Universidad de las Ciencias Informáticas

Facultad 6

Título: Propuesta de Arquitectura para el Sistema de Software Minero GeolMin.

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

Autora: Thais Amigot Mora.

Tutor: Ing. Armando Ortíz Cabrera.

Ciudad de la Habana, Junio 27 del 2011
"Año 53 de la Revolución"

"Saber no es suficiente, debemos aplicar. Desear no es suficiente, debemos hacer."

Johann Wolfgang von Goethe.

Dedicatoria

A mi mamá, papá, hermanita, a Yusley, a mi familia y amigos que me han acompañado en esta etapa de mi vida y que de una forma u otra la enriquecieron personal y profesionalmente.

Muchas Gracias.

Agradecimientos

A mi madre y mi padre por los atinados consejos, por la preocupación, por siempre apoyarme en todas mis decisiones, por su cariño, por su comprensión, por la espera, por... A ellos por todo.

A mis hermanos por el apoyo que me brindaron en estos años.

A Yusley, mi novio, por tantas cosas… pero sobre todo por quererme y alentarme con su infinito, interminable… inalcanzable… optimismo.

A mi abuelita, tíos y resto de la familia. A mis amigos.

A todos Muchas Gracias.

Declaración de Autoría

Declaro que soy la única autora de este trabajo y autorizo a la Facultad 6 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de ____del año 2011.

_____	_____
Thais Amigot Mora	Armando Ortiz Cabrera
Autor	Tutor

Datos de Contacto

Tutor: Ing. Armando Ortiz Cabrera.

Especialidad de graduación: Ingeniero en Ciencias Informáticas.

Categoría docente: Instructor.

Años de experiencia en el tema: 4

Correo electrónico: aortizc@uci.cu

Universidad de la Ciencias Informáticas, La Habana, Cuba.

Resumen

En la Universidad de las Ciencias Informáticas (UCI), específicamente en la Facultad 6, se encuentra el Centro de Desarrollo Geoinformática y Señales Digitales (GEySED). Dentro de este centro, el proyecto Minería tiene como principal objetivo, el desarrollo de sistemas de software que permitan el control de datos geológicos y principales procesos en la minería. Actualmente este proyecto se encuentra inmerso en el desarrollo del software minero GeolMin, que permitirá la modelación y visualización de objetos geológicos.

Sin lugar a dudas, la robustez y flexibilidad del software, son elementos notables que ponen en evidencia un grupo de cambios y mejoras con el transcurso del tiempo. Estos elementos de calidad, se garantizan mediante la confección de una arquitectura de software. El presente trabajo investigativo tiene como objetivo fundamental conformar una arquitectura para el sistema de software minero GeolMin, la misma deberá estar basada en el principio de orientar el desarrollo de sistemas al software libre.

Para alcanzar los objetivos propuestos se realizó el análisis de los sistemas mineros existentes, así como los principales conceptos, estilos y patrones arquitectónicos. Se hace además un estudio de las principales tendencias, metodologías y herramientas actuales, con el fin de obtener una arquitectura que responda a las buenas prácticas de programación. Se hace una evaluación de la arquitectura propuesta, buscando garantizar una solución robusta que garantice la calidad del software.

Palabras Claves: Arquitectura de software, Estilos Arquitectónicos, Patrones de diseño, Proyecto GeolMin, Software Libre.

Índice

Índice de tablas

Índice de figuras

INTRODUCCIÓN

Actualmente la informatización en cada esfera de la sociedad se ha convertido en una necesidad; destacándose el campo investigativo, formativo y productivo. Gracias al constante desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC), los procesos son automatizados, propiciando una superación y mejoras en todo el ámbito socioeconómico. Esto dio paso a la utilización de un conjunto de sistemas especialmente diseñados para la planificación de la explotación minera.

En Cuba, se ha tratado de incluir la tecnología informática para reemplazar procedimientos manuales, aumentar la calidad y disminuir los costos en diversas áreas de la sociedad. Tratándose muchas veces con productos de software propietarios. Tal es el caso de la actividad geólogo minera, para la cual la adquisición de licencias impuso una dependencia tecnológica, con la utilización de software privativos como GEMCOM, SURPAC y DATAMINE.

La aplicación de la informática en la geología minera cubana tomó fuerzas a finales de los años 90, desarrollando herramientas como: TIERRA, SIM, CORTE y MICRONIQ. Las principales entidades que han tenido el compromiso del desarrollo de las mismas son: la Universidad de Pinar del Rio, el Instituto Superior Metalúrgico de Moa (ISMM), las Industrias del Níquel, las Empresas Geomineras, el Instituto Superior Politécnico José Antonio Echeverría (ISPJAE) y en los últimos dos años la Universidad de las Ciencias Informáticas (UCI).

A pesar de los esfuerzos por explotar estas tecnologías, siguen presente algunas deficiencias que surgen del mal aprovechamiento de las mismas y frenan el desarrollo progresivo en esta rama. Si se analiza cómo se ha llevado a cabo este proceso en el país, sobresale la dependencia tecnológica de las empresas hacia las grandes compañías desarrolladoras y comercializadoras de los sistemas de software mineros.

Es por ello que el centro de Geoinformática y Señales Digitales (GEySED), de la Facultad 6, en la Universidad de las Ciencias Informáticas, cuenta con el proyecto Minería. El cual surge con la idea de desarrollar sistemas de software para facilitar el trabajo con los yacimientos minerales. El sistema minero GeolMin que actualmente se encuentra en desarrollo, permitirá la modelación y visualización de objetos

geológicos. Mediante la inserción de datos iniciales, se podrán obtener proyecciones de pozos mineros para el trabajo con los minerales.

El desarrollo de estas aplicaciones muchas veces carecen de documentación o no tienen ninguna, lo que hace que el código generado apenas se pueda reutilizar. Actualmente no existen vistas generales que representen la estructuración de los componentes del sistema, lo que dificulta entender como estos fueron desarrollados. Además, no existe una línea base que guíe el desarrollo de los distintos procesos del sistema identificados, lo que provoca constantes cambios y que los trabajadores no posean una visión de lo que se quiere realizar. Estas deficiencias conllevan a que el proyecto no se termine en el tiempo planificado y que el costo del mismo aumente.

La arquitectura de software también se encarga de asegurar que la aplicación contenga atributos esenciales como: eficiencia, usabilidad, flexibilidad, facilidad de mantenimiento, seguridad e integridad. Por la importancia que tiene definir una arquitectura tanto para el producto como para el proceso de desarrollo de software, surge la necesidad de conformar un diseño de alto nivel para el software minero GeolMin, basada en el principio de orientar el desarrollo del sistema al software libre, que cumpla con los atributos de calidad asociados, los servicios y funcionalidades que espera el usuario.

Teniendo en cuenta la situación anteriormente expuesta se identifica como **problema científico** de la investigación: ¿Cómo definir la organización estructural de los componentes del sistema de software minero GeolMin?

Se determinó como **objeto de estudio**: La arquitectura de software y como **campo de acción**: La arquitectura de software para el software minero GeolMin.

Como **idea a defender** se plantea que: Si se define un marco arquitectónico con la selección de los patrones adecuados, se logrará el diseño de una arquitectura robusta y flexible que responda a las necesidades del sistema a implementar.

Se define como **objetivo general** de la investigación: Diseñar la arquitectura del sistema de software minero GeolMin.

Para cumplir con el objetivo general propuesto y darle solución a la situación problemática planteada se trazan un grupo de **tareas investigativas**:

- Caracterizar las diferentes propuestas arquitectónicas para software minero, tomando posición al respecto.
- Seleccionar propuestas de estilos arquitectónicos a partir del estudio del negocio.
- Seleccionar patrones arquitectónicos a utilizar.
- Diseñar la propuesta arquitectónica que cumpla con los requerimientos del sistema.
- Diseñar las vistas de la arquitectura.
- Evaluar la arquitectura definida.

Para lograr un mejor entendimiento de la situación actual y cumplir íntegramente las tareas de la investigación científica se pondrán en práctica los siguientes **métodos científicos**:

Métodos Teóricos:

Analítico-Sintético: Permitió realizar un análisis de los documentos y teorías existentes con el fin de obtener los elementos más importantes que se relacionan con el objeto de estudio, permitiendo seleccionar adecuadamente los patrones de arquitectura y el modelo arquitectónico a utilizar.

Histórico-Lógico: Permitió realizar un análisis cronológico de la trayectoria hasta la actualidad, del proceso de desarrollo de los sistemas utilizados para la minería en Cuba. Tomando en cuenta las tecnologías utilizadas para estas herramientas, se conocerá con exactitud los antecedentes y las tendencias actuales, los estilos y enfoques arquitectónicos.

