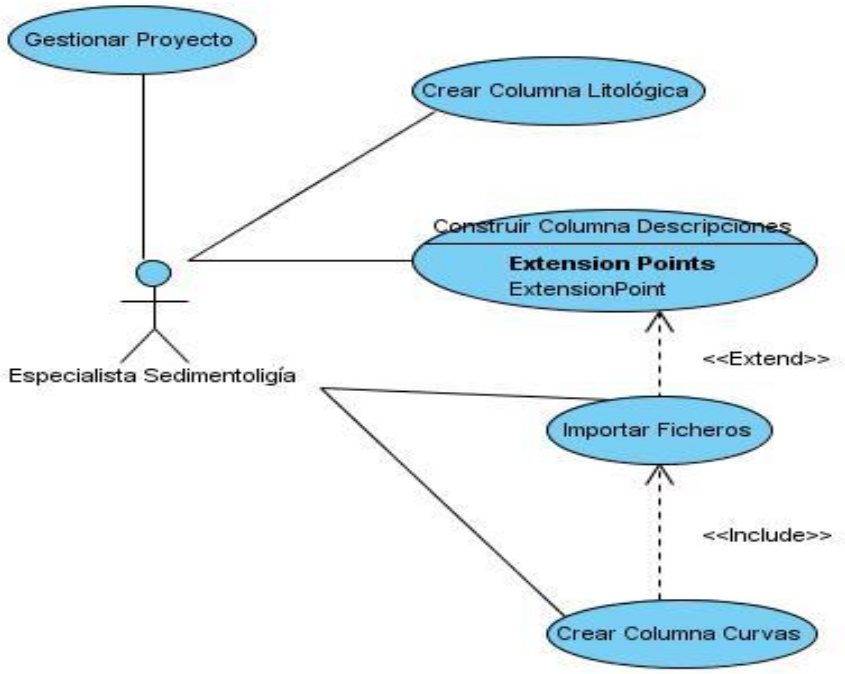


Figure 5 Diagrama de CUS Arquitectónicamente Significativos.

### 3.5.2 Vista de Procesos

La Vista de Procesos captura aspectos de sincronización y concurrencia, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Esta vista es usada cuando el sistema presenta procesos concurrentes o hilos, ya que controla los mismos. Es utilizada sobre todo para las clases activas, además es muy útil para realizar análisis de integridad y tomar decisiones de integración con otros sistemas.

El sistema a desarrollar en este caso, no presenta procesos concurrentes y no se hace necesario, por tanto, la representación de esta vista.

### 3.5.3 Vista Lógica

La vista lógica se basa fundamentalmente en lo que el sistema debe brindar en términos de servicios a sus usuarios. Es una abstracción del modelo de diseño e identifica a gran escala paquetes, subsistemas, clases, y las relaciones que existen entre ellas, ya sean de dependencia o de uso. Esto brinda una descomposición que potencia el análisis funcional, además de servir para identificar mecanismos y elementos de diseño comunes a diversas partes del sistema.

El siguiente diagrama nos muestra la organización de clases en paquetes y subsistemas:

