UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
Facultad #9

DISEÑO DE LA ARQUITECTURA DE SOFTWARE PARA EL SISTEMA DE ANÁLISIS DE REGISTROS DE POZOS PETROLEROS (ANPER).

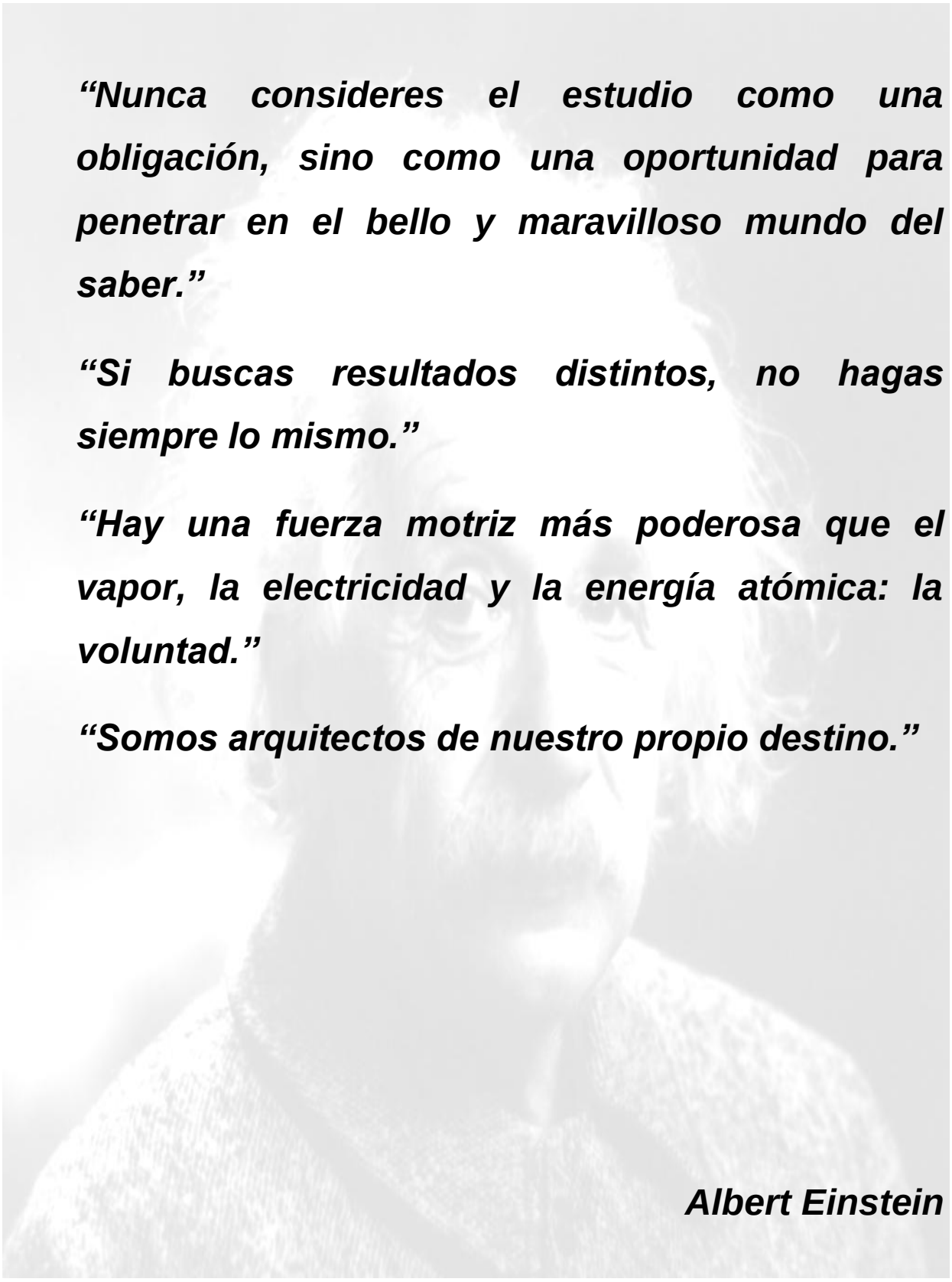
TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS.

AUTOR: Francisco Vega Leyte Vidal

TUTOR: Ing. Danis Rego Castillo

CO-TUTOR: Ing. Armando Ortíz Cabrera

Ciudad Habana, julio del 2010.
"Año del 52 de la Revolución"



“Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber.”

“Si buscas resultados distintos, no hagas siempre lo mismo.”

“Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energía atómica: la voluntad.”

“Somos arquitectos de nuestro propio destino.”

Albert Einstein

DEDICATORIA

Gracias a Dios Todo Poderoso y la Virgen de la Caridad del Cobre por siempre acompañarme y haberles dado salud a mis padres para ver este sueño realidad.

A mis padres por haber participado conmigo en los momentos felices y tristes de mi vida, por haber confiado en mí, por haber respetado mis decisiones, por todo el cariño y afecto que me han brindado porque todo lo que soy y seré es gracias a ustedes.

Mami te agradezco todo lo que has hecho en la vida por mí, solo por el simple hecho de darme la posibilidad de venir al mundo es lo más hermoso que me ha podido pasar en la vida. Por ser la única mujer que en cualquier circunstancia en la que yo me encuentre me querrá como el primer día, por la educación que me has enseñado, por ser la mujer que más admiro y quiero, por malcriarme en todo momento y por ser yo, como tú dices: "Tu vida".

Papi gracias por ser un guía para mí, por enseñarme a compartir con los demás, por entenderme como soy y hacerme ver sin imposiciones el camino correcto, por ser el único amigo que no me fallará y estará conmigo siempre, por estar siempre al lado mío y el de mi madre, por enseñarme que debo responsabilizarme por mis actos, por enseñarme que la familia está por encima de todas las cosas de la vida, por haberme enseñado el otro lado de la vida y por enseñarme que no importa lo que pase en la vida por grande que sea "Que más adelante hay casas y bastante pueblo por conocer ".

Ustedes dos pueden vivir convencidos que no les fallaré y que pueden contar conmigo en todo momento que lo único que yo no seré es un mal hijo, este logro es para ustedes. GRACIAS.

A mi hermana por ser la mejor hermana del mundo, por haberme ayudado cuando podía, por haber estado cerca de mis padres en los momentos que yo no pude, por ser una amiga incondicional, por haberme cuidado

cuando era pequeño, por guiarme y enseñarme valores humanos, este logro también es gracias a ti.

A mis abuelos en especial a Sonia por ser mi segunda madre, por todo el cariño y afecto que me ha dado. A Nilo Solera que aunque ya no está presente este es un sueño hecho realidad.

A toda mi familia en general le dedico cada línea de este trabajo, por darme lo mucho y lo poco.

A mis amigos de Palma Soriano Yannier Moreno López y Yuri Fernando Castellano Odio.

A mis queridos amigos de la UCI que de una forma u otra estuvieron estos cinco maravillosos años conmigo, los cuales más que mis amigos son mis hermanos.

A todos muchas gracias……

Paco

Agradecimientos

A mis padres y familia por estar siempre a mi lado.

A mis amigos de la UCI por ser mi familia y poder contar con ellos en todo momento, haberlos conocido ha sido uno de los regalos más grandes que me ha dado la vida, créanme que olvidarlos será imposible.

A todos los profesores que me guiaron en estos cinco años en especial a mi tutor y cotutor, por darme fuerzas para seguir adelante, por su dedicación y apoyo incondicionalmente.

A mi oponente y a los miembros del jurado por sus oportunas críticas y por su apoyo incondicional.

A fin a todos aquellos que siempre confiaron en mí.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al Centro de Investigaciones del Petróleo (CEINPET) y a la Universidad de Ciencias Informática (UCI) a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los _5_ días del mes de _julio_ del año _2010_.

Francisco Vega Leyte Vidal

_____

Firma del Autor

Danis Rego Castillo

_____

Firma del Tutor

DATOS DE CONTACTO

Tutor:

Nombre y apellidos: Danis Rego Castillo.

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: drego@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: Armando Ortíz Cabrera.

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: aortizc@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

El petróleo es un recurso natural no renovable que genera la mayor parte de la energía que se consume en el mundo. Nuestro país cuenta con el Centro de Investigaciones del Petróleo (CEINPET) que se dedica a dar respuesta de forma integral a toda la actividad petrolera desde la exploración hasta la refinación. Dentro de los procesos que se realizan en este centro se encuentra el análisis petrofísico de los núcleos y registros de pozos extraídos de los yacimientos, tanto en exploración como en explotación.

Este proceso se realiza actualmente con productos propietarios los cuales restringen su uso por medio de licencia, pero para disponer de estos sistemas y de sus actualizaciones hay que pagar grandes cantidades de dinero. Es entonces que surge la idea de desarrollar un software netamente cubano que permita redecir los costos por importaciones y que satisfaga las necesidades y a la vez respalde las investigaciones y análisis de los pozos petroleros.

En aras de materializar la idea enunciada anteriormente, un grupo de desarrollo perteneciente a la Facultad 9 de la Universidad de las Ciencias Informáticas, se planteó la tarea de desarrollar el Sistema de Análisis de Registros de Pozos Petroleros (ANPER), el cual dará soporte al proceso de análisis petrofísico a partir de la lectura de los registros de pozos. Como parte del proceso de desarrollo de este software se identificó la necesidad de lograr un producto escalable, modificable y reutilizable, por lo que se decidió realizar la presente investigación que tiene como objeto de estudio: el proceso de análisis petrofísico a partir de la lectura de los registros de pozos.

Luego de un estudio del estado del arte fueron caracterizadas otras aplicaciones informáticas ya existentes que le dan solución al proceso de análisis petrofísico. Además se seleccionaron las herramientas a utilizar en el desarrollo de la aplicación, así como el lenguaje de programación y la metodología de desarrollo que guiará dicha aplicación. Como resultado se diseñaron las vistas arquitectónicas de la arquitectura de software definidas por la metodología de desarrollo a utilizar y se validó dicha arquitectura llegando a la conclusión que cumple con los requerimientos no funcionales del sistema.

PALABRAS CLAVES

Arquitectura, Tecnologías, Vistas, Proceso de análisis petrofísico.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El petróleo se origina de una materia prima formada principalmente por detritos de organismos vivos acuáticos, vegetales y animales, que vivían en los mares, en las lagunas o en las desembocaduras de los ríos. El petróleo es un recurso natural no renovable que aporta el mayor porcentaje de energía que se consume en el mundo, por la importancia que tiene este hidrocarburo hoy día existen muchas empresas que se dedican al análisis petrofísico ya que con este análisis se obtiene información de las propiedades físicas de las rocas tales como la porosidad, la saturación de agua, la permeabilidad, hasta el análisis en laboratorio de los núcleos y la interpretación de registros de pozos mediante su visualización. Todo esto conlleva a que se realice de forma más eficiente la exploración y la posterior explotación de los yacimientos de hidrocarburos.

Cuba no está exenta del estudio y la producción de este hidrocarburo, aunque sus producciones sean pequeñas, cuenta con un Centro de Investigaciones del Petróleo (CEINPET) que se dedica a dar respuesta de forma integral a toda la actividad petrolera, desde la exploración hasta la refinación.

Dentro de los procesos que se realizan en este centro se encuentra el análisis petrofísico de los núcleos y registros de pozos extraídos de los yacimientos, tanto en exploración como en explotación. “La petrofísica es el estudio de las propiedades físicas y químicas que describen la incidencia y el comportamiento de las rocas, los sólidos y los fluidos”. **(1)** Para llevar a cabo los trabajos en dicha área, el centro cuenta con conjunto de productos propietarios que se encargan de interpretar y graficar la información existente en los registros de pozos. Dichos productos restringen su uso por medio de licencias, lo que provoca, unido a la falta de acceso a las actualizaciones por ser costosas para el centro, que sólo pueda ser usado en una computadora a la vez. Esto trae consigo que se acumule y se torne engorroso el trabajo, además de que parte de sus investigadores se vean excluidos de la utilidad que brindan dichas aplicaciones.

En aras de revertir lo expresado anteriormente, un equipo de desarrollo de la facultad 9 de la Universidad de las Ciencias Informáticas (UCI), se encuentra enfrascado en el desarrollo de un software para el análisis petrofísico que se adapte a las necesidades de los trabajadores del CEINPET.

Partiendo de la situación descrita se identificó como **problema a resolver**: ¿Cómo diseñar una arquitectura de software robusta y flexible que se adapte a las necesidades del Sistema de Análisis de Registros de Pozos Petroleros (ANPER)?