Inducción-Deducción: Este método fue empleado para el estudio y análisis de las diferentes alternativas existentes, lo que posibilitó identificar las que serán utilizadas en el desarrollo de la propuesta arquitectónica.

Modelación: Posibilitó representar el entorno del problema, modelando el diseño de la propuesta de arquitectura a través de diferentes diagramas y vistas, que sirven de ayuda para una mejor comprensión del objeto de estudio.

Métodos Empíricos:

Entrevista: Se entrevistaron a especialistas en la geología minera, también a desarrolladores y arquitectos del proyecto Minería, lo que permitió recopilar información sobre aspectos específicos inherentes a la solución del problema de investigación.

El presente trabajo de diploma está estructurado por 3 capítulos, a continuación se describe brevemente cada uno de ellos.

Capítulo 1: Fundamentación Teórica.

En este capítulo se define el estado del arte de las empresas y software relacionados con la minería, así como el análisis teórico sobre las tecnologías actuales, principales tendencias y conceptos asociados a la construcción de sistemas mineros y la Arquitectura de Software.

Capítulo 2: Línea Base de la Arquitectura.

En este capítulo se realiza la propuesta de solución al problema planteado inicialmente, regida por el artefacto Documento Descripción de Arquitectura que propone la Universidad de las Ciencias Informáticas, se tienen en cuenta los planteamientos de la metodología *Rational Unified Process* (RUP) y se detalla la misma a través de las vistas de la arquitectura.

Capítulo 3: Validación de la Propuesta de Arquitectura.

En este capítulo se realiza una evaluación a la propuesta de arquitectura con el objetivo de obtener los riesgos y puntos sensibles, permitiendo suprimir las amenazas que se puedan presentar. Se analizan métodos existentes para la evaluación de arquitecturas de software, así como la aplicación de uno de ellos a la solución que se propone.

CAPÍTULO #1

FUNDAMENTACIÓN TEÓRICA

En este capítulo se detallan los principales elementos teóricos relacionados con la solución propuesta. Se profundiza en los estilos y patrones arquitectónicos, principales tecnologías, metodologías y herramientas de desarrollo que serán utilizadas en la solución del problema. Para un buen entendimiento de la investigación realizada se describen los conceptos asociados que puedan resultar desconocidos.

1.1. CONCEPTOS ASOCIADOS AL DOMINIO DEL PROBLEMA

Existen un conjunto de conceptos que se relacionan con la investigación y el problema planteado, estrechamente relacionados con la minería y la arquitectura de software, los cuales se describirán a continuación para una mejor comprensión del trabajo en general.

1.1.1 GEOLOGÍA

La geología es la ciencia que estudia la sucesión de los rasgos y caracteres geográficos que ha ido adoptando la superficie terrestre, desde el momento de la primera consolidación de la litósfera hasta el presente. Subdivisiones: cosmología, paleontología, geología estructural, petrología, geomorfología, fisiografía, mineralogía. (Leyet, 2007)

1.1.2 MINERÍA

La minería es la obtención selectiva de los minerales y otros materiales de la corteza terrestre. También se denomina así a la actividad económica primaria relacionada con la extracción de elementos de los cuales se puede obtener un beneficio económico. Dependiendo del tipo de material a extraer la minería se divide en metálica y no metálica. (Florencia, 2009)

A través de la minería se puede, por un lado, obtener de manera selectiva minerales y algunos otros materiales desde la mismísima corteza terrestre y por otro lado, es también una actividad económica que despliegan muchos países del mundo a través de la cual extraen valiosos elementos de la tierra, como pueden ser el aluminio, cobre, hierro, plomo, oro, entre otros, con el objetivo de obtener un ventaja económica a partir de su comercialización.

El objetivo principal de este trabajo es de conformar una arquitectura de software candidata para el sistema minero GeolMin, que cumpla con las exigencias propuestas, obteniendo así una arquitectura adaptable, operativa y extensible. Por lo que a continuación se describen elementos esenciales de la arquitectura y se enfatiza principalmente en lo que se propone utilizar para el sistema.

1.1.3 ARQUITECTURA DE SOFTWARE

La arquitectura de software remonta sus antecedentes a la década de 1960. Edsger Dijkstra, David Parnas y Fred Brooks fueron unas de las primeras figuras que hablan sobre la arquitectura de software, desde entonces ha sido vista de disímiles formas, todas muy diferentes a la disciplina que se conoce actualmente. Existen muchas definiciones de arquitectura de software, pero con ninguna de ellas se ha estado totalmente de acuerdo, aunque es común entre los autores el concepto de arquitectura contemplado desde un alto nivel de abstracción.

La definición oficial de la arquitectura de software es la planteada por la IEEE Std2 1471-2000 y también adoptada por Microsoft, la cual expresa que: "*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.*" (ISO, 2007)

El libro de Ingeniería de software Un enfoque práctico, define que: *"La arquitectura de software de un sistema de programa o computación, es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente y las relaciones entre ellos".* (Pressman, 2001)

La metodología RUP plantea al respecto que la arquitectura de software, *"es la organización o la estructura de los componentes importantes del sistema que interactúan mediante interfaces, con componentes compuestos de interfaces y componentes cada vez más pequeños".* (IBM, 2003)

Como se puede apreciar en las definiciones anteriores, la mayoría de los autores coinciden en varios puntos, en cuanto a la estructura a grandes rasgos de un sistema, sus componentes, propiedades y las relaciones entre ellos. Al profundizar en este sentido, se puede inferir que de forma general una arquitectura de software establece los fundamentos para lograr una línea común de trabajo, que permita alcanzar los objetivos del sistema propuestos, siendo este el diseño de más alto nivel de la estructura de un sistema, que se manifiesta a los inicios del proceso de su creación.

La arquitectura de software brinda una vista general del sistema, compuesta por componentes que concretan como se va a realizar la aplicación, definiendo los objetivos prefijados como la adaptabilidad y flexibilidad. Para obtener una arquitectura de software correcta, en relación con lo que debe cumplir el sistema, se deben tener en cuenta principios de diseño como son los estilos y patrones arquitectónicos, patrones de diseño, patrones de asignación de responsabilidades, tecnología y herramientas, que se detallan a continuación.

1.1.4 ESTILOS ARQUITECTÓNICOS

El término estilo surge por la aparición de ciertas regularidades de configuración como respuesta a demandas comunes en los software. A estas respuestas se les da el nombre de estilos por analogía con el uso del término en la arquitectura de edificaciones.

Según Pressman, *un estilo describe una categoría de sistema que abarca un conjunto de componentes que realizan una función requerida por el sistema, un conjunto de conectores que posibilitan la comunicación, la coordinación y cooperación entre los componentes, las restricciones que definen como se integran los componentes para conformar el sistema, y los modelos semánticos que facilitan al diseñador el entendimiento de todas las partes del sistema*. (Pressman, 2001)

Dentro de los estilos arquitectónicos se pueden mencionar: (Reynoso, 2004)

- Estilos de Flujo de Datos.
  - ✓ Tubería y filtros.
- Estilos Centrados en Datos.
  - ✓ Arquitecturas de Pizarras o repositorios.
- Estilos de Llamada y Retorno.
  - ✓ Modelo-Vista-Controlador.
  - ✓ Arquitectura en capas.
  - ✓ Arquitectura basada en componentes.
  - ✓ Arquitectura Orientada a Objetos.

- Estilos de Código Móvil.
  - ✓ Arquitecturas 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.
  - ✓ Arquitecturas Basadas en Recursos.

Después de haber estudiado cada uno de los estilos anteriormente mencionados, se ha decidido utilizar para el sistema la combinación de tres de los tipos de estilos como son:

- **Arquitecturas en Capas:** El estilo en capas es como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. (Garlan, 1994)
- **Arquitecturas Orientadas a Objetos:** Los componentes del estilo se basan en principios OO o sea el encapsulamiento, la herencia y el polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación; las interfaces están separadas de las implementaciones; en cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. (Reynoso, 2004)
- **Arquitecturas Basadas en Componentes:** Los sistemas de software basados en componentes se basan en principios definidos por una ingeniería de software específica. Los componentes son las unidades de modelado, diseño e implementación, las interfaces están separadas de las implementaciones, y las interfaces y sus interacciones son el centro de incumbencias en el diseño arquitectónico. Los componentes soportan algún régimen de introspección, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución. (Reynoso, 2004)

La decisión de utilizar Llamada y Retorno, combinando tres de los estilos que esta encierra, viene dada por la facilidad de reutilización de software que proporciona. La ventaja de utilizar el estilo orientado a objetos, radica en que como los objetos están débilmente acoplados, la modificación de ellos puede efectuarse sin comprometer a los otros. Por otra parte, el estilo basado en componentes permite reemplazar una nueva versión del sistema por la existente sin impacto en el mismo o en sus componentes. También el estilo multicapas permite la distribución de las mismas en varios nodos físicos de computo y en varios procesos, lo que puede mejorar el desempeño, la coordinación y el compartir la información en un sistema de cliente-servidor.