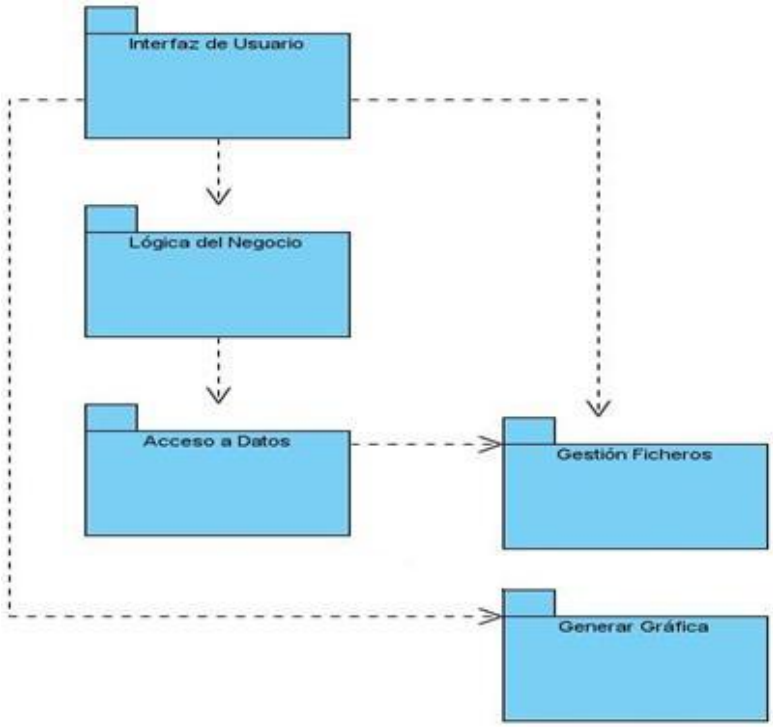


Figure 6 Diagrama de paquetes.

La siguiente figura ilustra la organización del sistema en Subsistemas más pequeños, con el objetivo de proveer una mayor reutilización y facilitar su mantenimiento.

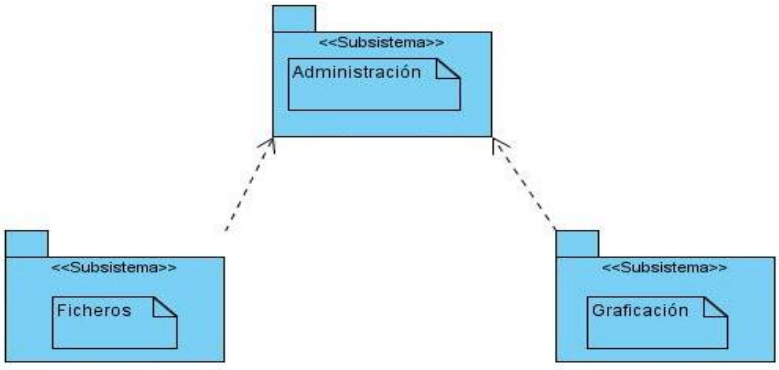


Figure 7 Diagrama de Subsistemas.

Los subsistemas principales identificados son:

- Subsistema Administración: Contiene la realización de los Casos de Usos correspondientes con la gestión de usuarios, roles y permisos sobre el sistema.
- Subsistema Graficación: Contiene la realización de los Casos de Usos correspondientes con la gestión de graficación de las distintas curvas, columnas y descripciones correspondientes.
- Subsistema Fichero: Contiene la realización de los Casos de Usos correspondientes con la gestión de ficheros como importación y exportación de los mismos.

### 3.5.4 Vista de Implementación

La Vista de Implementación se enfoca en la descomposición del software en capas y subsistemas de implementación, ya sean componentes de código fuente, binarios o ejecutables. Presenta la organización de los módulos del software en el ambiente de desarrollo del mismo; además provee una vista de la trazabilidad de los elementos de diseño de la vista lógica para la implementación. Con esta vista se abren caminos para los desarrolladores y para la visualización de la implementación permitiendo tomar decisiones respecto a las tareas del desarrollo.

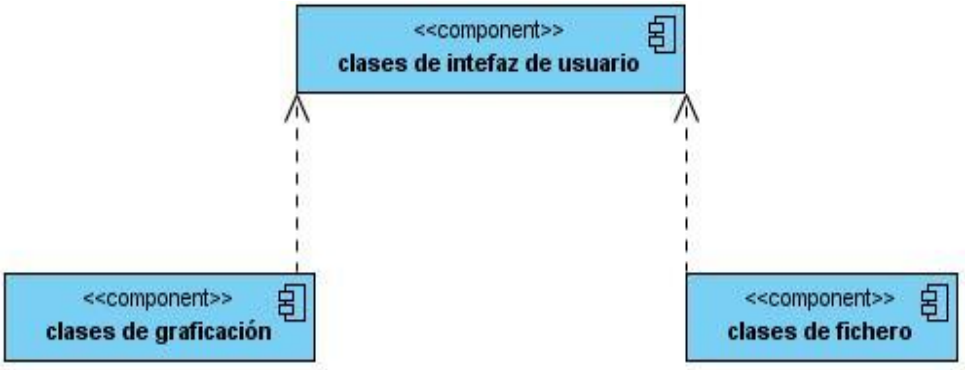


Figure 8 Diagrama de Componentes.