Realizando un análisis del problema planteado se determinó como **objeto de estudio**: el proceso de análisis petrofísico a partir de la lectura de los registros de pozos y como **campo de acción**: diseño y descripción de la arquitectura base para el Sistema de Análisis de Registros de Pozos Petroleros (ANPER).

Para dar solución al problema se trazó como **objetivo general:** diseñar una arquitectura de software robusta y flexible para el Sistema de Análisis de Registros de Pozos Petroleros (ANPER), que permita al grupo de desarrollo tomar decisiones sobre los aspectos dinámicos y estáticos más significativos de la organización del sistema.

Para llevar a cabo la investigación se plantean las siguientes **tareas de investigación**:

- Caracterizar las diferentes propuestas arquitectónicas.
- Comparar las propuestas arquitectónicas caracterizadas, teniendo en cuenta aspectos positivos y negativos.
- Caracterizar los patrones arquitectónicos seleccionados.
- Definir las librerías de clases y herramientas a utilizar en el desarrollo.
- Refinar y diseñar una propuesta de arquitectura que cumpla con los requerimientos.
- Modelar la arquitectura propuesta.
- Validar de la arquitectura propuesta.

Se plantea como **idea a defender** que: si se diseña una arquitectura de software que cumpla con los requerimientos y atributos de calidad establecidos para el Sistema de Análisis de Registros de Pozos Petroleros (ANPER), se garantizará un producto modificable, reutilizable, flexible y robusto.

Para la realización de la investigación se hizo necesario aplicar métodos científicos dentro de los que se incluyen **métodos teóricos y empíricos**.

Métodos teóricos:

- **Analítico-Sintético**, se aplicó con el objetivo de analizar y aumentar los conocimientos entorno al objeto de estudio mediante la consulta de

bibliografía científica, para después, haciendo uso de la síntesis, lograr resumir y exponer los resultados obtenidos del análisis.

- **Histórico-Lógico**, se utilizó para analizar a nivel internacional, las empresas que utilizan software para el análisis de procesos petrofísicos en pozos petroleros.
- **Inductivo-Deductivo**, se aplicó con el objetivo de definir los procesos de análisis petrofísicos de manera general hasta llegar a las especificidades de cada software utilizado por CEINPET.
- **Modelación**, se empleó para modelar la Arquitectura de Software propuesta.

Métodos empíricos:

- **Observación**, se empleó con el objetivo de obtener un registro visual de toda la información referente a los procesos de análisis petrofísicos y el funcionamiento del centro en una situación real.
- **Entrevista,** se utilizó con el objetivo de obtener información confiable y real de las necesidades del centro.

Como **posibles resultados** se espera:

- Diseño de la arquitectura de software que soporte los nuevos requerimientos del sistema.
- Documentos de los artefactos Ingenieriles.

El documento de la investigación se encuentra estructurado por la presente introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas, anexos y un glosario donde se expresa la terminología de algunas palabras cuyo significado resulta necesario para la comprensión de las ideas expuestas en el presente documento.

Capítulo 1. Fundamentación teórica.

En este capítulo se darán a conocer conceptos asociados al dominio del problema, se describirá la situación problemática, el contexto donde se enmarca el objeto de estudio y se conocerá el estado del arte de empresas y aplicaciones informáticas asociadas a la petrofísica.

Capítulo 2. Tendencias y tecnologías actuales a desarrollar.

En este capítulo se plantean conceptos relacionados con arquitectura, estilos, y patrones, se dan a conocer las ventajas y desventajas de los estilos de Llamada y Retorno, se define el lenguaje de programación y su vinculación con el framework, se realiza un análisis de las metodologías de desarrollo y se fundamenta la utilización de las herramientas.

Capítulo 3. Presentación de la solución propuesta.

En esta sección se definen los requisitos no funcionales del sistema y se describe la propuesta de la arquitectura basada en las 4+1 Vistas de la Arquitectura de Software. Además se valida la arquitectura propuesta para detectar riesgos.

CAPÍTULO 1: Fundamentación teórica.

1.1 Introducción

Los hidrocarburos están formados por carbono, hidrógeno, oxígeno, nitrógeno y azufre. La composición media del petróleo sería 85%“C”, 12%”H” y 3% “S+O+N”. Del petróleo se fabrica una variedad de productos químicos como la turbosina que es la gasolina utilizada en aviones jet, la gasolina de aviación para uso de aviones con motores de combustión interna, el diesel de uso común en camiones y guaguas y el asfalto en la construcción.

Por todo lo anteriormente expuesto, es irrefutable la suma importancia que este hidrocarburo tiene a nivel mundial, es por esto que en nuestro país existe el CEINPET, que se encarga de dar respuesta a toda la actividad petrolera desde la exploración hasta la refinación. En esta institución se realizan varias actividades dentro de las que se destacan el análisis petrofísico de los registros de pozos y reservorios. La evaluación de los pozos y reservorios está estrechamente asociada con la interpretación de los registros y con el estudio y análisis en el laboratorio de los núcleos extraídos de los pozos de petróleo.

En este capítulo se abordará sobre las principales características de algunas aplicaciones que se dedican al análisis petrofísico. Además se expondrán conceptos relacionados con el dominio del problema.

1.2 Conceptos asociados al dominio del problema

Petrofísica: “es el estudio de las propiedades físicas y químicas que describen la incidencia y el comportamiento de las rocas, los sólidos y los fluidos”. **(1)**

Según el Glosario de las reservas de hidrocarburos de México se define como **núcleo**: “muestra cilíndrica de roca tomada de una formación durante la perforación, para determinar su permeabilidad, porosidad, saturación de hidrocarburos, y otras propiedades asociadas a la productividad”. **(14)**

Núcleos de pozos: “son muestras de las formaciones rocosas que se extraen puntualmente en los pozos. El estudio y análisis de estos se realizan en laboratorios”. **(38)**

Figura 1 Núcleos de Pozos.

Registro de pozos: "son mediciones que se realizan con sensores remotos constituidos por buzo, cable y registrador de superficie. Estos registros se relacionan con las propiedades físicas, mecánicas, eléctricas y radiactivas de las secuencias de rocas que descubre un pozo". **(38)**

Según el Glosario de las reservas de hidrocarburos de México se define como **registro de pozo:** "representa la información sobre las formaciones del subsuelo obtenidas por medio de herramientas que se introducen en los pozos, y son de tipo eléctrico, acústico y radioactivo. El registro también incluye información de perforación y análisis de lodo y recortes, de núcleos y pruebas de formación". **(14)**

Flujo de fluido: "es el movimiento de fluido por poros y fracturas dentro de rocas permeables en un reservorio. Generalmente, el flujo de fluido sigue la Ley de Darcy[1], entonces el mismo puede ser simulado con un modelo del reservorio". **(2)**

Porosidad: "es una medida de la capacidad de almacenamiento de fluidos que posee una roca y se define como la fracción del volumen total de la roca que corresponde a espacios que pueden almacenar fluidos". **(3)**

Porosidad: "fracción de una roca que es ocupada por poros". **(39)**

Permeabilidad: "es la medición de la capacidad de una roca para transmitir fluidos. Para que sea permeable, la roca debe tener poros o fracturas interconectados, por lo tanto, hay una relación general entre la porosidad y la permeabilidad". **(1)**

[1] El agua se mueve a través de un medio poroso en forma líquida, en respuesta al gradiente hidráulico.

Según el Glosario de las reservas de hidrocarburos de México se define como **permeabilidad**: “facilidad de una roca para dejar pasar fluidos a través de ella. Es un factor que indica si un yacimiento es, o no, de buenas características productoras”. **14)**

1.2.1 Clasificación de los formatos de registros de pozos.

Los datos digitales representados en estos registros se pueden clasificar en dos categorías principales: Código Estándar Americano para el Intercambio de Información (ASCII, por sus siglas en inglés) y Binario. **(4)**

1- ASCII

El formato Log ASCII Standard (LAS), comprende archivo de datos individuales escritos en ASCII. Representa el encabezado del registro de pozo y las curvas ópticas en forma digital. . El .LAS es un formato estándar para la industria petrolera que sirve para almacenar la información de los registros de pozos. Estos ficheros están estructurados por secciones:

~V - Versión: Datos de la versión del archivo.

~W - Well: Datos de identificación del pozo.

~C - Curve: Identificación de los diferentes perfiles registrados.

~P - Parameter: Parámetros y Constantes.

~O - Other: Otra información de importancia.

~A - ASCII: Contiene los datos de las lecturas de los registros de cada intervalo.

```
~VERSION INFORMATION
VERS.           2.0   :CWLS Log ASCII Standard - VERSION 2.0
WRAP.           NO    :One Line per depth step
DLM.          SPACE   :Column Data Section Delimiter Character
PROD.  Schlumberger   :LAS Producer
PROG.  DLIS to ASCII 14C0-302                        :LAS Program name and version
CREA.        2007/10/15 12:06                        :LAS Creation date {YYYY/MM/DD hh:mm}
DLIS_CREA.  2007-Oct-15 11:29                        :DLIS Creation date and time {YYYY-MMM-DD hh:mm}
SOURCE.     Canasi-100_(OH)_FMI_NGS_HRLA_TLD_15October2007.DLIS  :DLIS File Name
FILE-ID.     FMI_NGS_HRLA_TLD_048PUP                 :File Identification Number
#-------------------------------------------------
~WELL INFORMATION
#MNEM.UNIT      DATA             DESCRIPTION
#---- ------ --------------   -----------------------------
STRT .M      1613.5          :START DEPTH
STOP .M      3275.5          :STOP DEPTH
STEP .M       0.5            :STEP
NULL .          -999.25      :NULL VALUE
COMP .        SHERRITT INT. OIL & GAS                :COMPANY
WELL .        Pozo                                   :WELL
#-------------------------------------------------------------------------------
~PARAMETER INFORMATION
#MNEM.UNIT    VALUE                    DESCRIPTION
#---- -----   ---------------------    -------------------------
BSAL .PPM     7900.00000               :Borehole Salinity
MRT  .DEGF     139.99992               :Maximum Recorded Temperature
RMB  .OHMM       0.10791               :Resistivity of Mud - BHT
#-------------------------------------------------------------------------------
~CURVE INFORMATION
#MNEM.UNIT   API CODE                                DESCRIPTION
#---- -----  --------                                -----------------------
DEPT .M                                              :DEPTH (BOREHOLE) {F10.1}
DEVI .DEG                                            :Hole Deviation {F13.4}
HAZI .DEG                                            :Hole Azimuth {F13.4}
CGR  .GAPI                                           :Computed Gamma Ray {F13.2}
POTA .                                               :Potassium {F13.4}
SGR  .GAPI                                           :Gamma Ray {F13.2}
THOR .PPM                                            :Thorium {F13.4}
URAN .PPM                                            :Uranium {F13.4}
HTEM .DEGC                                           :HTC Temperature {F13.2}
GR   .GAPI                                           :Gamma-Ray {F13.2}
#-------------------------------------------------------------------------------
#
#     DEPT        DEVI         HAZI         CGR          POTA         SGR          THOR         URAN
#
~Ascii | Curve
    1613.5      63.6953     160.0896       83.65       0.0429       41.67       3.9336      -5.5657
    1614.0      63.6953     160.0896       83.65       0.0429       41.67       3.9336      -5.5657
    1614.5      63.6953     160.0896       83.65       0.0429       41.67       3.9336      -5.5657
    1615.0      63.6953     160.0896       83.65       0.0429       41.67       3.9336      -5.5657
    1615.5      63.6953     160.0896       83.65       0.0429       41.67       3.9336      -5.5657
```

Figura 2 Archivo Log ASCII Standard (.LAS)

2- Binario

El formato de datos binarios Log Information Standard (LIS) fue producido a partir de los sistemas de adquisición de Schlumberger[2]. Era el formato convencional de datos dentro de la industria del perfilaje hasta que fue superado por el Digital Log Interchange Standard (DLIS).

La información que contienen los registros de pozos, son interpretados por aplicaciones informáticas especializados que permiten la visualización de las curvas que representan las propiedades físicas.

1.3 Objeto de estudio

Durante la perforación de los pozos petroleros, tanto en la etapa de exploración como en la parte de perforación, el proceso de análisis petrofísico a partir de la lectura de los registros de pozos es importante para los especialistas del centro ya que a partir de esta lectura se obtienen propiedades físicas de las rocas como: la porosidad, la saturación de agua, la permeabilidad, hasta el análisis en laboratorio de los núcleos de

[2] Es una de las empresas que brinda servicios a la industria petrolera, es una de las empresas líder en este sector.

pozos. Todo esto conlleva a que se realice de forma más eficiente la exploración y la posterior explotación de los yacimientos de hidrocarburos.