La escalabilidad, reusabilidad, modificabilidad y usabilidad del sistema, son otros de los motivos que favorecen esta decisión. Durante el diseño y la programación, los estilos serán refinados mediante técnicas de patrones, refactorización, entre otros.

1.1.5 PATRONES

El propulsor de la idea de patrones fue Christopher Alexander en la década de 1970. Un patrón es una solución a un problema en un contexto, codifica conocimiento específico acumulado por la experiencia en un dominio. Un sistema bien estructurado está lleno de patrones. (Reynoso, 2004)

Un patrón es una guía estructurada que describe el problema y la solución de los principios y estilos que sirven de guía en la creación de sistemas de software. Según Craig Larman, un patrón es un par problema/solución con nombre, que se puede aplicar en nuevos contextos; con consejos acerca de cómo aplicarlo en ciertas situaciones o discusiones.

Estos suelen incluir una descripción detallada de su propósito y el escenario de uso en cuanto a su aplicabilidad. Proponiendo a su vez una solución concreta y señalando las consecuencias de utilizar este patrón. Hace mención de una lista de patrones relacionados así como ejemplos de implementación de los mismos.

Entre las categorías de los patrones se puede encontrar:

- **Patrones de arquitectura:** Están relacionados con la interacción de objetos dentro o entre niveles arquitectónicos. Se aplican durante la fase de diseño inicial para resolver problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, rendimiento, modularidad y

acoplamiento. Expresan un esquema organizativo estructural fundamental para sistemas de software.

- **Patrones de diseño:** Proporcionan un esquema para refinar los subsistemas o componentes de un sistema software y las relaciones entre ellos. Describe estructuras recurrentes para comunicar componentes que resuelven un problema de diseño en un contexto particular.

- **Patrones de análisis:** Se aplican durante el análisis para solucionar problemas relacionados con el modelo de dominio, completitud, integración y equilibrio de objetivos múltiples. Describen un conjunto de prácticas destinadas a elaborar modelos de los conceptos principales de la aplicación que se va a construir.

- **Patrones de proceso u organización:** Tratan todo lo concerniente con el desarrollo o procesos de administración de proyectos, técnicas y estructuras de organización.

- **Patrones de idioma:** Reglan la nomenclatura en la cual se escriben, se diseñan y desarrollan los sistemas. Los modismos o expresiones idiomáticas son el nivel más bajo de abstracción en un sistema de patrones. Son específicos a un lenguaje de programación determinado.

Según los aportes y resultados que se obtienen al aplicar el uso de patrones se puede asegurar que funcionan. Pues sus soluciones han sido utilizadas con anterioridad de manera repetida. Corresponden con problemas que no son específicos de un caso concreto, sino que se presentan una y otra vez en distintas aplicaciones, por lo que son reutilizables. Para el desarrollo del sistema se han decidido utilizar patrones de diseño y de análisis. Con la aplicación de estos tipos de patrones, se puede definir correctamente la estructura del sistema en todas sus fases de desarrollo. A continuación se describirán los seleccionados para su uso en el desarrollo del sistema.

1.1.6 PATRONES DE DISEÑO

Definición [Gamma et al. 95]: Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí. Adaptada para resolver un problema de diseño general en un contexto particular. Son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias). Los Patrones de Diseño nos hablan de cómo construir software, de cómo utilizar las clases y los objetos de forma conocida.

Los patrones de diseño se clasifican según su propósito como:

- **Patrones de Creación:** Tratan la creación de instancias. Abstraen la forma en la que se crean los objetos, permitiendo tratar las clases a crear de forma genérica, dejando para más tarde la decisión de qué clases crear o cómo crearlas. Estos además permiten que el sistema sea independiente de cómo se crean, componen o representan sus objetos.

- **Patrones Estructurales:** Cómo agrupar y organizar objetos. Tratan la relación entre clases, la combinación clases y la formación de estructuras de mayor complejidad. Además tratan de conseguir que los cambios en los requisitos de la aplicación no ocasionen cambios en las relaciones entre los objetos. Estudian cómo se relacionan los objetos en tiempo de ejecución. Sirven para diseñar las interconexiones entre los objetos.

- **Patrones de Comportamiento:** Cómo se relacionan en ejecución grupos de objetos. Tratan la interacción y cooperación entre clases. Estudian las relaciones de llamadas entre los diferentes objetos, normalmente ligados con la dimensión temporal. Estos hablan de como interaccionan entre si los objetos para conseguir ciertos resultados.

Dentro de los **Patrones de Creación** se encuentran:
Patrones de Creación de Clase:

- Factoría Abstracta
- Builder

Patrones de Creación de Objeto:

- Método Factoría
- Prototipo
- Singleton

De los cuales se utilizarán en el proyecto los siguientes:

- ***Abstract Factory*** **(Fábrica abstracta):** Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.

- ***Factory Method*** **(Método de fabricación):** Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear.

- ***Singleton*** **(Instancia única):** Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

Dentro de los tipos de **Patrones Estructurales** se encuentran:
Patrones de Estructurales de Clase: usa herencia para componer interfaces o implementaciones.

- Herencia Múltiple
- Class Adapter

Patrones de Estructurales de Objeto:

- Object Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

De los cuales se han seleccionado para su utilización en el proyecto los siguientes:

- **Herencia Múltiple:** Una clase que hereda de otras y combina sus propiedades.

- ***Decorator*** **(Envoltorio):** Añade funcionalidad a una clase dinámicamente.

- ***Facade*** **(Fachada):** Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.

Los **Patrones de Comportamiento** que se utilizarán en el proyecto son:

- ***Chain of Responsibility*** **(Cadena de responsabilidad):** Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.

- ***Iterator*** **(Iterador):** Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.

- ***Mediator*** **(Mediador):** Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.

- ***Observer*** **(Observador):** Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.

- ***State*** **(Estado):** Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.

Los patrones de diseño, hacen más fácil reutilizar con éxito los diseños y arquitecturas. Ayudan a elegir entre diseños alternativos. Hacen a un sistema reutilizable y evitan alternativas que comprometen la reutilización. Se trata de patrones ligados al paradigma de programación orientado a objetos y su capacidad potencial no está en su utilización individual sino conjunta para abordar problemas se presentan habitualmente en las fases de diseño y programación.

1.1.7 PATRONES GRASP

GRASP (Patrones Generales de Software para Asignación de Responsabilidades, *General Responsibility Assignment Software Patterns,* en inglés)**.** Una de las cosas más complicadas en la programación orientada a objeto consiste en elegir las clases adecuadas y decidir como estas clases deben interactuar. Para ello se propone el uso de un conjunto de estos patrones, como son:

- **Controlador:** Consiste en asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control.

- **Experto:** Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada.
- **Creador:** Este patrón ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases.
- **Alta Cohesión:** La información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la clase.
- **Bajo Acoplamiento:** Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases.
- **Polimorfismo:** Cuando se identifican variaciones en un comportamiento, asignar la clase (interfaz) al comportamiento y utilizar polimorfismo para implementar los comportamientos alternativos.
- **Fábrica Pura:** Construir clases que se encarguen de construir los objetos adecuados en cada momento (factorías).
- **Variaciones protegidas:** Es el principio fundamental de protegerse del cambio, de tal forma que es susceptible a modificaciones, se envuelva en una interfaz, utilizando el polimorfismo para crear varias implementaciones y posibilitar implementaciones futuras. De manera que quede lo menos ligado posible al sistema, de forma que cuando se produzca la variación, repercuta lo mínimo.

1.2. APLICACIONES EXISTENTES QUE ANTECEDEN ESTA INVESTIGACIÓN

En la actualidad existen en la caja de herramientas de los geólogos y mineros cubanos varios software de uso general, tanto software programados en el país como comerciales. Dentro de estos se encuentran:

GEMCOM

Las soluciones integradas que desarrolla Gemcom abarcan desde las fases de exploración, evaluación de recursos, diseño de minado, optimización, planeamiento minero y control de leyes de producción. Hasta la reconciliación y balance metalúrgico a lo largo de la línea de producción. Desde su fundación en 1985, Gemcom ha estado a la vanguardia de la innovación. En la actualidad es uno de los proveedores

mundiales de soluciones de software de minería. Proporciona un conjunto de servicios informativos y profesionales sobre la ingeniería y análisis de negocios; gestión de programas e implementación de soluciones mineras.