### 3.5.5 Vista de Despliegue

La vista de despliegue muestra la distribución física del sistema y cómo estarán distribuidos los componentes de la aplicación en ellos, describe el mapeo entre el software y el hardware reflejando su aspecto distribuido. Existe además una traza directa del modelo de implementación, ya que los

componentes físicos deben estar almacenados en nodos, lo que incluye la asignación de tareas provenientes de la vista de procesos en los nodos.

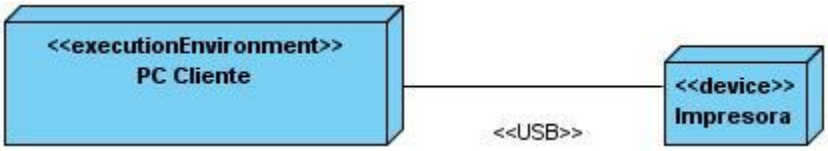


Figure 9 Diagrama de Despliegue.

**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:

PC Cliente

- Se requiere una PC Pentium III o superior, con procesamiento mínimo de 800Mhz.
- Se requiere 256 MB de RAM o superior.
- Se requiere 10 GB de disco duro o superior.

Dispositivo Impresora: Satisfacen los requisitos de los clientes de impresión de Graficas generadas por la aplicación. La forma de conectarse al dispositivo es a través del protocolo de comunicación USB.

USB (Bus Serie Universal): es un protocolo de transferencia de datos desde un dispositivo digital y viceversa.

### 3.6 Requerimientos no Funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. 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. En muchos casos los requisitos no funcionales son fundamentales en el éxito del producto además de formar una parte significativa de las especificaciones. También ayudan a marcar la diferencia entre un producto bien aceptado por los usuarios y uno con poca aceptación.

Para el desarrollo de la actual aplicación se tuvieron en cuenta los siguientes requisitos no funcionales:

### 3.6.1 Usabilidad

- La aplicación debe ser de fácil comprensión, navegación, configuración y utilización tanto para usuarios con un nivel alto de experiencia, como de niveles inferiores en el campo de la informática.
- La información debe ser mostrada de forma lógica y organizada.

### 3.6.2 Apariencia o Interfaz Externa

- El sistema debe brindar una interfaz externa de diseño sencillo y accesible a cualquier tipo de usuario, independientemente de su nivel de conocimiento de la aplicación.

### 3.6.3 Confiabilidad

- El sistema debe identificar claramente los diferentes niveles de usuario, garantizando la privacidad de toda la información existente.

### 3.6.4 Soporte

- El sistema debe presentar un equipo de soporte que le de mantenimiento al sistema cada 30 días.

### 3.6.5 Rendimiento

- El sistema debe tener un tiempo de respuesta rápido, de acuerdo con la funcionalidad que se esté usando en cualquier momento.

### 3.6.6 Soporte

- Se brindará servicios de instalación y configuración.
- La aplicación debe estar documentada y proveer el código fuente previendo futuras modificaciones en el mismo para potenciar su alcance o eficiencia.

### 3.6.7 Requerimiento de ayuda y documentación

- El sistema debe contar con un manual de usuario.
- El sistema debe contar con una ayuda disponible en cualquier momento que el usuario desee realizar una consulta para facilitar su trabajo con el mismo.

### 3.6.8 Requerimientos de Hardware

- Se requiere una PC Pentium III o superior, con procesamiento mínimo de 800Mhz.

- Se requiere 256 MB de RAM o superior.
- Se requiere 10 GB de disco duro o superior.

### 3.6.9 Portabilidad

- El sistema debe ser independiente de plataforma siendo compatible con cualquier sistema operativo.