1.3.1 Descripción general

Cuba cuenta con un centro avalado por una alta calificación y que acumula una gran experiencia de trabajo, este se encarga de dar respuesta de forma integral a toda la actividad petrolera, desde la exploración hasta la refinación, en el proceso de investigación.

Todo el proceso de análisis petrofísico que se realiza en el CEINPET es a través de estudios en laboratorios de núcleos y registros de pozos. Los núcleos son muestras de las formaciones rocosas que se extraen puntualmente en los pozos y los registros representan la información sobre las formaciones del subsuelo obtenidas por medio de herramientas que se introducen en los pozos. De la interpretación de los registros de pozos se obtienen valores de las propiedades físicas de las rocas como la permeabilidad, la saturación del agua, la estructura y la porosidad.

El CEINPET es un centro de alto reconocimiento y prestigio a nivel nacional, este se ha servido del desarrollo mundial de las tecnologías y ha adquirido un conjunto de software propietarios que garantizan la interpretación de los registros de pozos de forma eficiente, pero para disponer de estos sistemas y de sus actualizaciones hay que pagar grandes cantidades de dinero.

1.3.2 Descripción actual del dominio del problema

Actualmente el CEINPET realiza el proceso de análisis petrofísico a través de software propietarios reconocidos mundialmente pues no cuenta con un software netamente cubano para este análisis, estos sistemas informáticos no están desarrollados específicamente para las necesidades del centro por lo que el mismo se encuentra enfrascado en la búsqueda de otras soluciones que satisfagan las necesidades de sus trabajadores.

Un software utilizado por el centro es el Hidrocarbure Data Systems (**HDS**), en el mismo los trabajadores tienen una experiencia avanzada y hasta el momento los resultados son buenos, la función primaria del software es interpretar las perforaciones y en algunos casos los registros eléctricos de las perforaciones usando cualquiera de las unidades de registro existentes.

Este software es capaz de recibir y procesar los datos tanto de registros de pozo como de terreno, permitiendo al usuario identificar áreas de saturación de hidrocarburos y el cálculo de parámetros petrofísicos.

Tiene como inconveniente que se tiene una versión vieja ya que las actualizaciones son caras, otra dificultad es que no cuenta con un servicio que permita cargar diferentes registros de pozos a la vez, para que los trabajadores del CEINPET puedan hacer una comparación de los mismos, al ser propietario cuenta con una llave única para su uso restringiendo su utilización solamente en una computadora a la vez.

Otro software utilizado es el Interactive Petrofísica (**IP**) (Interactive Petrophysics), el mismo usa los modelos deterministas y probabilísticos para calcular la porosidad, la saturación de agua, los volúmenes de sales y otras propiedades. Como beneficios se encuentran: simplificación de los flujos de trabajo utilizando modelos de interpretación, edición de datos completos y soluciones petrofísicas “todo en un paquete de PC” y vincular a muchas fuentes de datos en tiempo real para las decisiones de interpretación instantánea. Otra ventaja que brinda es cargar diferentes registros de pozos lo que posibilita una comparación más cómoda de estos.

Entre las características comunes que poseen estos sistemas se encuentran:

- Importación de archivos .LAS.
- Representación de las curvas de los registros originales y los resultados de la interpretación.
- Introducción de fórmulas y ecuaciones propuestas por el usuario.
- Gráficos de propiedades cruzadas y correlaciones entre pozos.

1.3.3 Situación problemática

El CEINPET se encarga de dar respuesta a toda la actividad petrolera de Cuba, dentro de la misma se encuentra el análisis petrofísico, que se encarga del estudio de las propiedades físicas de las rocas a través de análisis en los registros de pozos.

El análisis petrofísico actualmente se realiza con productos propietarios, dentro de los cuales se destacan el **HDS** y el **IP**, que aunque realizan de forma eficiente el análisis petrofísico, implican el pago de costosas licencias para su uso.

Estos productos propietarios son elaborados para empresas que se dedican a la investigación del petróleo y dentro de esta al análisis petrofísico, lo que se aleja de las necesidades reales de los trabajadores del CEINPET y en muchas ocasiones se

necesita una experiencia avanzada en la utilización de los mismos. Otro inconveniente es que luego de ser adquiridos no se puede acceder a su código fuente para una posible mejora.

Para poder obtener las actualizaciones de estos productos utilizados en el centro se hace necesario pagar una nueva suma de dinero lo que constituye un obstáculo para la institución debido a la implicación económica que esto representa y debido al bloqueo en muchas ocasiones se realiza por terceras personas disminuyendo la probabilidad de obtener un producto fiable.

En aras de revertir lo expresado anteriormente, un equipo de desarrollo de la facultad 9 de la UCI, se encuentra enfrascado en el desarrollo de un software para el análisis petrofísico para pozos petroleros y que se adapte a las necesidades de los trabajadores del CEINPET.

1.4 Análisis de otras soluciones existentes

A nivel mundial existen empresas reconocidas que se dedican a la fabricación de software para el análisis petrofísico, estos software son propietarios, por lo que la UCI se ha dado a la tarea de desarrollar un software para el CEINPET.

Entre los diferentes sistemas de software existentes se destaca **NeuraPrep,** elaborado por la empresa **Neuralog** orientado a cubrir la brecha entre la vectorización y la utilización de la información final por parte del petrofísico.

Este permite la preparación de los registros, normalización y re-escalamiento de curvas, cambio en la lectura de curvas, generación de curvas sintéticas con parámetros preestablecidos a partir de curvas originales, filtrado de curvas con alto contenido de ruido y otras funciones necesarias para el análisis petrofísico.

NeuraPrep incluye las siguientes funcionalidades para el procesamiento de datos:

- Generación de curva única de resistividad/conductividad a partir de la utilización de una curva de resistividad y una curva de conductividad, seleccionando un punto de corte por registro.
- Filtrado de curvas con alto contenido de ruido, utilizando diferentes tipos de filtros.
- Generación de curvas sintéticas a partir de curvas vectoriales con parámetros preestablecidos.

- Función para la unión de curvas de diferentes corridas de manera visual y precisa.
- Normalización de curvas.
- Corrección en profundidad de curvas.

Este software está diseñado para importar cualquier tipo de formato .LAS y puede convertirlos a un formato estándar facilitando el trabajo del usuario. **(5)**

Otro software que realiza análisis petrofísico es el **HDS**. Su misión se enmarca en ser un fuerte proveedor de software de Análisis Petrofísico de Registros, para los geólogos, ingenieros y petrofísicos. Sus productos están desarrollados sobre la plataforma Windows. **(6)**

Su principal exponente es el **HDS 2000**. La función primaria del software es interpretar las perforaciones y en algunos casos los registros eléctricos de las perforaciones usando cualquiera de las unidades de registro Métrica o Inglesa. El software es capaz de recibir y procesar los datos tanto de registros de pozo como de terreno, permitiendo al usuario identificar áreas de saturación de hidrocarburos y el cálculo de parámetros petrofísicos.

Petrolog es otro software que realiza análisis petrofísico. La versión original del mismo fue lanzada por primera vez para la industria petrolera en 1982. Desde entonces, ha sido portado a todos los sistemas operativos populares, (Unix, Linux, Windows 95/98/2000/XP). En 2006, el software ha sido portado a Microsoft Windows. NET framework usando el lenguaje C Sharp. **(7)**

Petrolog es una instancia especializada en gestión de datos de registro, puede importar y exportar datos en varios estándares de la industria incluyendo LIS y LAS.

1.5 Conclusiones parciales

Después de haber analizado el mercado fue posible percatarse de la importancia de la petrofísica ya que esto permite que se realice de forma más eficiente la exploración y la explotación de los yacimientos de hidrocarburo.

Además se esclareció que CEINPET no cuenta con un software netamente cubano que reúna las características indispensables para el análisis petrofísico y que el mismo se adapte a las necesidades de sus trabajadores haciendo el trabajo más cómodo y rápido.

CAPÍTULO 2: Tendencias y tecnologías actuales a desarrollar.

2.1 Introducción

En el presente capítulo se definirán conceptos de arquitectura, estilos y patrones. Se argumentará sobre las tecnologías y herramientas utilizadas en el desarrollo de la aplicación se analizarán las metodologías de desarrollo de software, así como el lenguaje de modelado y la herramienta CASE con la que se modelará la solución propuesta. Además se dará a conocer el lenguaje de programación utilizado y su vínculo con el framework. También se refinarán los aspectos arquitectónicos de la línea base de la arquitectura propuesta por el Ing. Raudel Chávez Arroyo en el año 2009. Línea base es una especificación o producto revisado y aprobado formalmente, que sirve como base para el desarrollo posterior, y puede ser modificado.

2.2 Arquitectura de software

Los primeros antecedentes de la Arquitectura de Software(**AS**) surgen en la década de 1960, pero esta permaneció en un estado de congelación por muchos años, hasta los principios del 1990 que se vuelve a retomar esta idea de **AS** por Dewayne Perry y Alexander Wolf. **(8)**

El término de la **AS** lleva más de una década por lo que existen varias definiciones y todavía no se ha llegado a una definición común entre la mayoría de los estudiosos del tema. La IEEE-1471-200 define: “La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.” **(9)**

Paul Clements define: “La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.” **(10)**

“La Arquitectura de Software de un sistema de programa o sistema de computación es la estructura o las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las

relaciones entre ellos.” **(11)**

Philippe Kruchten define: “La arquitectura de software, tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad, y disponibilidad”. **(12)**

Se puede decir que la **AS** es una visión abstracta de alto nivel, que propone como van a estar relacionados los componentes de software, además tiene una relación muy estrecha con los requisitos no funcionales del sistema: que expresan las propiedades que debe tener el sistema, define estilos o combinación de estos para una solución y es esencial para el éxito o el fracaso de un proyecto.

2.2.1 Estilos arquitectónicos

La aplicación de estilos arquitectónicos es primordial en el desarrollo de cualquier software. Existen diferentes conceptos asociados al tema pero es válido aclarar que ninguna definición reprocha a la otra.

Mary Shaw y Paul Clements identifican los estilos arquitectónicos como: “un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo”. **(13)**

Los estilos arquitectónicos tradicionalmente se reconocen como las cuatro “C” de la **AS:**

- Componentes (Elementos).
- Conectores.
- Configuraciones.
- Restricciones (constraints). **(10)**

2.2.1.1 Ejemplos de estilos arquitectónicos.

- **Estilos de Flujo de Datos.**
  - Tubería y filtros.
- **Estilos Centrados en Datos.**

- Arquitecturas de Pizarra o Repositorio.

- **Estilos de Llamada y Retorno.**
  - Arquitecturas en Capas.
  - Modelo Vista Controlador (MVC).
  - Arquitecturas Orientadas a Objetos.
  - Arquitecturas Basadas en Componentes.
- **Estilos de Código Móvil.**
  - Arquitectura de Máquinas Virtuales.
- **Estilos heterogéneos.**
  - Sistemas de control de procesos.
  - Arquitecturas Basadas en Atributos.
- **Estilos Peer-to-Peer.**
  - Arquitecturas Basadas en Eventos.
  - Arquitecturas Orientadas a Servicios (SOA).
  - Arquitecturas Basadas en Recursos. **(10)**

Todos los sistemas combinan uno o varios estilos por lo que el Sistema de Análisis de Registros de Pozos Petroleros (ANPER) no está exento de esto y su organización será regida por uno o varios estilos. Tomando los resultados de la línea base de la arquitectura para el Sistema de Análisis de Registros de Pozos Petroleros (ANPER) se centró el estudio en los **Estilos de Llamada y Retorno,** haciendo un estudio comparativo de los mismos.

2.2.1.2 Arquitecturas en Capas

Es 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. **(17)**

Figura 3 Arquitectura en Capas.

En este estilo por definición los niveles inferiores no pueden usar las funcionalidades ofrecida por los niveles superiores, las interacciones entre capas usualmente proceden por invocación de procedimientos y resulta útil cuando se pueden identificar distintas clases de servicios que pueden ser articulados jerárquicamente.