SURPAC

Surpac tiene aplicación en el área geológica minera, específicamente en: exploración geológica; estimación de reservas; diseño en minería a cielo abierto y subterránea; planificación minera y levantamiento topográfico. Tiene una serie de soluciones computacionales para el trabajo geológico minero, entre las cuales se puede mencionar:

- Surpac Vision: Software geológico minero que cubre desde las tareas de exploración hasta la planificación de la mina. Este programa se caracteriza por tener un fácil manejo y gran potencialidad al manejar información de distintos formatos, además de poder realizar conexiones de trabajos múltiples desde internet.

- Minesched: Software de planificación minera de desarrollo y planificación de la producción desde el corto hasta largo plazo en minería subterránea y de cielo abierto. Este programa tiene interfaz directa con MS Project y Excel, permitiendo al planificador obtener flexibilidad y un mejor manejo del plan minero al analizar múltiples opciones en un corto tiempo.

- Tailings: Es un programa que permite simular el llenado de un tranque de relaves, considerando puntos de descarga y diferentes pendientes de depositado.

DATAMINE

Fundada en 1981. El análisis del círculo de planeamiento minero y la solución global propuesta por Datamine comenzó con un riguroso análisis del mercado minero, que luego se fundamentó en la mejora continua de toda la línea de productos y en el desarrollo de todas aquellas tecnologías que permitieran cubrir todas las áreas del negocio minero, en cualquiera de sus variantes.

Es aplicable a todas las áreas del quehacer minero, permitiendo obtener grandes avances en la industria minera del Hierro, del Oro, Níquel, Fosfatos, Diamantes, Cobre, Bauxita, Carbón, Lignite, Platino, Petróleo, y otros minerales. Los usos más comunes del sistema son; la captura y análisis de la información,

exploración, geología, geoquímica, mecánica de rocas, topografía, modelamiento geológico, diseño de mina a cielo abierto y subterráneas, planeamiento minero y áreas relacionadas a los estudios ambientales.

1.3. ESTADO ACTUAL DEL NEGOCIO

A continuación se describe el estado actual del negocio, donde se fundamenta el objetivo y el problema planteado en la investigación.

1.3.1 FUNDAMENTACIÓN DEL PROBLEMA

Para la rama de la minería cubana, se tomó la decisión de adquirir licencias de software, comenzando con GEMCOM y más tarde SURPAC, DATAMINE y otros que fueron adquiridos por entidades nacionales o introducidos por los socios extranjeros en las empresas mixtas formadas para el níquel y otros minerales, que si bien respondieron a ciertas urgencias, también impusieron una dependencia tecnológica, trayendo consigo el costoso pago de versiones, licencias, capacitación y servicios.

Por estas razones, se plantea de forma implícita que la solución a la problemática minera es la creación de un software cubano de minería. Para un diseño correcto del mismo se hace necesario identificar los componentes que conformarán el sistema a desarrollarse y las relaciones existentes entre los mismos, aspectos que maneja, diseña y detalla la arquitectura de software como tal. Obteniéndose un diseño arquitectónico basado en tecnologías no privativas se desarrollará una aplicación informática bien diseñada, con una correcta interacción entre los componentes que la conforman y una mejor representación de los requisitos funcionales y no funcionales identificados.

En esta investigación se pretende dar solución al problema a resolver: ¿Cómo definir la organización estructural de los componentes del sistema de software minero GeolMin?, de forma tal que se conforme una arquitectura robusta para el software, que permita establecer una línea base, que sirva de guía a los trabajadores y usuarios del sistema.

1.3.2 FUNDAMENTACIÓN DEL OBJETIVO

Uno de los problemas más comunes en el desarrollo de software es que el tiempo aplicado a este proceso sea muy extenso. Esto es provocado por una mala definición de la arquitectura, lo que influye en el proceso del mismo. Otro de los factores que atentan contra el desarrollo ágil del software es que se encuentra poca o ninguna documentación que el sistema pueda tener, lo que provoca que el código generado pocas veces se pueda reutilizar.

El proyecto Minería, no cuenta con una metodología que guíe a los involucrados en el desarrollo del sistema del software minero GeolMin. Además no cuenta con documentos que definan las vistas de la arquitectura de la aplicación, soporte tecnológico a desarrolladores, clientes y expertos en negocios; tampoco se ha conceptualizado ni experimentado con distintos enfoques arquitectónicos. Por lo que se hace necesario analizar patrones y estilos arquitectónicos alternos para derivar la estructura que corresponda con los requisitos del cliente. Además crear documentos de modelos y componentes, especificaciones de interfaces y validar la arquitectura contra requerimientos y suposiciones.

Por estas razones el objetivo general que se persigue con esta investigación, es diseñar la arquitectura de software del sistema de software minero GeolMin, con la que se espera obtener un producto final exitoso. Esto ayudará a obtener un sistema de software minero completo, eliminando dificultades presentadas por las herramientas de software producidas por el país anteriormente.

1.4 TENDENCIAS Y TECNOLOGÍAS A UTILIZAR

El creciente uso de soluciones informáticas en los procesos productivos y sociales, ha ocasionado el crecimiento en la confiabilidad y la calidad que estos deben tener. Para ello en la actualidad se cuenta con un conjunto de herramientas, tecnologías y metodologías que aseguran la calidad, así como el empleo de buenas prácticas para el desarrollo las mismas. Se indagará sobre sus principales características, ventajas y desventajas, así como los beneficios que proporcionan para su selección y aplicación en el sistema a desarrollar.

1.4.1 METODOLOGÍA DE DESARROLLO A UTILIZAR

Una metodología de desarrollo de software es un proceso en el que se define “quién” está haciendo “qué”, “cómo” y “cuándo”. Las metodologías son el conjunto de técnicas y procedimientos que permiten conocer los elementos necesarios para definir un proyecto de software, es la etapa fundamental para lograr los objetivos buscados con dicho proyecto, básicamente sirve para subir la "calidad" del software. Con el objetivo de buscar una metodología que se ajuste al medio en que se desarrollará el sistema, se analizaron algunas de las más usadas en el mundo.

Dentro de las metodologías analizadas se encuentran: Programación extrema (XP, por sus siglas en inglés), Proceso Unificado de Rational (RUP), *Feature Driven Development* (FDD), *Microsoft Solution Framework* (MSF) y *Agile Unified Process* (AUP). La información referente a cada una de ellas se recopiló

en el documento del estado del arte sobre las metodologías candidatas a utilizar en el proyecto, donde se realizó una comparación teniendo en cuenta características, ventajas y desventajas.

Por sus atributos y gran aceptación, se decidió escoger RUP como metodología a utilizar para el desarrollo del sistema GeolMin. RUP es la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. No es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización. Es apropiada para proyectos grandes, requiere de un equipo de trabajo capaz de administrar un proceso complejo en varias etapas, guiando a cada trabajador durante todo el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue.

Estas razones ayudan a la selección de la metodología RUP para guiar el desarrollo del sistema permitiendo estar al tanto de la evolución del software en cuatro fases (Inicio, Elaboración, Construcción, Transición), lo que establece objetivos a alcanzar bien definidos. Otro de los motivos de su selección, son las principales características que esta presenta:

- **Dirigida por casos de usos:** Basándose en los casos de uso, los desarrolladores crean una serie de modelos de diseño e implementación que los llevan a cabo.
- **Centrada en la arquitectura:** Describe a través de representaciones en diferentes modelos, los elementos más representativos del sistema a desarrollar.
- **Iterativo e incremental:** Mediante las fases en que se divide, permite agilizar el tiempo de desarrollo del sistema.

La representación de la arquitectura será mediante el modelo de las 4+1 vistas que utiliza RUP y es propuesto por Philippe Kruchten. Esencialmente estas vistas son extractos que ilustran los elementos "significativos arquitectónicamente" de los modelos. Este uso de múltiples vistas permite abordar los intereses de los distintos *stakeholders*[1] de la arquitectura por separado: usuarios finales, desarrolladores, ingenieros de sistemas, administradores de proyecto, y manejar los requisitos funcionales y no funcionales

[1] Desde el punto de vista del desarrollo de sistemas, es aquella persona o entidad que está interesada en la realización de un proyecto o tarea.

separadamente. Las vistas se diseñan mediante un proceso centrado en la arquitectura, motivado por escenarios y desarrollado iterativamente. (Kruchten, 1995)

1.4.2 LENGUAJE DE MODELADO

Al definir RUP como la metodología de desarrollo a utilizar, se debe escoger el lenguaje de modelado a utilizar. El cual permitirá describir con cierto grado de formalismo, para que puedan ser entendidos por los clientes o usuarios, aquello que se modela.