### 3.6.10 Seguridad

- El sistema debe proteger la información de personal no autorizado.
- El sistema debe presentar niveles de seguridad de acuerdo con los permisos de usuarios que puedan acceder al mismo.
- La información debe estar organizada diferenciando las opciones de acceso a usuarios con diferentes roles.
- Los datos generados por la aplicación estarán serializados, además de que los usuarios comunes dígase secretarias o personal que no esté autorizado o facultado para hacer uso de la aplicación, solo tendrá permiso de impresión, autenticación y cambio de contraseña. Una vez autenticado si es uno de los especialistas en sedimentología podrá realizar las demás prestaciones que permite la aplicación.

## 3.7 Conclusiones Parciales

Es este capítulo se ha definido la línea base de la arquitectura del Sistema para Graficar Columnas Litológicas de pozos de petróleo, para ello se tomaron en cuenta aspectos arquitectónicamente significativos que tienen que ver con las tecnologías y herramientas a utilizar para el desarrollo de la aplicación. Se hizo también un organigrama de la arquitectura para analizar la estructura organizativa del ámbito de desarrollo de dicho sistema, además de haber analizado la distribución del equipo de trabajo y recursos existentes. Fue de vital importancia en este capítulo la utilización de las 4+1 vistas de la arquitectura, definidas por la metodología RUP.

**Conclusiones Generales**

Durante el decursar de esta investigación se le dió cumplimiento al objetivo propuesto: una solución arquitectónica del Sistema para Graficar Columnas Litológica de pozos de petróleo, mediante un estudio minucioso y una documentación adecuada. Esta es una arquitectura sólida y escalable, no obstante, debe mantenerse actualizada a lo largo de la vida del sistema reflejando cambios y adiciones necesarias para el mismo.

Es de vital importancia una profunda comprensión por parte del equipo de trabajo de esta arquitectura, pues de ello depende que el sistema a desarrollar cumpla con todos los requerimientos planteados a lo largo del documento, es decir, que sea un sistema, robusto y comprensible a todos los usuarios que interactuarán con el software de una forma u otra, garantizando que esto sea de manera sencilla y rápida.

Se han generado como resultado de este estudio los artefactos correspondientes al rol Arquitecto de Software: Especificación de Requerimientos Documento de Arquitectura de Software.

Se ha llegado también a la conclusión de que luego de una exposición de los principales aspectos de la descripción arquitectónica se logró seleccionar el estilo arquitectónico adecuado, Arquitectura en Capas, junto a una serie de patrones y restricciones arquitectónicas necesarias. Además se definieron las principales tecnologías y herramientas a utilizar en el desarrollo del sistema, como son el Lenguaje Unificado de Modelado (UML) y Visual Paradigm, como herramientas de modelado; lenguaje de programación C++ con el framework QT.

Como parte además de esta arquitectura se configuraron los puestos de trabajo por roles en el equipo de desarrollo, se realizó un organigrama de la arquitectura como elemento organizativo del ámbito de desarrollo del sistema a implementar, y se utilizaron las 4+1 vistas de la arquitectura, definidas por la metodología RUP, que permitieron una visión más detallada de la solución arquitectónica del sistema.

Finalmente, se considera que esta arquitectura cumple con todos los requisitos que se necesitan para el desarrollo del Sistema para Graficar Columnas Litológicas de pozos de petróleo.

**RECOMENDACIONES**

Para el posterior desarrollo de este trabajo investigativo, sobre la solución arquitectónica del Sistema para Graficar Columnas Litológicas, se recomienda:

- Profundizar y refinar esta propuesta de arquitectura durante todo el ciclo de desarrollo del software, con el objetivo de adaptarse a posibles cambios y obtener mejoras.
- Certificar la arquitectura mediante métodos de evaluación de la misma (ATAM), para identificar las principales debilidades arquitectónicas que puedan existir.
- Evaluar la utilización de esta solución arquitectónica para proyectos similares que utilicen resultados de esta investigación o de la industria petrolera.