Por las condiciones del proyecto se ha decidido utilizar el estilo en tres capas, que resuelve el problema que si ocurre algún cambio se tendría que realizar modificaciones sólo en la capa requerida y no en todas las capas. Tiene soporte para la estandarización, ya que el cambio de nivel no afecta el resto, proporciona alta reutilización, desarrollos paralelos en cada capa, aplicaciones más robustas debido al encapsulamiento, soporta un diseño basado en niveles de abstracción crecientes, lo que permite la partición de un problema complejo en una secuencia de pasos incrementales. Además este estilo posee mayor flexibilidad lo que permite agregar nuevos módulos para dotar al sistema de nuevas funcionalidades y proporciona una alta escalabilidad para el sistema.

El estilo en tres capas permite separar la capa presentación ó interfaz de usuario (IU) de la capa de negocio y a su vez de la capa de acceso a datos.

Capa de Presentación: es la que el usuario ve, con la que interactúa, es la encargada de recoger los datos del usuario, enviárselos a la capa de negocio para que esta los procese, recibir estos resultados y mostrarlos. Esta sólo se comunica con la capa de negocio.

Capa de Negocio: es la encargada de recibir las peticiones de los usuarios y enviarlas después de su proceso. Se denomina capa de negocio (e incluso lógica de negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse.

Capa de Acceso a Datos: se centra en la manipulación y recuperación de la información, esta capa estará estrechamente vinculada con la interpretación de la información de los ficheros .LAS, además esta se encarga de la integridad de estos ficheros, su estructura y organización.

2.2.1.3 Modelo Vista Controlador (MVC)

MVC separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes. **(15)**

- Modelo. El 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).
- Vista. Maneja la visualización de la información.
- Controlador. Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

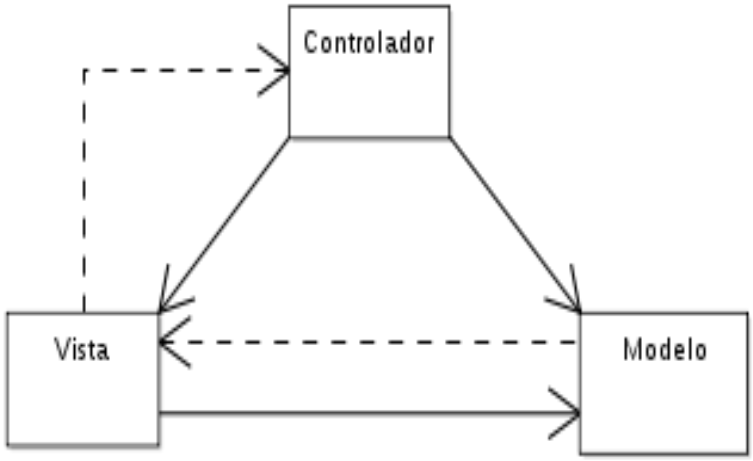


Figura 4 MVC.

Ventajas:

- Separa el modelo de la vista, la vista no depende del modelo ni el modelo de la vista.

- Soporte de vistas múltiples.
- Adaptación al cambio.

A pesar de las ventajas que ofrece este estilo, tiene desventajas en su uso, gran acoplamiento entre las vistas y el controlador con el modelo, ineficiencia en el acceso a datos desde la vista, relación estrecha entre la vista y el controlador. **(16)** Por tales razones se descarta este estilo.

2.2.1.4 Arquitecturas Orientadas a Objetos

Este estilo tiene varios nombres alternativos como: **Arquitecturas Basadas en Objetos**, **Abstracción de Datos** y **Organización Orientada a Objetos**. Sus componentes son los objetos, o más bien instancias de los tipos de datos abstractos.

Los componentes de este estilo se basan en principios **OO**: 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. **(8)**

Ventajas

- Se puede modificar la implementación de un objeto sin afectar a sus clientes.
- Es posible descomponer problemas en colecciones de agentes en interacción.
- Un objeto es ante todo una entidad reutilizable en el entorno de desarrollo.

No se utilizará este estilo ya que presenta granularidad muy fina para sistemas grandes, contiene problemas de efectos colaterales en cascada (Si se cambia un objeto, se deben modificar todos los que lo invocan) y para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad.

2.2.1.5 Arquitectura Basada en Componentes

Existen varias definiciones de componentes, pero Clemens Alden Szyperski proporciona una que es bastante operativa: “un componente de software es una unidad de composición con interfaces especificadas contractualmente y solamente dependencias explícitas de contexto. Un componente de software puede ser desplegado independientemente y está sujeto a composición por terceros”. **(17)**

Ventajas:

- Reutilización del software.
- Simplifica las pruebas. Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- Simplifica el mantenimiento del sistema. Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
- Mayor calidad. Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

Este estilo a pesar de las ventajas que brinda queda descartado ya que el sistema que se está desarrollando no tiene varios componentes y este estilo se usa cuando se tienen varios componentes hacer que se comuniquen entre sí.

2.3 Patrones de Diseño

La palabra patrón surge a finales de los 70 por Christopher Alexander quien comentaba que un patrón describe un problema que ocurre una y otra vez, pero a partir del libro “Design Patterns: Elements of Reusable Object-Oriented Software” de Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, conocidos por el grupo de los cuatro, donde se caracterizan a los patrones de diseño. **(18)**

Christopher Alexander define: “Cada patrón es una regla de tres partes, que expresa una relación entre un contexto, un problema y una solución. Como un elemento en el mundo, cada patrón es una relación entre un contexto, un sistema de fuerzas que ocurren repetidamente en ese contexto y una configuración espacial que permite que esas fuerzas se resuelvan entre sí.” **(19)**

Larman Craig en el libro UML y patrones: Introducción al análisis y programación orientada a objetos define: “Un patrón es una pareja de problema/solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas.” **(20)**

Juan José Moreno Navarro en su artículo Arquitectura de Software: Curso de Software basado en Componentes define que: “Un patrón es una solución probada que se puede aplicar con éxito a un determinado tipo de problemas que aparecen

repetidamente en algún campo.” **(21)**

Con estos conceptos se puede decir que un patrón deberá tener un nombre, saber el problema que va a resolver, que sea un concepto ya probado, las consecuencias buenas y malas que pueden traer al usarlo.

Dentro de los patrones que se utilizarán están los patrones GRASP[3] que describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones, estos son muy importantes ya que se refieren a cuestiones muy básicas y aspectos fundamentales de diseño.

2.3.1 Patrón Creador

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos este patrón propone que clase será la responsable de crear una instancia de otra clase.

2.3.2 Patrón Experto

Es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Permite asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. **(23)**

Ventajas

- Se conserva el encapsulamiento de la información, ya que los objetos utilizan su propia información para llevar a cabo las tareas. Normalmente, esto conlleva un bajo acoplamiento, lo que da lugar a sistemas más robustos y más fáciles de mantener.
- Se distribuye el comportamiento entre las clases que contienen la información requerida, por lo tanto, se estimula las definiciones de clases más cohesivas y “ligeras” que son más fáciles de entender y mantener. **(20)**

2.3.3 Patrón Bajo Acoplamiento

Este patrón asigna una responsabilidad para mantener el bajo acoplamiento que no es más que tratar de que una clase no dependa de muchas otras, así esa clase no tendrá muchas dependencias, facilitando la reutilización.

[3] Patrones de Principios Generales para Asignar Responsabilidades.

Este patrón no puede verse separado del patrón Experto y del patrón Alta Cohesión.

Ventajas

- No afectan los cambios en otros componentes.
- Fácil de entender de manera aislada.
- Conveniente para reutilizar. **(20)**

2.3.4 Patrón Alta Cohesión

Este patrón define que una clase tiene responsabilidades moderadas en un área funcional y colabora con otras clases para llevar a cabo las tareas.

Una clase con alta cohesión tienen un número relativamente pequeño de métodos, con funcionalidad altamente relacionada, y no realiza mucho trabajo. Colabora con otros objetos para compartir el esfuerzo si la tarea es extensa. Además es ventajosa porque es relativamente fácil de mantener, entender y reutilizar.

Ventajas

- Se incrementa la claridad y facilita la comprensión del diseño.
- Se simplifican el mantenimiento y las mejoras. **(20)**

Como patrones GOF se utilizarán:

2.3.5 Patrón Singleton

Este patrón garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella.

Es recomendable usarlo cuando:

- Deba haber exactamente una instancia de una clase y esta deba ser accesible a los clientes desde un punto de acceso conocido.
- La única instancia debería ser extensible mediante herencia y los clientes deberían ser capaces de utilizar una instancia extendida sin modificar su código.

Ventajas

- Acceso controlado a la única instancia. **(24)**

2.3.6 Patrón Fachada (Facade)

Proporcionar una interfaz unificada para un conjunto de interfaces en un subsistema, haciéndolo más fácil de usar.

Ventajas

- Oculta a los clientes de la complejidad del subsistema y lo hace más fácil de usar. **(25)**

2.4 Framework

Framework: Es un conjunto de clases que cooperan y forman un diseño reutilizable para un tipo específico de software que brinda una guía arquitectónica que parte el diseño en clases abstractas y define sus responsabilidades y colaboraciones. **(26)**

Framework: es una infraestructura software que crea un entorno común para integrar aplicaciones e información compartida dentro de un dominio dado. **(26)**

En resumen, se puede decir entonces que: un framework para el desarrollo es un conjunto de clases que cooperan y forman un diseño reutilizable formando una infraestructura que facilita y agiliza el desarrollo de las aplicaciones. **(26)**

A partir del estilo seleccionado se ha decido utilizar el framework 3.5 Net, es un componente integral de Windows que admite la creación y la ejecución de la siguiente generación de aplicaciones y servicios Web XML. El diseño de .NET Framework está enfocado a cumplir los objetivos siguientes:

- Proporcionar un entorno coherente de programación orientada a objetos, en el que el código de los objetos se pueda almacenar y ejecutar de forma local pero distribuida en Internet o ejecutar de forma remota.
- Proporcionar un entorno de ejecución de código que reduzca lo máximo posible la implementación de software y los conflictos de versiones.
- Ofrecer un entorno de ejecución de código que fomente la ejecución segura del mismo, incluso del creado por terceras personas desconocidas o que no son de plena confianza.
- Proporcionar un entorno de ejecución de código que elimine los problemas de rendimiento de los entornos en los que se utilizan secuencias de comandos o intérpretes de comandos.

- Ofrecer al programador una experiencia coherente entre tipos de aplicaciones muy diferentes, como las basadas en Windows o en el Web.

- Basar toda la comunicación en estándares del sector para asegurar que el código de .NET Framework se puede integrar con otros tipos de código. **(27)**

Para el estilo en capas definido anteriormente se ha decidido proponer el uso del framework .NET par la implementación de cada una de las capas.

Capa de Presentación o Interfaz de Usuario (IU): se utilizará la biblioteca de clases **System.Windows.Forms** que contiene las clases necesarias para crear aplicaciones basadas en formularios y ventanas de Windows. Entre estas clases se pueden encontrar formularios, cuadros de diálogo y controles gráficos necesarios para construir una interfaz de usuario rica en elementos visuales que permita una mejor interacción con la aplicación.

Capa de Negocio: El framework 3.5 Net contiene dos componentes principales: Lenguaje Común de Ejecución (CLR por sus siglas en inglés) y las bibliotecas de clases de .Net Framework. CLR administra la memoria, ejecución de subprocesos, ejecución de código, comprobación de la seguridad del código, compilación y demás servicios del sistema; y las bibliotecas de clases de .NET Framework es una colección de tipos reutilizables que se integran estrechamente con el CLR. Las bibliotecas de clases permiten la administración de cadenas, recolección de datos, conectividad de bases de datos y acceso a archivos. En esta capa se utilizará la biblioteca de clases **System.Drawing** que contiene las clases necesarias para dibujar gráficas. Entre estas clases podemos encontrar además de gráficos, imagen, mapa de bit, pinceles y fuente que define un formato concreto para el texto, incluyendo tipo de letra, tamaño y atributos de estilo. Además se utilizará la biblioteca de clases **Exception** que representa los errores que se producen durante la ejecución de la aplicación.

Capa de Acceso a Datos: se trabajará considerablemente con el manejo de ficheros para ello se utilizará la biblioteca de clases **System.IO** que contiene tipos que permiten la lectura y escritura de archivos. Entre las clases que se utilizarán están StreamReader para el proceso de importación y StreamWriter para el proceso de exportación. Además se utilizará la biblioteca de clases **Exception.**

2.4.1 Lenguajes de programación

Java

Este lenguaje de programación es libre y entre sus características se encuentra:

Orientado a Objeto: fue diseñado como un lenguaje orientado a objetos desde sus inicios. Los objetos agrupan en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos.

Distribuido: proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

Indiferente a la Arquitectura: está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows .NET, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos.