El lenguaje a utilizar será UML en su versión 2.0, por ser el más eficiente y apropiado como medio potente y efectivo para describir, administrar y modelar el desarrollo de software. El objetivo de utilizar este lenguaje, es avanzar de la especificación inicial a un plan de implementación y para comunicar dicho plan a todo el equipo de desarrolladores del software minero GeolMin.

El Lenguaje Unificado de Modelado (UML) se utiliza para visualizar, especificar, construir y documentar los artefactos de un sistema software. RUP propone el uso de este lenguaje de modelado para la descripción de los artefactos propuestos por sus flujos de trabajo en cada una de las distintas fases. El proceso de RUP combinado con UML constituye la metodología más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

UML tiene una gran cantidad de propiedades que han sido las que realmente, han contribuido a hacer de él un estándar para la industria de software. Algunas de estas propiedades son:

- Concurrencia, es un lenguaje distribuido y adecuado a las necesidades de conectividad actuales y futuras.
- Reemplaza a decenas de notaciones empleadas con otros lenguajes.
- Modela estructuras complejas.
- Las estructuras más importantes que soportan tienen su fundamento en las tecnologías orientadas a objetos, tales como objetos, clase, componentes y nodos.
- Emplea operaciones abstractas como guía para variaciones futuras, añadiendo variables si es necesario.

1.4.3 HERRAMIENTA DE MODELADO

Las herramientas CASE (Ingeniería de Software Asistida por Computadora) fueron creadas para el desarrollo de la ingeniería de software. Se acoplan con las metodologías para dar una forma de representar sistemas y suponen una forma de abstracción del código fuente, a un nivel donde la arquitectura y el diseño se hacen más aparentes y fáciles de entender y modificar. Entre las herramientas CASE orientadas a UML están: Poseidon for UML, ArgoUML, Enterprise Architect, MagicDraw UML, Borland Together, Rational Rose y Visual Paradigm, siendo las dos últimas las más utilizadas, y las que se analizarán en esta investigación.

Rational Rose

Es la herramienta que permite construir y desarrollar a través del UML los casos de uso en diferentes diagramas, modelando los flujos de trabajo por los que transita el desarrollo de un software. Permite la realización de los diferentes diagramas y la posterior generación del código, todo orientado a objetos. Posee librerías que facilitan la obtención de una ingeniería inversa sobre diferentes lenguajes como Java, C++, Corba, XML_DTD, ADA, Visual Basic. Esta herramienta posibilita gestionar la evolución del ciclo de vida de un proyecto de software. Racional Rose aligera la implementación al automatizar los modelos arquitectónicos. Además, permite visualizar, entender y refinar los requerimientos y arquitectura antes de introducirse en el código.

Visual Paradigm

Es una herramienta multiplataforma pues posee la capacidad de ejecutarse sobre diferentes sistemas operativos. Es muy fácil de usar, ya que presenta un ambiente gráfico agradable e intuitivo para el usuario. Visual Paradigm permite además, dibujar todos los tipos de diagramas de clases, generar código desde diagramas y viceversa. Cuenta con las siguientes características principales:

- Licencia: Gratuita y Comercial.
- Generación de código en diferentes lenguajes de programación.
- Generación de código para distintos SGBD (Sistemas Gestores de Bases de Datos).
- Se puede integrar con diferentes IDE (Entorno de Desarrollo Integrado).
- Se integra eficientemente con subversión.

- Genera documentación de lo modelado.
- Fácil de instalar y actualizar.

Para el modelado del sistema se ha seleccionado Visual Paradigm, la cual es una poderosa herramienta CASE que utiliza UML para el modelado. Esta herramienta soporta el ciclo de vida completo del desarrollo de software, análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Además ayuda a una rápida construcción de la aplicación, proporcionando una alta calidad y mejoras a un menor tiempo de desarrollo. Es una herramienta colaborativa, pues soporta a múltiples usuarios trabajando sobre el mismo proyecto y a la vez permite realizar el control de versiones, característica que tiene un impacto positivo en proyectos que cuentan con amplios equipos de desarrollo como el sistema GeolMin.

Visual Paradigm permite además, dibujar todos los tipos de diagramas de clases, generar código desde diagramas y viceversa. Genera la documentación del mismo automáticamente en varios formatos como Web o Pdf (*Portable Document Format*), la cual puede ser utilizada para la conformación de los documentos del Expediente de Proyecto. La herramienta CASE también facilita demostraciones interactivas, abundantes tutoriales y proyectos UML. Es una herramienta multiplataforma, permite su uso en cualquier sistema operativo y presenta una licencia comercial y gratuita, siendo estas unas de las ventajas que presenta con respecto al Rational Rose.

1.4.4 LENGUAJE DE PROGRAMACIÓN

Un lenguaje de programación es un sistema notacional para describir computaciones en una forma legible tanto para la computadora como para los usuarios. (Louden, 2004)

Cada lenguaje tiene ventajas y desventajas que lo hacen más o menos potente ante una necesidad de software determinada, por lo que no se puede afirmar que exista un lenguaje que cubra todas las necesidades requeridas por un software. De aquí que la importancia de que a la hora de seleccionar el lenguaje de programación para un determinado proyecto de software, se haga un análisis de estos a partir de las necesidades de software. Son muchos los lenguajes que pudieran satisfacer las necesidades del proyecto, pero se decidió enfatizar en el análisis de los que más se utilizan en la UCI. Entre los que se encuentran:

- C++
- Java
- Python

Analizando las características de los lenguajes que anteriormente se mencionan, se decide usar C++. Para su selección se tuvieron en cuenta varios factores como opciones y facilidades que brindan así como sus debilidades. Este a pesar de ser un lenguaje muy reconocido, es un poco difícil de entender y presenta algunas limitaciones en su programación. Es un lenguaje de programación extremadamente largo y complejo.

Entre sus principales características posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel: posibilidad de redefinir los operadores (sobrecarga de operadores), e identificación de tipos en tiempo de ejecución. Está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel. Los compiladores de C++ generan código nativo con un alto grado de optimización en memoria y velocidad, minimizando la implementación de recursos tales como el polimorfismo y la expansión de patrones lo que lo convierte en uno de los lenguajes más eficientes.

Basándose en las ventajas que ofrece el lenguaje en cuanto a eficiencia, soporte para programación orientada a objeto, optimización y velocidad de generación de código de los compiladores y la implementación rápida y fácil de programas con el uso de la biblioteca VTK[2], es que se decide escoger C++ como lenguaje de programación. Siendo VTK uno de los elementos necesarios para la visualización de los diferentes objetos con los que trabajará el sistema.

La información de los lenguajes mencionados se recoge en el documento de Estudios Comparativos sobre los Lenguajes de Programación. Que es otro de los documentos que ayudan a la descripción y documentación de esta investigación.

[2] ***Visualization Toolkit:*** Es un conjunto de librerías de clases de C++. Consiste en un sistema de visualización por software que permite la representación de geometrías 2D y 3D, soportando una amplia variedad de algoritmos de visualización y modelado.

1.4.5 HERRAMIENTAS DE DESARROLLO

Las herramientas de desarrollo o Framework tienen como objetivo fundamental, facilitar el desarrollo de software.

Un Framework define una arquitectura adaptada a las particularidades de un determinado dominio de aplicación, definiendo de forma abstracta una serie de componentes[3] y sus interfaces, y estableciendo las reglas y mecanismos de interacción entre ellos. (Perera, 2007)

Framework QT

Es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario producida por la división de software Qt de la empresa Nokia, que entró en vigor después de la adquisición por parte de Nokia de la empresa noruega *Trolltech*. Es robusto y ha posibilitado la implementación de grandes proyectos como Entorno de Escritorio K (KDE)**,** *Google Earth***,** *Skype***,** *VirtualBox*, etc. Distribuida bajo los términos de GNU (*General Public License*), es software libre y de código abierto. Cuenta actualmente con un sistema de triple licencia: GPL v2/v3 para el desarrollo de software de código abierto y software libre y la licencia de pago QPL (*Q Public License*) para el desarrollo de aplicaciones comerciales.

QT Creator

Es un excelente Entorno Multiplataforma de Desarrollo Integrado (en inglés: *Integrated Development Environment* o IDE), compuesto por una serie de herramientas que utilizan los programadores para desarrollar código. Se utiliza para realizar aplicaciones en C++ de manera sencilla y rápida. Está basado en la librería QT y cuenta con las siguientes características principales:

- Editor avanzado para C++.
- Herramientas para la administración y construcción de proyectos.
- Completado automático y resaltado de código.
- Depurador visual.
- Soporta lenguajes como: C#/.NET (Qyoto), Python (PyQt), PHP (PHP-QT), Java (QTJambi), Ada, Pascal, Perl, PHP y Ruby.
- Está disponible para las plataformas: Linux, Mac OSX; Windows, Windows CE, Symbian y Maemo.