## REFERENCIAS BIBLIOGRÁFICAS

Trabajos citados


2002 dcc.uchile.cl. [En línea] [Citado el: 13 de 12 de 2009.] http://www.dcc.uchile.cl.

1997 Domains of Concern in Software Architectures and Architecture Description Languages. California1997

2000 Extreme Programming Explained. Embrace ChangeAddison-Wesley Professiona20000201616416

Proceedings of the 21st International Conference on Software Engineering (ICSE'99), Los Angeles. Rosenblum, Elisabetta Di Nitto y David. 1999. Los Angeles : s.n., 1999.

2003"Towards a reference architecture for Web services". Conference and ExpositionFiladelfia

1996A Survey of Architecture Description Languages. AlemaniaProceedings of the International Workshop 1996

Enero, 1994An Introduction to Software ArchitectureUSAEnero, 1994

Ayala, Ramon Montero. bibliodoc.uci.cu. [En línea] [Citado el: 22 de enero de 2010.] http://bibliodoc.uci.cu/pdf/reg01501.pdf.

Ballestero, Carlos rafael. 2007. Arquitectura de Software del Sistema de GEstion de Informacion. Ciudad de la Habana : s.n., 2007.

Booch. 1999. 1999.

Canal, Carlos. marzo 2005. Arquitectura, marcos de trabajo y patrones. Universidad de Málaga : s.n., marzo 2005.

Casanovas, Josep. 2004. www.desarrolloweb.com. [En línea] 9 de septiembre de 2004. [Citado el: 15 de 1 de 2010.] http://www.desarrolloweb.com/articulos/1622.php.

Christopher Alexander. 1977. www.microsoft.com . [En línea] 1977. [Citado el: 10 de 12 de 2009.] http://www.microsoft.com.

Chung, Martin. 2002. msdn.microsoft.co. [En línea] 2002. [Citado el: 27 de enero de 2010.] http://msdn.microsoft.com/library/default.asp?url=/library.

Editorial MKM. 2008. http://www.mkm-pi.com. [En línea] febrero de 2008. [Citado el: 10 de marzo de 2010.] http://www.mkm-pi.com.

ERIKA CAMACHO, FABIO CARDESO, GABRIEL NUÑEZ. 2004. prof.usb.ve. [En línea] abril de 2004. [Citado el: 3 de enero de 2010.] http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf.

2004Estilos y Patrones en la Estrategia de Arquitectura de MicrosoftBuenos Aires2004

2007Evaluando Arquitecturas de SoftwareMexicoBrainworx20070888.

2007Evaluando Arquitecturas de Software. Parte 1 Panorama GeneralMexico20070888

Fowler. 2002. 2002.

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. 1996. Oriented Software Architecture. A System of Patterns. 1996.

Galeon. 2006. [En línea] 2006. [Citado el: 30 de 4 de 2010.] http://www.galeon.com/zuloaga/Doc/ArqCapasSI.doc.

Gallardo, Ing. Paul Lorenzo. 2009. www.slideshare.net. [En línea] 2009. [Citado el: 20 de enero de 2010.] http://www.slideshare.net/BiblioSher/base-de-datos-2636450.

Garcia, Joaquin. 2005. www.ingenierosoftware.com. [En línea] 27 de mayo de 2005. [Citado el: 23 de enero de 2010.] http://www.ingenierosoftware.com.

Gastón. 2007. www.scribd.com. [En línea] 10 de octubre de 2007. [Citado el: 21 de enero de 2010.] http://www.scribd.com/doc/19475538/Java-Su-Historia-Ediciones-Versiones-y-Caracteristicas-Como-Plataforma-y-Lenguaje-de-Programacion?secret_password=&autodown=pdf.

Giron, Marco Antonio Pereira. 2007. www.scribd.com. [En línea] 2007. [Citado el: 2 de marzo de 2010.] http://www.scribd.com/doc/2080534/UML.