Portable: la indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas.

Dinámico: el lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.

Applets: Java puede ser usado para crear dos tipos de programas: aplicaciones independientes[4] y Applets[5]. **(28)**

Robusto: realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Maneja la memoria

[4] Las aplicaciones independientes se comportan como cualquier otro programa escrito en cualquier lenguaje.

[5] Las applets son pequeños programas que aparecen embebidos en las páginas Web, como aparecen los gráficos o el texto, pero con la capacidad de ejecutar acciones muy complejas, como animar imágenes, establecer conexiones de red, presentar menús y cuadros de diálogo para luego emprender acciones.

para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria. **(29)**

C Sharp (C#)

Los creadores de este lenguaje son Scott Wiltamuth y Anders Hejlsberg. No se puede decir las características de este lenguaje sin mencionar algunas de las características de la plataforma .Net ya que la misma tiene una implicación directa con este lenguaje.

Sencillez: elimina muchos elementos que otros lenguajes incluyen y que son innecesarios en .NET. Por ejemplo:

El tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile (no como en C++), lo que facilita la portabilidad del código.

Orientado a objetos: soporta todas las características propias del paradigma de programación orientada a objetos: encapsulación, herencia y polimorfismo.

Seguridad de tipos: incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente, lo que evita que se produzcan errores difíciles de detectar por acceso a memoria no perteneciente a ningún objeto.

Eficiente: en C# todo el código incluye numerosas restricciones para su seguridad y no permite el uso de punteros. Sin embargo, y a diferencia de Java, en C# es posible saltarse dichas restricciones manipulando objetos a través de punteros. Para ello basta marcar regiones de código como inseguras (modificador unsafe) y podrán usarse en ellas punteros de forma similar a cómo se hace en C++, lo que puede resultar vital para situaciones donde se necesite una eficiencia y velocidad de procesamiento grandes.

Extensibilidad de operadores: para facilitar la legibilidad del código y conseguir que los nuevos tipos de datos básicos que se definan a través de las estructuras estén al mismo nivel que los básicos predefinidos en el lenguaje, al igual que C++ y a diferencia de Java, C# permite redefinir el significado de la mayoría de los operadores incluidos los de conversión, tanto para conversiones implícitas como explícitas cuando se apliquen a diferentes tipos de objetos. **(30)**

2.5 Selección de la metodología adecuada

Para la selección de una metodología adecuada es necesario hacer una breve comparación de las Metodologías Ágiles y las Metodologías Tradicionales, esta breve comparación se establece en la siguiente tabla.

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el Proyecto.	Cierta resistencia a los cambios.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas Políticas/normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles.
Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos.

Tabla 1 Diferencias entre metodologías ágiles y las tradicionales.

Según la comparación establecida se ha decidido que la **AS** para el Sistema de Análisis de Registros de Pozos Petroleros (ANPER) estará regida por una metodología tradicional ya que los requerimientos del mismo no son variables, por tal razón se hace necesario que las decisiones sean tomadas al inicio del proyecto para poder hacer entrega de un producto que cumpla con las funcionalidades deseadas. Proponen abundante documentación, esto posibilita aprovechar los elementos reutilizables que sean necesarios utilizar en otras aplicaciones. Al proyecto contar con una planificación de tareas y artefactos que deben ser generados en cada una de las fases por la que transite el producto no es necesario que el cliente sea parte del equipo de desarrollo,

basta con planificar reuniones entre las dos partes para esclarecer las dudas que vayan surgiendo en el desarrollo de dicha aplicación. Además definen que la **AS** es esencial y se expresa a través de modelos, esto ayudaría a desarrollar una arquitectura robusta y flexible y con los modelos ayudaría a que el grupo de desarrollo tenga una visión de los elementos arquitectónicamente significativos.

Entre las metodologías tradicionales existentes hemos decidido utilizar RUP[6] ya que es en la actualidad uno de los más usados por las empresas de software y es validado continuamente para el perfeccionamiento de su uso. Por los excelentes resultados obtenidos en la UCI con el uso de esta metodología. Está concebido para que desde el inicio del proceso, se establezca una definición acertada del proyecto, haciendo innecesarias las reconstrucciones parciales posteriores. Además está dirigido a la programación orientada a objetos que permite obtener sistemas escalables en el tiempo que no necesitarán grandes inversiones de recursos en sus modificaciones posteriores.

RUP es un proceso formal: Provee un acercamiento disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad que satisfaga los requerimientos de los usuarios finales. Utiliza UML como lenguaje de notación, fue desarrollado por Rational Software, y está integrado con toda la Suite Rational de Herramientas. **(31)**

Los principales elementos de RUP son:

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

En RUP se han definido las actividades en grupos lógicos definiéndose nueve flujos de trabajo principales, los seis primeros son conocidos como flujos de ingeniería[7] y los tres últimos como de apoyo[8].

RUP presenta 4 fases por las cuales transita el software, estas son:

- Inicio o Conceptualización.

[6] Proceso Unificado de Desarrollo.
[7] Modelamiento del Negocio, Requerimiento, Análisis y Diseño, Implementación, Prueba e Instalación.
[8] Administración de Configuración, Administración del Proyecto y Cambio y Ambiente.

- Elaboración.
- Construcción.
- Transición.

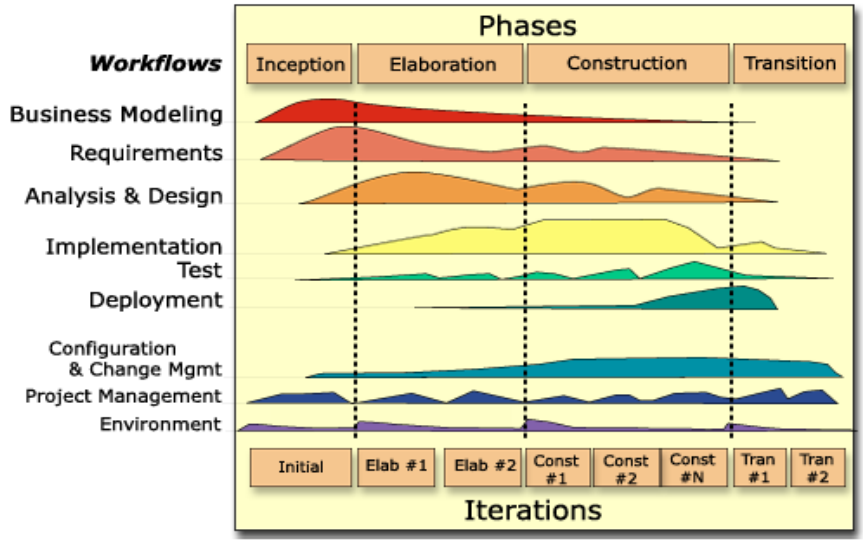


Figura 5 Fases de RUP.

Además RUP es una metodología muy importante ya que presenta las siguientes características:

- Guiada por casos de usos.
- Centrada en la arquitectura.
- Iterativo e incremental.

2.6 Selección del lenguaje de modelación

Para el modelado se utilizará UML[9] siguiendo la metodología RUP. UML es el lenguaje de modelado más conocido y usado en la actualidad, está respaldado por OMG (Object Management Group), diseñado para visualizar, especificar, construir y documentar software orientado a objeto. **(32)**

UML no es un método, supone una abstracción de un sistema para llegar a construirlo en términos concretos, divide cada proyecto en un número de diagramas que representan las distintas vistas del proyecto y juntos representan la arquitectura del mismo. **(33)**

[9] Lenguaje Unificado de Modelado.

Es importante decir que UML es un lenguaje, el más estandarizado por la industria diseñado por los autores Ivar Jacobson, Grady Booch y James Rumbaungh, este se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware y organizaciones del mundo real. **(34)**

UML define una notación que se expresa como diagramas que sirven para representar modelos/subsistemas o parte de ellos, define una estructura para ir del análisis al diseño y de este a la implementación. **(35)**

UML está compuesto por **elementos**, **diagramas** y **relaciones**, estas relaciones pueden ser de: dependencia, asociación, generalización-especialización y realización.

En general se puede decir que un modelo UML describe lo que supuestamente hará un sistema pero no dice como implementarlo.

2.7 Herramienta CASE para el modelado con UML

Dentro de las herramientas CASE[10] más populares para el modelado con UML, para la creación de distintos tipos de diagramas se encuentran el Rational Rose y el Visual Paradigm for UML.

Para el modelado con UML utilizaremos la herramienta CASE Visual Paradigm for UML ya que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Entre sus características se encuentra:

- Soporte UML versión 2.1.
- Ingeniería de ida y vuelta.
- Ingeniería inversa.
- Importación y exportación de ficheros XML.
- Editor de Detalles de Casos de Uso- Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Generador de informes para generación de documentación.

[10] Ingeniería de Software Asistida por Computadora.

- Integración con Visio-Dibujo de diagramas UML con plantillas de MS Visio Editor de figuras. **(36)**

Además de todas esas características es necesario resaltar que el Visual Paradigm for UML es una herramienta fácil de utilizar ya que es intuitiva para el usuario y es multiplataforma.

2.8 Otras herramientas

Por la utilización del framework 3.5 Net y el lenguaje de programación C# se ha decidido utilizar el IDE[11] Visual Studio este es una completa suite de herramientas para la formación de aplicaciones de escritorio y aplicaciones empresariales basadas en Web. Este es un IDE que permite el desarrollo ágil de aplicaciones como el ya antes conocido Borlan Delphi.

Lenguajes soportados:

- C/C++
- C#
- VB.net
- XAML
- HTML/CSS
- Java script
- ASP.net
- XML/XSLT

Sistemas operativos:

- Windows

Entre otras características se encuentran:

- **Completamiento de código:** Rápido completamiento de código, aunque menos opciones a la vez por lo que hay que usar un cursor para moverse.
- **Editor de código fuente:** La edición es simple y fácil. El cambio entre el diseñador de formularios y el editor de código es el doble clic en un control.
- **Interfaz gráfica de usuario y herramientas RAD[12]:** Excelente editor de Formularios de Windows.

[11] Entorno de Desarrollo Integrado.
[12] Desarrollo Rápido de Aplicaciones.

Por razones externas al proyecto, expresadas en la necesidad de desarrollar un producto de alta calidad en un tiempo mínimo, se escogió como lenguaje de desarrollo C#, con Visual Studio como IDE y framework .Net. Se plantean a continuación una serie de indicadores que avalan el uso de esta tecnología.

1. El lenguaje es impartido por la docencia de la universidad, por lo que no se empleó tiempo por concepto de capacitación.

2. Tiempos de desarrollos mínimos debido al dominio medio sobre el lenguaje, IDE y framework.

3. A pesar de haber escogido software propietario para el desarrollo de la aplicación, se piensa la posibilidad de migrar en un futuro a software libre, es por eso que se escogió esta tecnología basados en la capacidad de migración del lenguaje y del framework a la Plataforma Mono**.**

2.9 Conclusiones parciales

En este capítulo se realizó un análisis de las tecnologías y herramientas a utilizar para la construcción y desarrollo del sistema a implementar. Se fundamentó la selección del estilo seleccionado ya que brinda como ventaja fundamental el desarrollo en paralelo, así como los patrones que regirán el diseño y las responsabilidades de las clases. Se deja claro que aunque se utilizará como IDE Visual Studio esto no tendrá grandes problemas en un futuro ya que se podrá migrar hacia la Plataforma Mono. Además la construcción del software estará regida por una metodología tradicional, específicamente RUP, ya que está concebido para que desde el inicio del proceso, se establezca una definición acertada del proyecto, haciendo innecesarias las reconstrucciones parciales posteriores.

CAPÍTULO 3: Presentación de la solución propuesta.

3.1 Introducción

En este capítulo se abordará el concepto de requerimiento no funcional, así como los diferentes requerimientos que el sistema debe soportar. Además se tiene como objetivo principal la comprensión arquitectónica global del sistema, utilizando para ello las vistas arquitectónicas definidas por la metodología RUP, las mismas facilitan la comprensión de la arquitectura propuesta y aportan argumentos para la identificación de los elementos claves del sistema en desarrollo, proporcionando información al cliente sobre las decisiones arquitectónicas en la construcción del producto.

3.2 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, por ejemplo, pudiera desearse que el sistema responda dentro de un intervalo de tiempo especificado o que obtenga los resultados de los cálculos con un nivel de precisión dado. En muchos casos los requisitos no funcionales son fundamentales en el éxito del producto. Normalmente están vinculados a requisitos funcionales, es decir una vez que se conozca lo que el sistema debe hacer podemos determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser.