[3] Parte física y reemplazable de un sistema que se ajusta y proporciona la realización de un conjunto de interfaces.

- Es distribuido bajo tres tipos de licencias: *Qt Commercial Developer License*, Qt GNU LGPL v. 2.1 y Qt GNU GPL v. 3.0.

De acuerdo a las características que presenta y facilidades de uso, se determinó como IDE a utilizar el QT Creator, que trae integrado el conjunto de bibliotecas para el desarrollo de interfaces gráficas de QT y ofrece una gama amplia de clases para el trabajo con el lenguaje de programación C++. Esto hace más cómodo el trabajo del programador a través de la reutilización de código y se logra una muy buena fluidez con el lenguaje. Presenta gran estabilidad y desempeño al implementar programas con un código más legible y organizado.

1.4.6 HERRAMIENTA DE GESTIÓN DE BASE DE DATOS

Un Sistema Gestor de Bases de Datos (SGBD), consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a esos datos. El principal objetivo de un SGBD es proporcionar un entorno que sea a la vez conveniente y eficiente para ser utilizado al extraer y almacenar información de la base de datos. (IGAC, 2009)

Base de Datos Embebidas.

Las bases de datos embebidas operan simbólicamente dentro de la aplicación. Lo que significa que su código esta entrelazado o incrustado como parte del programa que lo alberga. Estas reducen los gastos relacionados con la red en cuestión de llamadas, simplifica la administración y el despliegue de la aplicación. Por lo que se hace innecesaria la configuración de una red o su administración. Entre las Bases de Datos Embebidas se encuentra SQlite, la cual se describe brevemente a continuación.

SQlite

La necesidad de que persista la información en el software minero GeolMin, requiere la selección de alguna herramienta que facilite la gestión de los datos, por lo que se determinó utilizar el gestor de bases de datos SQlite en el desarrollo del sistema. La selección estuvo condicionada por varias razones, entre ellas se encuentran:

- SQLite tiene una pequeña memoria y una única biblioteca necesaria para acceder a bases de datos, lo que lo hace ideal para aplicaciones de bases de datos incorporadas.
- Realiza operaciones de manera eficiente y es más rápido que MySQL y PostgreSQL.

- Se ejecuta en muchas plataformas y sus bases de datos pueden ser fácilmente portadas sin ninguna configuración o administración.
- Implementa un gran subconjunto de la ANSI – 92 SQL estándar, incluyendo sub-consultas, generación de usuarios, vistas y triggers.
- Cuenta con diferentes interfaces del API, las cuales permiten trabajar con C++, PHP, Perl, Python, Ruby, Tcl, groovy, etc.
- SQLite es de dominio público, y por tanto, es libre de utilizar para cualquier propósito, sin costo y se puede redistribuir libremente.
- Cuenta con Registros de longitud variable, lo que permite tener como resultado un pequeño archivo de base de datos y optimización de la velocidad de la misma puesto que hay menos información desperdiciada que leer y recorrer.

Por estas características se utiliza este SGBD además de la necesidad de trabajar en máquinas independientes y almacenar la información sin utilizar un servidor de bases de datos o de ficheros, buscando optimizar el tiempo de trabajo mientras se va desarrollando el sistema. Además que cumple con las características básicas para desarrollar la aplicación. Una vez instalado el sistema y listo para su utilización, se continuará el uso de este SGBD para hacer salvas locales de la información, asegurando la integridad del sistema.

Postgre SQL

El sistema está enfocado a ser una aplicación Cliente-Servidor para la cual es necesario utilizar un gestor de bases de datos en el Servidor. Ante esta necesidad se tomó la determinación de utilizar Postgre SQL.

La selección estuvo condicionada por varias razones, entre ellas se encuentran:

- Postgre SQL es altamente extensible, soporta los tipos de datos base, así como: fechas, elementos gráficos, cadenas de bits, entre otros.
- Soporta integridad referencial, la cual es utilizada para garantizar la validez de la información de la base de datos.
- Las funciones se pueden escribir en varios lenguajes. Tiene su propio lenguaje PLP/SQL y es capaz de interpretar funciones implementadas en TCL, Java, Perl, Python, C y C++.

- Usa una arquitectura proceso-por-usuario cliente/servidor. Teniendo un proceso maestro que se ramifica para proporcionar conexiones adicionales para cada cliente que se intente conectar.
- Postgre SQL corre en la mayoría de los Sistemas Operativos más utilizados incluyendo, Linux y Windows.

También se realizó un análisis de algunos SGBD como Oracle y SQLServer, las cuales a pesar de ser herramientas potentes, su uso se ve limitado al ser privativas y su adquisición muy costosa. Otro de los gestores analizados fue MySQL, que es de código abierto, robusto y rápido, pero no está diseñado para aplicaciones grandes, siendo esta una de sus limitantes al ser el sistema a desarrollar grande y complejo. Postgre SQL posee características de código abierto, constituyendo uno de los SGBD libres más avanzado. Está derivado del paquete Postgres escrito en Berkeley, distribuida bajo licencia BSD[4]. Los requisitos para su instalación son mínimos y posibilita realizar copias de seguridad evitando la pérdida de la información.

Teniendo en cuenta las políticas de migración a software libre de la universidad, y las características mencionadas anteriormente; Postgre SQL es el que más se adecua a las necesidades del sistema, para almacenar grandes volúmenes de datos geológicos y permitir el acceso de diferentes usuarios al mismo tiempo, acreditando permiso a cada uno de ellos.

1.5 TENDENCIA DEL ROL DE ARQUITECTO DE SOFTWARE

El arquitecto de software crea la arquitectura del sistema junto con otros desarrolladores para conseguir que un sistema tenga un alto rendimiento, calidad y sea completamente funcional. Este además posee la responsabilidad técnica más importante, pues selecciona patrones de arquitectura, estilos y tecnologías para establecer las dependencias entre subsistemas para cada uno de esos distintos intereses.

El verdadero objetivo de un arquitecto de software es cumplir con las necesidades de la aplicación de la mejor forma posible, con el estado actual de la tecnología y un coste que la aplicación pueda soportar. En otras palabras, ser capaz de implementar la funcionalidad de la aplicación, es decir, (los casos de uso) de manera económica, ahora y en el futuro.

[4] Es la licencia de software otorgada principalmente para los sistemas BSD (*Berkeley Software Distribution*). Es una licencia de software libre permisiva, con menos restricciones en comparación con la GPL, estando muy cercana al dominio público. Esta licencia permite el uso del código fuente en software no libre.

1.5.1 FUNDAMENTACIÓN DEL ROL DE ARQUITECTO DE SOFTWARE

El rol del arquitecto dentro de un equipo de desarrollo de software, es dirigir el desarrollo de la Arquitectura de Software del sistema. Esto incluye la promoción y creación de soporte para las decisiones técnicas claves, que restringen el diseño global y la implementación para el proyecto.

Esto habitualmente incluye la identificación y la documentación de los aspectos arquitectónicamente significativos del sistema, que incluye Línea base de la arquitectura, Documento de Descripción de Arquitectura de Software, Arquitectura de Información, Modelo de análisis, Modelo de diseño, Modelo de despliegue, Modelo de implementación, Prototipos de arquitectura, visualizar el comportamiento del sistema, crear los planos del sistema, definir la forma en la cual los elementos del sistema trabajan en conjunto, y proveer una guía técnica clara y consistente.

Además, establecer la estructura global de cada vista de la arquitectura: la descomposición de las vistas, el agrupamiento de elementos y las interfaces de los mismos. La metodología RUP señala que los arquitectos moldean el sistema para darle una forma. La arquitectura debe diseñarse para permitir que el sistema evolucione, no sólo en su desarrollo inicial, sino también a lo largo de las futuras generaciones. Para encontrar esa forma, los arquitectos deben trabajar sobre la comprensión general de las funciones claves, es decir, sobre los casos de uso claves del sistema.

1.5.2 ARTEFACTOS CONSTRUIDOS

Los principales Artefactos a desarrollar por el Arquitecto son: (IBM, 2003)

- Documento Descripción de la Arquitectura (4+1 Vistas)
- Modelo de Despliegue.
- Modelo de Implementación
- Protocolos.
- Interfaces.