GNU Consultores. 2010. gnuconsultores. [En línea] 2010. [Citado el: 1 de 4 de 2010.] http://www.gnuconsultores.com/es/ingenieria/desarrollo/escritorio.

Gutierrez, Jorge A. Saavedra. 2007. jorgesaavedra.wordpress.co. [En línea] 2007. [Citado el: 25 de enero de 2010.] http://jorgesaavedra.wordpress.com/2006/08/17/patrones-grasp-craig-larman/.

Huawei Technologies Co., Ltd. 2009. Huawei. [En línea] Huawei Technologies Co., Ltd, 2009. http://www.huawei.com/es/catalog.do?id=-4.

Huidobro, José Manuel. 2008. IPTV, la televisión a través de internet. 2008.

Ivan Ayala Catari, Yecid Tomas Roero Marca. 2007. www.scribd.com. [En línea] 2007. [Citado el: 17 de enero de 2010.] http://www.scribd.com/doc/25374125/Estudio-de-Herramientas-CASE-de-Soporte-a-UML-y-UML2.

Jara, Omar Hurtado. www.monografias.com. [En línea] [Citado el: 22 de enero de 2010.] http://www.monografias.com/trabajos27/evolucion-patrones/evolucion-patrones.shtml.

José H. Canós, Patricio Letelier y Mª Carmen Penadés. Métodologías Ágiles en el Desarrollo de Software. Valencia : DSIC -Universidad Politécnica de Valencia.

Juan Pablo López, Juan Pablo Martín, y Javier de la Rosa. 2007. forge.morfeo-project.org/. [En línea] 19 de noviembre de 2007. [Citado el: 27 de enero de 2010.] http://forge.morfeo-project.org/wiki/index.php/D.3.2.2_Modelos_avanzados_de_comunicaci%C3%B3n_de_recursos#Estilo_basado_en_c.C3.B3digo_m.C3.B3vil.

Kiccillof, Carlos Reinoso y Nicolas. 2004. sophia.javeriana.edu.co. [En línea] marzo de 2004. [Citado el: 25 de enero de 2010.]
http://sophia.javeriana.edu.co/~cbustaca//Arquitectura%20Software/Documentos/Introduccion/Articulos/Reyn2004a.pdf.

Kiccillof, Carlos Reynoso – Nicolás. 2004. www.willydev.net. [En línea] marzo de 2004. [Citado el: 27 de enero de 2010.] http://www.willydev.net/descargas/prev/Estiloypatron.pdf.

Lafuente, Guillermo Javier. 2001. gidis.ing.unlpam.edu.ar. [En línea] 2001. [Citado el: 16 de enero de 2010.] http://gidis.ing.unlpam.edu.ar/personas/glafuente/uml/uml.html.

Luis Giraldo, Yuliana Zapata. 2005. http://hugolopez.phi.com. [En línea] 24 de septiembre de 2005. [Citado el: 16 de enero de 2010.] http://hugolopez.phi.com.co/docs/download/file=Giraldo-Zapata-Herramientas%20de%20ISW.pdf,_id=17.

Macias, Yolanda Benitez. 2008. Sistema para la gestión y control de la cartelera del canal cultural de la Universidad de las Ciencias Informáticas. Ciudad de la Habana : Universidad de la Ciencias Informáticas, 2008.

MagicDraw. 2009. www.magicdraw.com/. [En línea] 28 de diciembre de 2009. [Citado el: 18 de enero de 2010.] http://www.magicdraw.com/.

Martínez, Yarisel Rodríguez. 2009. Diseño de un Sistema Gestor de Guía Electrónica de Programación. Habana : Universidad de las Ciencias Informáticas, 2009.

Mary Garlan, David and Shaw. 1994. An Introduction to Software Architecture. Estados Unidos : s.n., 1994.

MINISTERIO DE INDUSTRIA, TURISMO Y COMERCIO. 2009. TDT(Televisión Digital Terrestre). [En línea] 1 de octubre de 2009. [Citado el: 1 de diciembre de 2009.] http://www.televisiondigital.es.