Los requisitos no funcionales forman una parte significativa de la especificación. Son importantes para que clientes y usuarios puedan valorar las características no funcionales del producto, pues si se conoce que el mismo cumple con toda la funcionalidad requerida, las propiedades no funcionales, como cuán usable, seguro, conveniente y agradable, pueden marcar la diferencia entre un producto bien aceptado y uno con poca aceptación. **(37)**

3.2.1 Requerimientos de Hardware.

Estaciones de Trabajo

Para nivel básico:

- Periféricos Teclado y Mouse PS2 o USB.
- 512 MB de memoria RAM.
- 1 GB de espacio libre en disco.

- CPU Pentium IV a 1.00 GHz.

Para nivel medio:

- Periféricos Teclado y Mouse PS2 o USB.
- 1 GB de memoria RAM.
- De 5 a 10 GB de espacio libre en disco.
- CPU Pentium IV a 2.00 GHz.

Para nivel óptimo:

- Periféricos Teclado y Mouse PS2 o USB.
- 2 GB de memoria RAM.
- De 10 a 20 GB de espacio libre en disco.
- CPU Pentium IV por encima de 2.00 GHz.

3.2.2 Requerimientos de Software.

Estaciones de Trabajo

- Se utilizará como Sistema Operativo, Windows 2000/XP/Vista.
- Se utilizará el Framework 3.5 Net.

3.2.3 Usabilidad, Eficiencia

- Será usable por cualquier tipo de usuario con experiencia, básica, media o avanzada en los procesos de análisis petrofísico.
- El sistema debe permitir que el usuario configure su entorno de trabajo.
- Se mostrará la información de forma lógica y correctamente estructurada.
- **Rendimiento:** El sistema debe garantizar un tiempo de respuesta de 2 a 4 segundos en dependencia de la complejidad del servicio.

3.2.4 Seguridad

- Se aplicarán las reglas de la “programación segura”, mediante el tratamiento de excepciones.

- El sistema permitirá además la verificación sobre acciones irreversibles; es decir, se le solicitará al usuario la confirmación al realizar operaciones como la eliminación.

- **Disponibilidad:** la aplicación debe estar disponible en todo momento para todos los trabajadores que intervengan en el proceso de análisis petrofísico mientras no haya un fallo en el sistema o un fallo en el fluido eléctrico.

3.2.5 Ayuda

- El sistema debe contar con un manual de usuario.
- El sistema debe garantizar que la opción de Ayuda esté visible todo el tiempo, para que el usuario la pueda consultar en tiempo de ejecución. (Esta ayuda debe tener aplicación en textos basados en ejemplos de imágenes).
- Desde cada parte del sistema se podrá acceder a la Ayuda.

3.2.6 Restricciones de acuerdo a la estrategia de diseño.

- El diseño de los componentes y el sistema en general se realizará bajo los principios y uso de la Programación Orientada a Objetos (**POO**).
- Se utilizará las librerías implícita en el framework 3.5 Net seleccionados para cada una de las capas del sistema.
  - ✓ Capa de Presentación: **System.Windows.Forms**.
  - ✓ Capa de Negocio: **System.Drawing**, **Exception.**
  - ✓ Capa de Acceso a Datos: **System.IO, Exception.**

3.3 Vistas Arquitectónicas

La representación de la **AS** se hará utilizando las vistas definidas por RUP:

Figura 6 Modelo 4+1 Vistas.

Estas vistas serán representadas en UML utilizando Visual Paradigm for UML como herramienta Case.

3.3.1 Vista de Casos de Uso

La vista de casos de uso o de escenarios es un extracto del modelo de casos de usos del sistema. La vista de casos de uso es igual al modelo de casos de uso, la única diferencia es que la vista de la arquitectura sólo tiene aquellos casos de usos y actores que son arquitectónicamente significativos mientras que el modelo de casos de uso tienen todos los casos de usos del sistema.

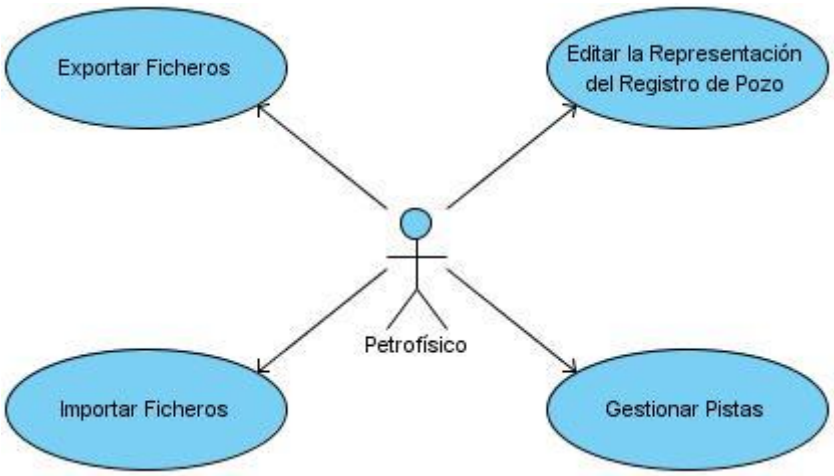


Figura 7 Vista de Casos de Uso.

2.2.1.6 Importar ficheros

Este caso de uso es el encargado de cargar los ficheros de tipo .LAS o .TXT y mostrar su información.

2.2.1.7 Exportar ficheros

Este caso de uso es el encargado de exportar los ficheros .LAS en su versión 3.0 o de llevar de las versiones 1.2 y 2.0 a la 3.0.

2.2.1.8 Editar la representación del registro de pozo

Realiza el estudio de la representación de los registros de pozos en vista a obtener las evaluaciones de las formaciones, permitiéndole al usuario la modificación de parámetros dentro de la misma.

2.2.1.9 Gestionar pistas

Gestiona todo lo referente a las pistas para después realizar el estudio de la representación de los registros de pozos.

3.3.2 Vista Lógica

La vista lógica apoya principalmente los requisitos funcionales, lo que el sistema debe brindar en términos de servicios a sus usuarios. Esta vista describe las clases más importantes, su organización en paquetes y subsistemas como las relaciones que existen entre ellos.

La siguiente figura ilustra la división del sistema en paquetes. Esto se realiza con el objetivo de proveer al sistema de una mayor reutilización y facilitar su mantenimiento.

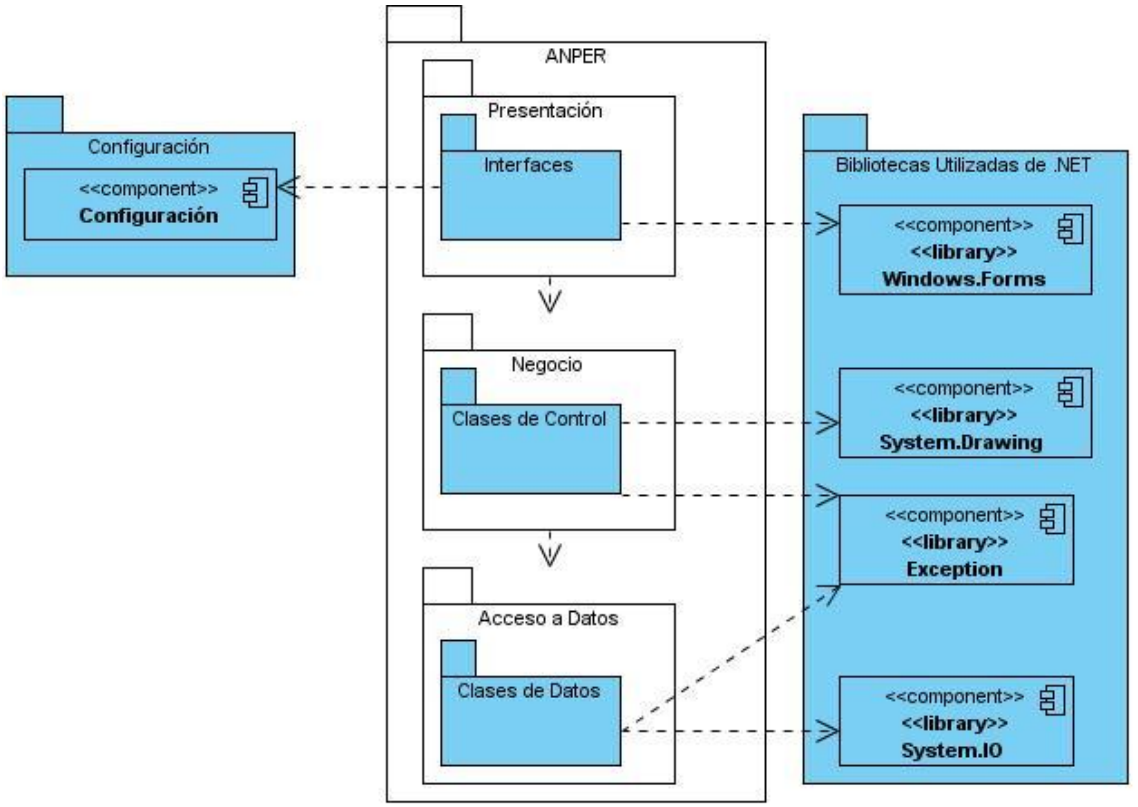


Figura 8 Vista Lógica.

A continuación se da una pequeña explicación de cada paquete del sistema:

- **Presentación:** está formado por los formularios del sistema, estos se crean con la librería **Windows.Forms** y son los que el usuario usará para comunicarse con el sistema.
- **Negocio:** contiene las clases controladoras que son las encargadas de recibir las peticiones de la capa de presentación y enviarlas después de su proceso, estas utilizan la librería **System.Drawing** y **Exception**.
- **Acceso a Datos:** está formado por las clases de acceso a datos que son las encargadas de la integridad y la organización de los ficheros .LAS, estas utilizan la librería **System.IO** y **Exception**.
- **Configuración:** es el encargado de configurar los formularios del sistema.

3.3.3 Vista de Desarrollo

La vista de desarrollo o de implementación se centra en la organización real de los módulos de software y describe la descomposición del software en subsistemas de implementación. En este diagrama se tiene en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software y la reutilización.

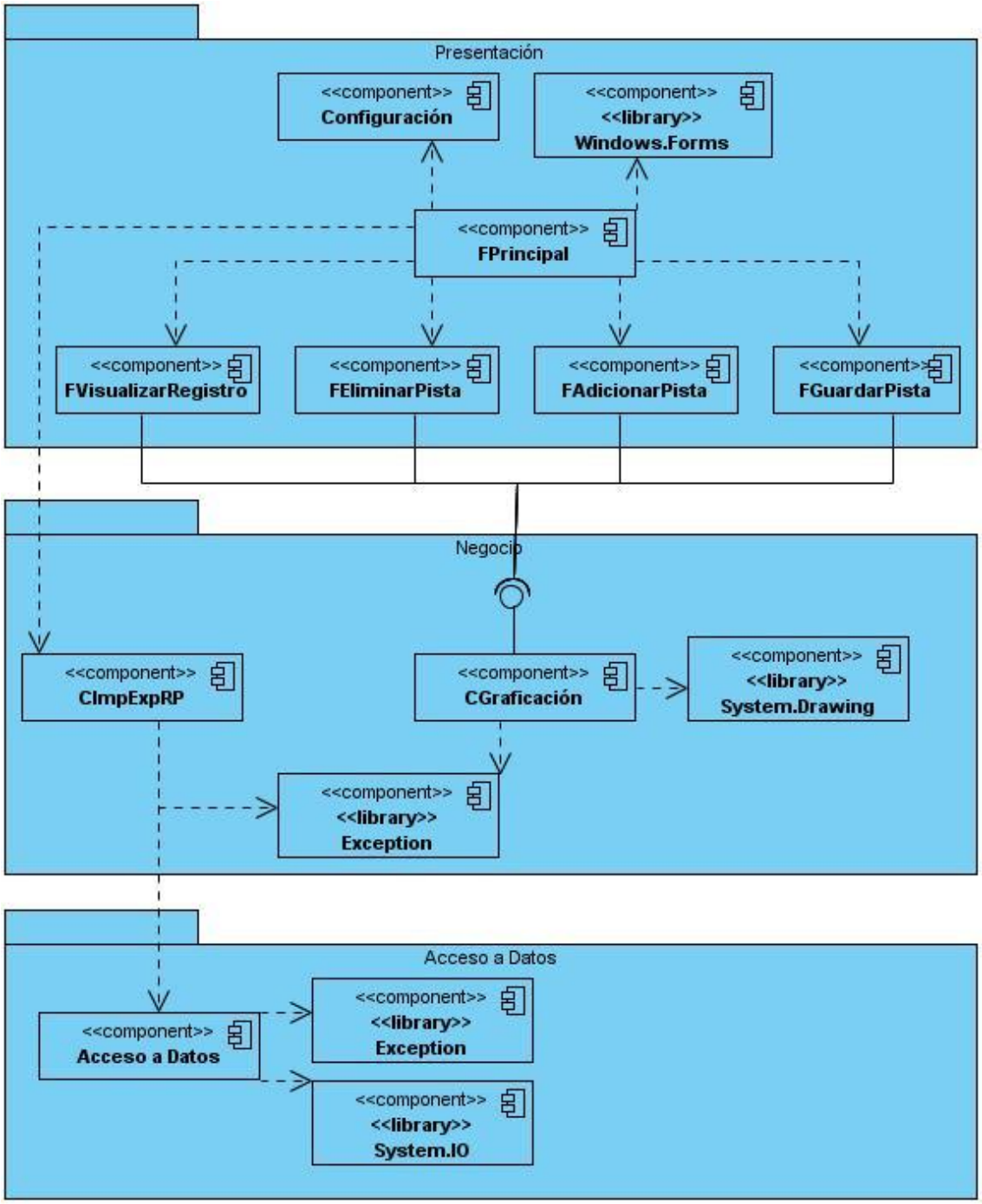


Figura 9 Vista de Implementación.