Las actividades principales realizadas por el Arquitecto deben ser:

- Análisis de la arquitectura.
- Priorizar los Casos de Uso.
- Identificar mecanismos de diseño.
- Estructurar el modelo de implementación.

- Reutilización de elementos de diseño existentes.
- Identificar los elementos de diseño.
- Describir la arquitectura en tiempo de ejecución.

1.6 CONCLUSIONES PARCIALES

En este capítulo se realizó un estudio detallado sobre los temas relacionados con la arquitectura de software, mostrándose conceptos fundamentales para su diseño. Se expusieron las características, ventajas y desventajas de las principales tecnologías y herramientas empleadas en el desarrollo de software, influenciando en la utilización del software libre.

Se definieron estilos y patrones de diseño que se utilizarán en las fases de diseño y programación, con el objetivo de obtener un sistema reutilizable. Se definió RUP como metodología a utilizar entre las estudiadas, utilizando UML para el modelado de los artefactos propuestos. Después de un estudio organizado y fundamentado, se determinan las tecnologías que van a ser utilizadas en el proceso práctico de la investigación. Se crean las bases para dar comienzo a la siguiente fase de la investigación.

CAPÍTULO #2

LÍNEA BASE DE LA ARQUITECTURA

En este capítulo se hace una propuesta de solución al problema planteado anteriormente en el diseño teórico de la investigación científica. La solución está regida por el documento rector de la arquitectura de software del proyecto de Minería, mediante una serie de vistas arquitectónicas para representar diferentes aspectos del sistema. Las descripciones se hacen a través de las 4+1 vistas: una vista de Procesos, una vista Lógica, una vista de Despliegue, una vista de Implementación y una vista de Casos de Uso.

2.1 ESTRUCTURA DEL EQUIPO DE DESARROLLO

El equipo de trabajo del proyecto Minería está distribuido de la siguiente forma:

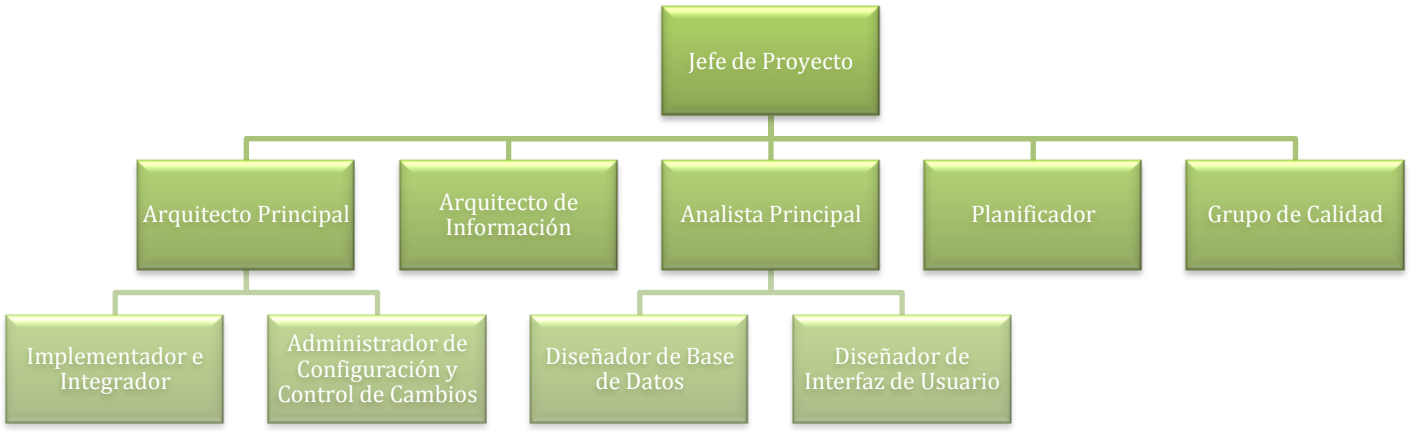

Figura 1: Estructura del equipo de desarrollo del proyecto Minería.

2.2 DESCRIPCIÓN DE LA ARQUITECTURA

Con el objetivo de lograr una mejor comprensión del sistema, organizar el desarrollo y fomentar la reutilización; contribuyendo de esta forma a una evolución del sistema más rápida y eficaz, a continuación se realiza la descripción de la Arquitectura.

Esta descripción tiene como propósito fundamental proporcionar una comprensión arquitectónica del sistema a clientes y otros usuarios, además de servir de guía para el equipo de trabajo durante todo el ciclo de desarrollo de la aplicación. La descripción se realiza utilizando las vistas arquitectónicas definidas por la metodología RUP, que muestran las diferentes características del sistema. Estas vistas serán representadas en UML utilizando Visual Paradigm como herramienta CASE de modelado.

2.2.1 PROPUESTA DE ARQUITECTURA DE SOFTWARE

Para el desarrollo del sistema GeolMin, se propone una arquitectura de software estructurada de la siguiente forma:

1. **Modelo base de arquitectura:** Cliente-Servidor.
2. **Estilo arquitectónico:** Se combinará el uso de tres estilos, estos son:
    - Arquitectura basada en Componentes.
    - Arquitectura Orientada a Objetos.
    - Arquitectura en Capas, (2 capas, 3 capas, 6 capas). *(ver Figura 7)*
3. **Patrones de diseño:** Ver epígrafe 1.1.7 Patrones de Diseño.
4. **Patrones GRASP:** Ver epígrafe 1.1.8 Patrones GRASP.
5. **Tipo de aplicación:** Desktop.

La arquitectura cliente/servidor es un modelo para el desarrollo de sistemas de información en el que las transacciones se dividen en procesos independientes que cooperan entre sí para intercambiar información, servicios o recursos. Se denomina cliente al proceso que inicia el diálogo o solicita los recursos y servidor al proceso que responde a las solicitudes. Este modelo a su vez estará dividido en dos capas, como se muestra en la Figura 2.

Capa cliente: Está diseñada para soportar todas las funcionalidades y procesamiento de la información.

Capa servidor: Esta diseñada para soportar y almacenar toda la información de los datos geológicos. Esto hace referencia a los servidores de bancos de datos y de ficheros.

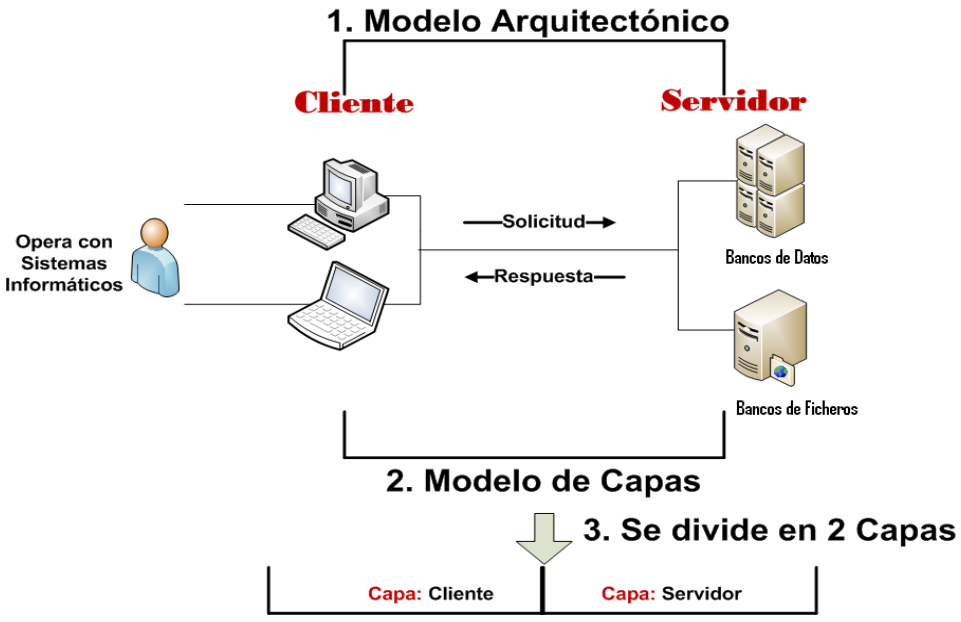

Figura 2: Modelo Cliente/Servidor.

2.3 REQUISITOS NO FUNCIONALES

Los requisitos no funcionales son propiedades o cualidades que hacen relación a las características del sistema que se aplican de manera general como un todo, más que a cualidades particulares del mismo. Estas propiedades hacen al producto atractivo, amigable, usable, rápido y confiable. Siendo la base de la arquitectura de software. El sistema GeolMin deberá cumplir con los siguientes requisitos no funcionales:

2.3.1 USABILIDAD

- La aplicación tendrá un ambiente sencillo y será fácil de manejar para los usuarios.
- El sistema contará con un demo, el cual explicará cómo se trabaja con la aplicación. Además se impartirán cursos de capacitación.