Mora, Sergio Luján. 2006. C++ paso a paso. 2006.

Nicolas Kiccillof, Carlos Reinoso. 2004. carlosreynoso.com.ar. [En línea] marzo de 2004. [Citado el: 25 de enero de 2010.] http://carlosreynoso.com.ar/wp-content/plugins/download-monitor/download.php?id=155.

Oktaba, Hanna. 2007. www.mcc.unam.mx. [En línea] 2007. [Citado el: 23 de enero de 2010.] http://www.mcc.unam.mx/~cursos/Algoritmos/javaDC99-2/patrones.html.

ORACLE Packs de Gestión de Base de Datos de Oracle. [En línea] [Citado el: 21 de enero de 2010.] http://www.oracle.com/global/es/products/database/db_manageability.html.

August 8, 1996Pattern Oriented Software Architecture 1 editionAugust 8, 19960471958697

1995Pattern-Oriented Software ArchitectureA System of PatternsInglaterra

1996Pattern-Oriented Software Architecture England1996 0 417 95869 7

Paul Bustamante, Iker Aguinaga, Miguel Aybar. 2004. www.tecnun.es. [En línea] 2004. [Citado el: 21 de enero de 2010.]

http://www.tecnun.es/asignaturas/Informat1/ayudainf/aprendainf/Cpp/basico/cppbasico.pdf.

Principled design of the modern Web architecture. Taylor, Roy Thomas Fielding y Richard. 2002. 2002.

Proyecto GNU. 2008. La Definición de Software Libre. [En línea] 2008. http://www.gnu.org/philosophy/free-sw.es.html.

Rational Software Corporation. 2003. www.slideshare.net. [En línea] 2003. [Citado el: 16 de enero de 2010.] http://www.slideshare.net/vivi_jocadi/rational-rose.

Rosas Garcia, ISC PROF LAURO SOTO. www.mitecnologico.com. [En línea] [Citado el: 22 de enero de 2010.] http://www.mitecnologico.com/Main/XmlAntecedentes.

Sanchez, María A. Mendoza. 2004. www.informatizate.net. [En línea] 7 de julio de 2004. [Citado el: 15 de 1 de 2010.]

http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html.

Seco, José Antonio González. 2001. http://www.programacion.com/tutorial/csharp/3/. [En línea] 2001. [Citado el: 21 de enero de 2010.] http://http://www.programacion.com/tutorial/csharp/3//tutorial/csharp/3/.

April 11, 2003Software Architecture in PracticeSecond Edition.April 11, 20030-321-15495-9

Universidad EAFIT. 2005. www.eafit.edu.co. [En línea] 17 de febrero de 2005. [Citado el: 23 de enero de 2010.]

http://www.eafit.edu.co/NR/rdonlyres/223A8F47-27B5-4EB8-B695-4097F745D701/0/Arquitectura.pdf.

Valdés, Damián Pérez. 2007. www.maestrosdelweb.com. [En línea] 26 de octubre de 2007. [Citado el: 20 de enero de 2010.] http://www.maestrosdelweb.com/principiantes/%C2%BFque-son-las-bases-de-datos/.
Valera, Isabel. www.monografias.com. [En línea] [Citado el: 20 de enero de 2010.] http://www.monografias.com/trabajos4/basesdatos/basesdatos.shtml#top.
Valle, José. 2005. http://www.monografias.com. [En línea] 2005. [Citado el: 15 de enero de 2010.] http://www.monografias.com/trabajos24/herramientas-case/herramientas-case.shtml..
Vázquez, Carlos Luis Serrano Rosales , Solangel Rodríguez. 2008. Desarrollo de Biblioteca de Métodos Numéricos (BMN), referente a Sistemas de Ecuaciones Lineales, Sistemas de Gran Dimensión y Poco Densos e Integración Numérica. Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2008.
Visual-Paradigm. 1995. [En línea] 1995. [Citado el: 16 de enero de 2010.] http://www.visual-paradigm.com/product/vpuml/.
www.monografias.com. [En línea] [Citado el: 21 de 1 de 2010.]
http://www.monografias.com/trabajos13/trsqlinf/trsqlinf.shtml#SOL.