3.3.4 Vista de Procesos

Esta vista suministra una base para la comprensión de la organización de los procesos del sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos concurrentes o hilos.

En este caso la solución propuesta no presenta procesos concurrentes por tanto no se hace representación de esta vista.

3.3.5 Vista Física

La vista física o de despliegue propone la distribución física de los componentes hardware y software en el sistema. Es un conjunto de nodos unidos por conexiones de comunicación. Se satisfacen además parte de los requisitos no funcionales de hardware y software.

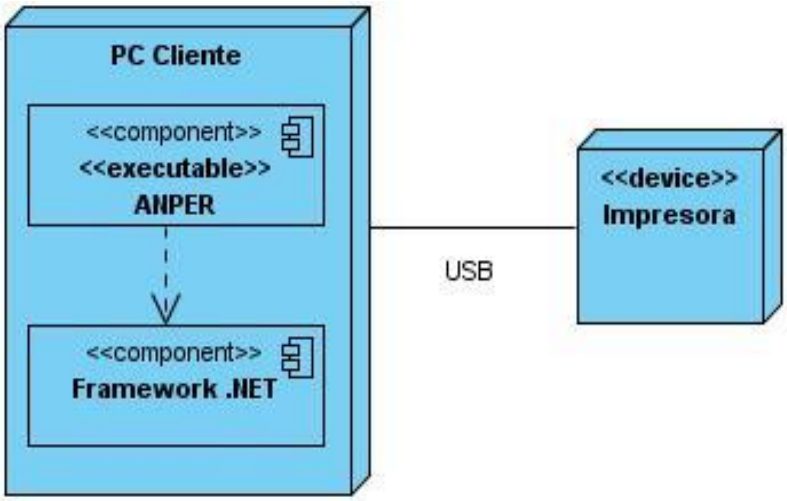


Figura 10 Vista de Despliegue.

Nodo PC Cliente: Contiene un ejecutable de la aplicación y el framework 3.5 Net instalado para poder correr dicha aplicación.

Dispositivo Impresora: satisface el requerimiento de impresión de la visualización del registro de pozo.

3.4 Validación de la solución propuesta

Después de haber realizado un estudio detallado de la tesis: Factibilidad del uso de métodos de evaluación de arquitectura en los proyectos productivos de la UCI, de la Ing. Dayana Calvo San Martin se decidió que la arquitectura propuesta para el Sistema de Análisis de Registros de Pozos Petroleros (ANPER) será evaluada utilizando el Método de Análisis de Acuerdos de Arquitectura (ATAM), que está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de los atributos de calidad y el método de evaluación SAAM. El método de evaluación ATAM comprende nueve pasos agrupados en cuatro fases (Ver Anexo 1).

Este método no se aplicará en su totalidad ya que el arquitecto entiende los procesos descritos en la misma porque ha sido parte del equipo de desarrollo y solo realizará

una pequeña evaluación de la arquitectura basándose en los atributos de calidad más importantes del sistema a implementar.

3.4.1 Atributos de calidad

Los siguientes atributos fueron identificados como de alta prioridad:

Seguridad: el sistema debe evitar intromisiones maliciosas en el mismo para la seguridad de los datos y debe permitir la verificación de acciones irreversibles.

Portabilidad: la aplicación debe ser implementada de tal forma que sea ejecutada en cualquier sistema operativo.

Disponibilidad: la aplicación debe estar disponible en todo momento que el trabajador lo desee.

Se identificaron los siguientes atributos de calidad con prioridad media:

Rendimiento: el sistema debe contar con un tiempo de respuesta de menos de 6 segundos en dependencia de la complejidad del servicio.

Usabilidad: la aplicación será usable por cualquier tipo de usuario con experiencia en los procesos de análisis petrofísico, además el sistema debe contar con cualidades que lo hagan atractivo y cómodo para el usuario.

3.4.2 Generación del árbol de utilidad

La tabla 2 muestran los atributos de calidad identificados en la validación de dicha arquitectura. Los escenarios están formados por pares, el primero de cada par indica la importancia que tiene para el sistema y el segundo la dificultad de su realización estimada por el arquitecto.

Atributos de Calidad	Escenarios
Seguridad	Todo aquel que cuente con un usuario y una contraseña de la máquina puede acceder a la aplicación.
	Si el usuario desea eliminar una pista la aplicación le brinda la posibilidad de cancelar dicha operación o de eliminarla.

Portabilidad	Sustituir el sistema operativo.
	Trasladar la aplicación a otra máquina.
Disponibilidad	La aplicación estará disponible los 365 días del año.
Rendimiento	El usuario desea importar un fichero .LAS el tiempo de respuesta del sistema para cargar toda la información es de 3 segundos.
Usabilidad	El usuario en un caso particular de la aplicación pide ayuda, el sistema debe ser capaz de brindar dicha información.
	Un usuario con más de dos años de experiencia en los procesos de análisis petrofísico debe dominar la aplicación en menos de una semana.

Tabla 2 Árbol de Utilidad.

Riesgos:

1. Constituye un riesgo que la aplicación no tenga ninguna seguridad para proteger la información que en ella se maneja.
2. Es importante destacar que con el avance y el auge que está teniendo el software libre en la actualidad constituye un riesgo y una desventaja que la aplicación no sea multiplataforma.
3. Con la arquitectura de software no se garantiza que la aplicación estará disponible las veinticuatro horas de los siete días de la semana y no se garantiza que no ocurra un fallo en el sistema.

Las otras operaciones que se realizan en los diferentes escenarios no constituyen riesgos para la aplicación ya que trasladar la aplicación a otra máquina es sencillo, el sistema tiene un tiempo de respuesta para la importación de los ficheros .LAS de tres segundos, la aplicación cuenta con un manual de usuario y una opción de Ayuda que estará siempre disponible para que el usuario la consulte, esta tiene explicación en textos basados en ejemplos de imágines.

Se llega a la conclusión de que la evaluación de la arquitectura de software es importante ya que permite encontrar puntos sensibles en el desarrollo de la aplicación, pero es válido destacar que los atributos de calidad que significan riesgos dentro de la aplicación no son de gran importancia ya que son atributos que dados los escenarios

definidos arrojan resultados negativos pero que no afectan la finalidad con la que fue concebida la aplicación desde el planteamiento del objetivo general de la investigación. Por lo que se plantea que la arquitectura propuesta cumple con los requerimientos no funcionales del sistema.

3.5 Conclusiones

En este capítulo se realizó la descripción de la arquitectura propuesta para el Sistema de Análisis de Registros de Pozos Petroleros (ANPER), haciendo referencia a los requerimientos no funcionales del sistema y a las vistas arquitectónicas definidas por RUP, se concluye que las 4+1 vistas son de gran importancia para el equipo de desarrollo ya que le permite un mejor entendimiento de la organización del sistema. Además estas vistas junto con los requerimientos no funcionales sirven para la documentación del artefacto Arquitectura de Software. También se validó la arquitectura y se llego a la conclusión que esta cumple con los requerimientos no funcionales.

CONCLUSIONES

Después de un estudio exhaustivo del mercado se concluye que la Petrofísica tiene gran importancia, ya que permite la obtención de propiedades físicas de las rocas tales como la porosidad, la permeabilidad y la saturación del agua, esto conlleva a que se realice de forma más eficiente la exploración y posterior explotación de los yacimientos de hidrocarburos.

Además se hizo posible arribar a las siguientes conclusiones:

- El valor fundamental del sistema a desarrollar está en el ahorro económico que constituye para el país y en un producto que se adapte a las necesidades verdaderas del CEINPET.

- Con el estudio detallado de los principales aspectos arquitectónicos se logró seleccionar el estilo arquitectónico adecuado para la solución, aprovechando la ventaja de desarrollos en paralelos en cada capa.

- Con el estudio detallado de los Patrones de Asignación de Responsabilidades y de los Patrones GOF, se le añadió al sistema claridad y sencillez. Los patrones favorecen la reutilización de clases ya existentes y la programación de clases reutilizables.

- A través de la descripción de la arquitectura mediante las vistas arquitectónicas definidas por RUP se garantizó que el grupo de desarrollo tuviera una visión de los elementos arquitectónicamente significativos y de todos los modelos del sistema.

- A través de la validación de la arquitectura de detectaron riesgos que pueden ser mitigados en la segunda versión del producto.

Finalmente se puede afirmar que se cumplieron los objetivos trazados, logrando definir una arquitectura que cumple con todos los requerimientos no funcionales del sistema.

RECOMENDACIONES

Al término del presente trabajo de diploma se recomienda:

- El refinamiento constante de la arquitectura propuesta en el ciclo de desarrollo.
- El estudio detallado de la Plataforma Mono para una futura migración.
- Valorar la posibilidad del uso de una Base de Datos para el trabajo con ficheros en otra versión del producto.
- Valorar la posibilidad de la seguridad del sistema, diseñando un gestionar usuario para la segunda versión del producto.

REFERENCIAS BIBLIOGRÁFICAS

1. **Famiglietti, Nathaly.** Petróleo:Estudio de la petrofísica. [En línea] [Citado el: 4 de diciembre de 2009.] http://www.seed.slb.com/v2/FAQView.cfm?ID=914&Language=ES.

2. Schlumberger Oilfield Glossary. [En línea] [Citado el: 4 de diciembre de 2009.] http://www.glossary.oilfield.slb.com/Display.cfm?Term=fluid%20flow.

3. **Da Silva, Angel.** Curso de Propiedades de la Roca Yacimiento. [En línea] [Citado el: 5 de diciembre de 2009.] http://www.lacomunidadpetrolera.com/cursos/propiedades-de-la-roca-yacimiento/.

4. **Brown, Trevor y Burko, Thomas.** Entrega de datos a tiempo. [En línea] [Citado el: 5 de diciembre de 2009.] http://www.slb.com/~/media/Files/resources/oilfield_review/spanish00/spr00/P34_55.ashx.

5. Productos de Neuralog. [En línea] [Citado el: 6 de diciembre de 2009.] http://www.adams-consulting.net/Neuralog_productos.htm.

6. Hydrocarbon Data Systems - HDS 2000 Log Anslysis Software. . [En línea] [Citado el: 6 de diciembre de 2009.] http://www.hds-log.com/.

7. Petrolog - Advanced Log Analysis Software. [En línea] [Citado el: 8 de diciembre de 2009.] http://www.petrolog.net/product-detail.asp?iSoftwareID=8.

8. **Reynoso, Carlos y Kiccillof, Nicolás.** Estilos y Patrones en la Estrategia de la Arquitectura de Microsoft. [En línea] [Citado el: 14 de enero de 2010.] http://www.willydev.net/descargas/prev/Estiloypatron.pdf.

9. IEEE. ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems. [En línea] [Citado el: 14 de enero de 2010.] http://www.sei.cmu.edu/architecture/start/definitions.cfm#Modern.

10. **Reynoso, Carlos.** Arquitectura de Software. [En línea] [Citado el: 16 de enero de 2010.] http://www.utim.edu.mx/~raycv/materias/ing_software/unidad_III/01%20Arquitectura%20de%20Software.pdf.

11. **Bass, Len, Clements, Paul y Kasman, Rick.** *Software Architecture in Practice.Second Edition.* 2004. 0321154959.

12. Ingenieria del Software IUSH: Diseño Arquitectonico. [En línea] [Citado el: 25 de abril de 2010.] http://ingsoftiush.blogspot.com/2010/03/diseno-arquitectonico.html.

13. **Shaw, Mary y Clements, Paul.** Una guía del campo a Boxology: Clasificación preliminar de los Estilos Arquitectónicos para Sistemas de Software. [En línea] [Citado el: 25 de enero de 2010.] http://translate.google.com/translate?hl=es&sl=en&u=http://citeseerx.ist.psu.edu/viewd

oc/download%3Bjsessionid%3D2E54D3EE89CA2BC2B843213414992CCC%3Fdoi%3D10.1.1.53.9745%26rep%3Drep1%26type%3Dpdf&prev=/search%3Fq%3D9.%2509Mary%2BShaw%2By%2BPau.

14. Glosario.Las reservas de hidrocarburos de México.[En línea][Citado el:14 de junio de 2010] http://www.sener.gob.mx/webSener/res/545/GLOSARIO.pdf

15. **Burbeck, Steve.** Applications Programming in Smalltalk-80(TM):How to use Model-View-Controller (MVC). [En línea] 1992. [Citado el: 22 de enero de 2010.] http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

16. **Prof. Ochoa, Sergio.** Introducción a los Patrones (Diseño y Arquitectura). [En línea] 2005. [Citado el: 29 de enero de 2010.] https://www.u-cursos.cl/ingenieria/2005/2/CC51A/1/material_docente/objeto/76454..

17. **Ing. Carrascoso Puebla, Yoan Arlet y Ing. Chaviano Gómez, Enrique.** Propuesta de arquitectura orientada a servicios para el módulo de inventario del ERP cubano. [En línea] 15 de mayo de 2009. [Citado el: 30 de enero de 2010.] http://www.gestiopolis.com/administracion-estrategia/erp-arquitectura-orientada-a-servicios.htm.

18. **Lago, Ramiro.** Patrones de diseño software. [En línea] abril de 2007. [Citado el: 16 de marzo de 2010.] http://proactiva-calidad.com/java/patrones/index.html.

19. **Alexander, Christopher.** *Timeless Way Of Building.* 1979. 0195024028 .

20. **Larman, Craig.** *Uml y patrones: Introducción al análisis y programación orientada a objetos.*

21. **Moreno Navarro, Juan José.** Arquitecturas Software.(Curso de Software basado en Componentes, junto a Lars-Ake Fredlund). [En línea] [Citado el: 20 de marzo de 2010.] http://babel.ls.fi.upm.es/~fred/sbc/arquitecturas_sw.pdf.

22. **Visconti, Marcello y y Astudillo, Hernán.** Patrones. Fundamentos de Ingeniería de Software. [En línea] [Citado el: 19 de marzo de 2010.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.

23. **Machorro Reyes, Ricardo Armando.** Los Patrones como un Medio del Diseño Orientado a Objetos. [En línea] [Citado el: 22 de marzo de 2010.] http://www.revistaupiicsa.20m.com/Emilia/RevMayAgo04/Machorro1.pdf.

24. **Welicki, León.** El Patrón Singleton. [En línea] [Citado el: 22 de marzo de 2010.] http://msdn.microsoft.com/es-es/library/bb972272.aspx.

25. **E. Cuesta, Carlos.** Patrones de Diseño. Ingeniería del Software I Universidad Rey Juan Carlos. [En línea] [Citado el: 23 de marzo de 2010.] http://kybele.escet.urjc.es/documentos/ISI/Patrones%20de%20Dise%C3%B1o.pdf.

26. **Cataldi, Zulma y Lage, Fernando J.** Sistemas Tutores Inteligentes: Procedimientos, métodos, técnicas y herramientas para su creación. [En línea] [Citado el: 25 de marzo de 2010.]

http://www.virtualeduca.info/ponencias2009/558/Entorno%20STI%20Frame%2003-09-09%20eje%205.doc.

27. Información general y conceptual sobre .NET Framework. [En línea] noviembre de 2007. [Citado el: 27 de marzo de 2010.] http://msdn.microsoft.com/es-es/library/zw4w595w.aspx.

28. Características del lenguaje Java. [En línea] [Citado el: 1 de febrero de 2010.] http://www.iec.csic.es/criptonomicon/java/quesjava.html.

29. Características de Java . [En línea] [Citado el: 2 de febrero de 2010.] http://www.manual-java.com/manualjava/caracteristicas-java.html.

30. Tutorial de C#.Características de C#. [En línea] [Citado el: 3 de febrero de febrero.] http://www.clikear.com/manuales/csharp/c10.aspx.

31. **G.Figueroa, Roberth, J.Solís, Camilo y A. Cabrera, Armando.** METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES. [En línea] [Citado el: 2 de abril de 2010.] http://adonisnet.files.wordpress.com/2008/06/articulo-metodologia-de-sw-formato.doc..

32. **Booch, Grady, Rumbaugh, James y Jacobson, Ivar.** El Lenguaje Unificado de Modelado. [En línea] [Citado el: 3 de abril de 2010.] http://elvex.ugr.es/decsai/java/pdf/3E-UML.pdf.

33. UML (Unified Modeling Language). [En línea] [Citado el: 29 de marzo de 2010.] http://www.scribd.com/doc/2080534/UML.

34. **Castro Jiménez, Eliseo.** UML - Lenguaje de Modelamiento Unificado. [En línea] [Citado el: 29 de marzo de 2010.] http://www.slideshare.net/ecastrojimenez/uml-lenguaje-de-modelamiento-unificado-presentation.

35. **Schmuller, Joseph.** *Aprendiendo UML en 24 Horas.* 200. 968-444-463-X.

36. Visual Paradigm for UML (ME) - (Paradigma Visual para UML (ME)) (Visual Paradigm for UML (ME)) 6.0 . [En línea] [Citado el: 5 de abril de 2010.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/.

37. Eva. Ingeniería de Software 1. Conferencia #6. Disciplina de Requisitos. Técnica de Caso de uso.. Curso 2009-2010. [En línea] [Citado el: 10 de abril de 2010.] http://eva.uci.cu/file.php/102/Curso_2009-2010/Semana_7/Conferencia_6/Materiales_Basicos/Conferencia_6.doc.

38. **Chávez, Arroyo Raudel.** TD_2780_09. [En línea] 2009. [Citado el: 6 de mayo de 2010.] http://bibliodoc.uci.cu/TD/TD_2780_09.pdf.

39**. Quartz,Grain.**Porosidad.Mediciones.[En línea][Citado el: 14 de junio de 2010] http://www.scribd.com/doc/22946001/Porosidad

BIBLIOGRAFÍA CONSULTADA

1. **David, Garlan y Mary, Shaw.** *An Introduction to Software Architecture.* 1994.

2. **Kruchten, Philippe.** Architectural Blueprints—The “4+1” View Model of Software Architecture. [En línea] noviembre de 1995. [Citado el: 25 de abril de 2010.] http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf.

3. **Fowler, Martin.** *Patterns Of Enterprise Application Architecture.* 2002. 9780321127426 .

4. **Clements, Paul, Kazman, Rick y Klein, Mark.** *Evaluating Software Architectures: Methods and Case Studies.* 020170482X.

5. **Kazman, Rick, Klein, Mark y Clements, Paul.** ATAM: Method for Architecture Evaluation. [En línea] agosto de 2000. [Citado el: 15 de mayo de 2010.] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.6014&rep=rep1&type=pdf.

6. **Blanco, Cuaresma, Sergio.** Mono: La plataforma .NET libre. [En línea] 11 de noviembre de 2003. [Citado el: 6 de abril de 2010.] http://www.willydev.net/descargas/prev/TodoMono.pdf.

7. **López, Natarén, César.** Mono: .NET libre y multi-plataforma. [En línea] 6 de octubre de 2005. [Citado el: 7 de abril de 2010.] http://primates.ximian.com/~cnataren/mono-oct6.pdf.

8. **Buschmann, Frank, Meunier, Regine, y otros.** *Design Pattern-Oriented Software Architecture. .* 0471958697 .

9. **López, Quiñones,Ricardo Alfonso.** Arquitectos de Software en Práctica. [En línea] [Citado el: 8 de mayo de 2010.] http://sophia.javeriana.edu.co/~cbustaca/Arquitectura%20Software/Clases/Ensayos_2008/Ricardo_Alfonso_Lopez_07.pdf.

10. **Cristía, Maximiliano.** Introducción a la Arquitectura de Software. [En línea] agosto de 2008. [Citado el: 20 de abril de 2010.] http://www.fceia.unr.edu.ar/ingsoft/intro-arq.pdf.

11. Revista Ambientum. [En línea] 2004. [Citado el: 27 de marzo de 2010.] http://www.ambientum.com/revista/2004_07/PETROLEO_imprimir.htm.

12. Arquitectura 3 Capas. [En línea] 6 de febrero de 2009. [Citado el: 30 de marzo de 2010.] http://kernelerror.net/programacion/php/arquitectura-3-capas/.

13. **López, Requena, Martín Luis.** Microsoft Solutions Framework. [En línea] agosto de 2005. [Citado el: 1 de abril de 2010.] http://www.malagadnug.org/ficheros/MSFMartinLuisReq.pdf.

ANEXOS

Anexo 1: Fases del método ATAM.

Fase 1:Presentación	
1. Presentación del ATAM.	El líder de evaluación describe el método a los participantes, trata de establecer las expectativas y responde las preguntas propuestas.
2. Presentación de las metas del negocio.	Se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico.
3. Presentación de la arquitectura.	El arquitecto describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio.
Fase 2: Investigación y análisis.	
4. Identificación de los enfoques arquitectónicos.	Estos elementos son detectados, pero no analizados.
5. Generación del árbol de utilidad.	Se refinan los atributos de calidad que engloban la "utilidad" del sistema (desempeño, disponibilidad, seguridad, modificabilidad, usabilidad, etc.), especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.
6. Análisis de los enfoques arquitectónicos	Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. En este paso se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance.
Fase 3: Pruebas	
7. Lluvia de ideas y establecimiento de prioridad de escenarios.	Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.
8. Análisis de los enfoques arquitectónicos.	Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.

Fase 4: Reporte	
9. Presentación de los resultados.	Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

Tabla 3 Pasos del método de evaluación ATAM.

Anexo 2: Encuesta Realizada al Jefe del Proyecto.

1. ¿Qué problemas presenta actualmente el CEINPET, que motiven a la realización de un software para el análisis petrofísico?
2. ¿Cuáles son los objetivos que se persiguen con la realización de un software para el análisis petrofísico? Enumérelos.
3. ¿Existe algún software cubano que se dedique al análisis petrofísico? ¿Cual (és)?

   Sí___ No___
4. Mencione el (los) software que se utiliza (n) en el CEINPET. Diga Ventajas y desventajas.
5. ¿Dichos software son propietarios?

   Sí___ No___
6. ¿De qué forma son adquiridos dichos software?
7. ¿Es muy costoso pagar la licencia del (de los) software utilizado (s) en el CEINPET?

   Sí___ No___
8. ¿Existe (n) otro (s) software que se dedique (n) al análisis petrofísico? Menciónelos.
9. ¿El (los) software usado (s) por el CEINPET le falta (n) alguna (s) funcionalidad (es)? Enumérelas.

   Sí___ No___
10. ¿Qué funcionalidad (es) se desea (n) para la primera versión del producto, para cuándo quieren el producto?

GLOSARIO DE TÉRMINOS

Schlumberger: Empresa que brinda servicios a la industria petrolera, es una de las empresas líder en este sector.

Hidrocarbure Data Systems (HDS): Software utilizado por el CEINPET para el análisis petrofísico.

Interactive Petrophysics (IP): Software utilizado por el CEINPET para el análisis petrofísico.

NeuraPrep: Software elaborado por la empresa **Neuralog** que se dedica al análisis petrofísico.

Petrolog: Software que se dedica al análisis petrofísico.

OO: Orientado a Objeto.

GRASP: Patrones de Principios Generales para Asignar Responsabilidades.

POO: Programación Orientada a Objeto.

Hipermedia: Término con que se designa al conjunto de métodos o procedimientos para escribir, diseñar o componer contenidos que tengan texto, video, audio, mapas y que además tenga la posibilidad de interactuar con los usuarios.