**Bibliografía**

1. Booch, Grady. Object-Oriented Analysis and Design. Second Edition. s.l. : Benjamin/Cummings, 1994.
2. Jacobson, Ivar, Grady Booch, and James Rumbaugh. El Proceso Unificado de Desarrollo de Software. México : Addison-Wesley, 1999.
3. Kruchten, Philippe. "Architectural Blueprints--The 4+1 View Model of Software Architecture". November 1995.
4. Larman, Craig. UML y Patrones, Introducción al análisis y diseño orientado a objetos. México : s.n., 1999.
5. Martin, Robert C. "Design Principles and Design Patterns".
6. Shaw, David Garlan and Mary. "An Introduction to Software Architecture", Advances in Software Engineering and Knowledge Engineering, Volume I. New Jersey : V.Ambriola and G.Tortora, World Scientific Publishing Company, 1993.
7. Buschmann, Meunier, Rohnert, Sommerlad, Stal - Wiley. A System of Patterns.
8. Fielding, Roy Thomas. Architectural Styles and the design of network-based software architectures. University of California : s.n., 2000.

**GLOSARIO**

**ActiveX:** Componente que se puede insertar en una página Web para proporcionar una funcionalidad que no está directamente disponible en HTML.

**BD:** Base de Datos.

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

**CUS:** Casos de Uso del Sistema.

**CEINPET:** Centro de Investigaciones del Petróleo.

**Columna Curvas:** En ellas estarán recogidas una serie de gráficas que se representan con curvas como son: la Velocidad de Perforación, Impregnación, Porosidad, Calcimetría y los Gases.

**Columna Descripciones:** Es la forma en que los geólogos pueden hacer de manera rápida anotaciones al margen de cada una de las secciones que están analizando.

**Cortes o Sartas:** Se refiere a Tuberías de acero que se unen para formar un tubo desde la barrena de perforación hasta la plataforma de perforación.

**Columnas Litológicas:** Un perfil litológico o columna litológica representa gráficamente la estratigrafía generalizada de un sector. Es donde se representa gráficamente los diferentes componentes litológicos por los que atraviesa el pozo.

**DIPP:** Dirección de Intervención y Perforación de Pozos.

**EPEPC:** Empresa de Producción y Extracción de Petróleo del Centro.

**Sedimentología:** Es el estudio y análisis de los sedimentos.

**Framework:** Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

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

**GRAP:** Patrones de Asignación de Responsabilidades.

**Geología:** Es la ciencia que trata de la forma exterior e interior del globo terrestre, de la naturaleza y de las materias que lo componen y de su formación; de los cambios o alteraciones que estas han experimentado desde su origen, y de la colocación que tienen en su actual estado.

**HDD:** Disco duro.

**HTTP:** Protocolo de transferencia de hipertexto, en inglés Hyper Text Transfer Protocol.

**IDE:** integración con herramientas de desarrollo.

**J2EE:** plataforma Java 2 Enterprise Edition.

**JVM:** Máquina Virtual de Java.

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

**MVC:** Modelo Vista Controlador.

**PetroSoft:** Soluciones Informáticas para la Industria del Petróleo.

**POO:** Programación orientada a objetos.

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

**RAM:** Memoria de Acceso Aleatorio, Random Access Memory.

**RNF:** Requerimientos no Funcionales.

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

**MINBAS:** Ministerio de la Industria Básica.

**SGBD:** Sistemas Gestores de Base de Datos.

**SOA:** Arquitectura Orientada a Servicios.

**UCI:** Universidad de las Ciencias Informáticas.

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

**Visual Interdev:** Es un entorno de desarrollo integrado (IDE) usado para crear aplicaciones web usando las tecnologías de Microsoft Active Server Pages (Páginas de Servidor Activo o ASP).

**XML:** Lenguaje de Marcas, en inglés Extensible Markup Language. XP: Programación Extrema, en inglés Extreming Programing.