2.3.2 CONFIABILIDAD

- La probabilidad de que el sistema funcione o desarrolle las funciones de trabajo con los datos geológicos en el terreno debe ser alta. Si este presentara algún desperfecto, será corregido en un período menor a un mes.

2.3.3 RENDIMIENTO

- El sistema deberá ser capaz de manipular grandes cantidades de datos, menor o igual a 1 Mg hasta 50 Mg.

2.3.4 EFICIENCIA

- La velocidad de procesamiento de la información, la actualización y la recuperación de la información, dependerán de la cantidad de información que tenga que procesar la aplicación.

2.3.5 SOPORTE

- El sistema una vez implementado deberá presentar un manual de usuario.
- El mantenimiento que recibirá la aplicación se efectuará en el tiempo estimado por el equipo de desarrollo y los clientes.

2.3.6 INTERFAZ DE USUARIO

- El sistema debe presentar interfaces amigables, de fácil uso para los usuarios.
- Se deberá mostrar de forma organizada las funcionalidades de la aplicación.

2.3.7 HARDWARE

Para Estaciones de trabajo:

- 1 Gb de espacio libre en disco.
- 1 Gb de RAM recomendable, mínimo 512 Mb.
- Tarjeta de Gráfico 512 Mb mínimo.

2.3.8 SOFTWARE

Para Estaciones de trabajo:

- Sistema Operativo: Windows XP o superior. Cualquier versión de Linux.

Servidor de Base de Datos:

- Gestor de Base de Datos: SQlite 3.0 y PostgreSQL.

2.3.9 SEGURIDAD

- Confidencialidad: La información manejada por el sistema deberá estar protegida de acceso no autorizado y divulgación.

- Integridad: La información manejada por el sistema debe ser objeto de cuidadosa protección contra la corrupción y estados inconsistentes. Se incluye también mecanismos de chequeo de integridad.

2.3.10 RESTRICCIONES DEL DISEÑO E IMPLEMENTACIÓN

El diseño y la implementación de la arquitectura deberá permitir la fácil integración o separación de componentes.

- Se utilizará el estándar de código propuesto por la universidad para el desarrollo de aplicaciones sobre C++.

Interoperabilidad:

- Se desarrollará bajo el estándar ANSI[5] de C++ como lenguaje de programación y QT como framework de desarrollo, en Qt Creator.

- Se utilizará la herramienta Case *Visual Paradigm for UML 2.0* para el diseño y modelado.

Uso de estándares de datos:

- Se utilizará la notación camello. Para más especificaciones revisar el documento de "Especificaciones de estándares de codificación".

Otras restricciones:

- El sistema operará con múltiples ficheros (*.xml, *.dat, *.las, *.map, *.txt, *.lis, *.jpg) donde se almacenan los datos geológicos adquiridos en el campo.

2.4 VISTAS ARQUITECTÓNICAS

A continuación se desarrolla la descripción de la arquitectura a través de las 4+1 vistas propuestas por Kruchten.

[5] Instituto Nacional Estadounidense de Estándares (ANSI, por sus siglas en inglés: American National Standards Institute).

2.4.1 VISTA DE CASOS DE USOS

La vista de casos de usos en el marco arquitectónico, representa los casos de usos arquitectónicamente significativos; seleccionados a partir de los catalogados de críticos en cada uno de los módulos. Estos serían los que modelan las principales funcionalidades del sistema de acuerdo con las exigencias del cliente.

Los casos de usos que se presentan y comentan a continuación son la base para el posterior funcionamiento del sistema, no se puede realizar ninguna de las operaciones propuestas si no se cuenta con la base conceptual para realizarlas. A partir de estas funcionalidades, se puede comprobar que la arquitectura funciona para los elementos fundamentales, además de su comportamiento estable.

Subsistema del Módulo CORE

Iniciar la aplicación: Este caso de uso permite iniciar la ejecución del programa.

Cargar configuración base: Permite cargar la configuración base del sistema para crear una estructura de configuración al ser inicializado el sistema.

Cargar plugins: Permite localizar el directorio de los plugins, carga los que están instalados por el sistema y devuelve un objeto del mismo.

Cargar la internacionalización: Permite crear una estructura de internacionalización para el idioma de la aplicación.

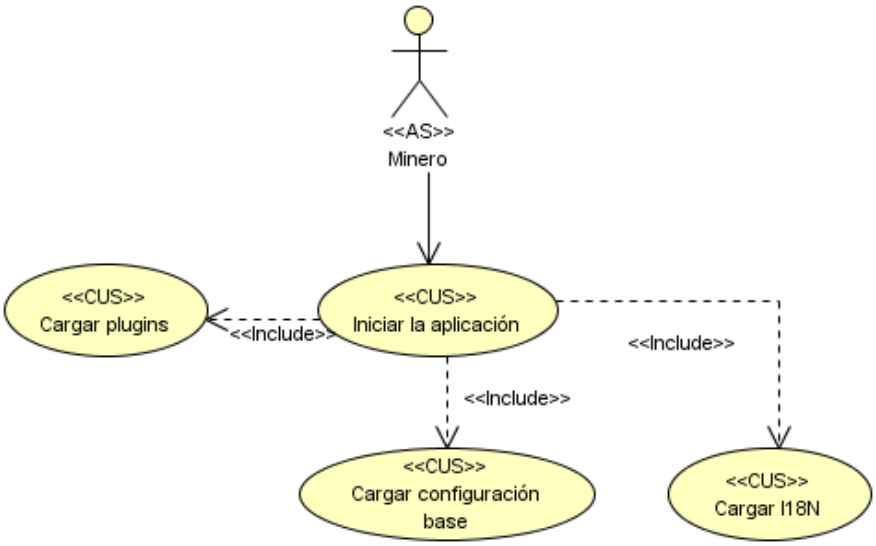

Figura 3: CU Arquitectónicamente significativos del Módulo CORE.

Subsistema del Módulo Gestión de la Información:

Crear proyecto: Este caso de uso permite crear un nuevo proyecto.

Importar proyecto: Permite cargar un proyecto guardado anteriormente.

Modelar pozo: Permite mediante una modelación en 3D y 2D, dar una idea de los pozos[6] con los que trabaja.

Cargar fichero: Este caso de uso se lleva a cabo con el objetivo de cargar los ficheros que almacenan la información generada por los pozos de sondeo.

[6] Son pozos de sondeo que se hacen en forma rápida para determinar la profundidad de la estratigrafía del terreno.

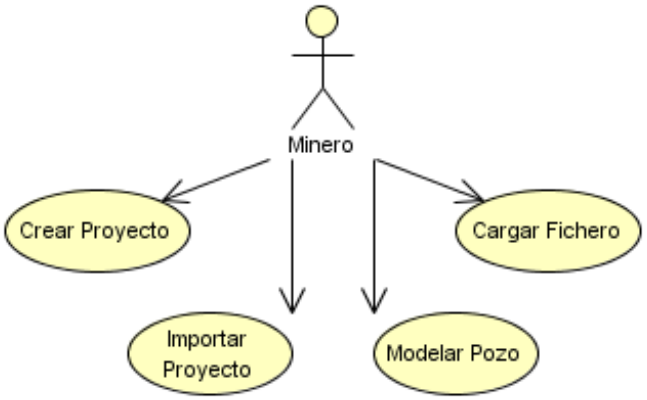

Figura 4: CU Arquitectónicamente significativos del Módulo Gestión de la Información.

Subsistema del Módulo de Visualización:

Visualizar objetos geométricos: Este caso de uso permite visualizar en el espacio 3D las figuras geométricas.

Visualizar objetos geológicos: Se visualizan en el espacio 3D los pozos de sondeos, el modelo de terreno digital y el cuerpo de minerales.

Proyectar figuras geométricas: Permite posicionar las figuras en un plano del espacio tridimensional.

Modelar plano de trabajo: Permite diseñar los objetos por planos.

Diseñar objetos geológicos: Permite dibujar por plano las geometrías que encierran al mineral según la representación del mismo en los pozos. Luego se triangulan los puntos para conformar el cuerpo.

Editar objetos geométricos: Permite reconstruir una figura nuevamente en caso de que se haya cometido algún error al diseñar.

Visualizar vista aérea: Permite visualizar una serie de puntos los cuales son la cabeza o entrada del pozo de sondaje, perteneciente al plano X-Z.

Visualizar vistas de perfiles: Este caso de uso muestra en el área de trabajo los pozos en los planos X-Y o Z-Y.

Obtener Bounding Box: Este caso de uso delimita el área donde se encuentran los objetos geológicos.

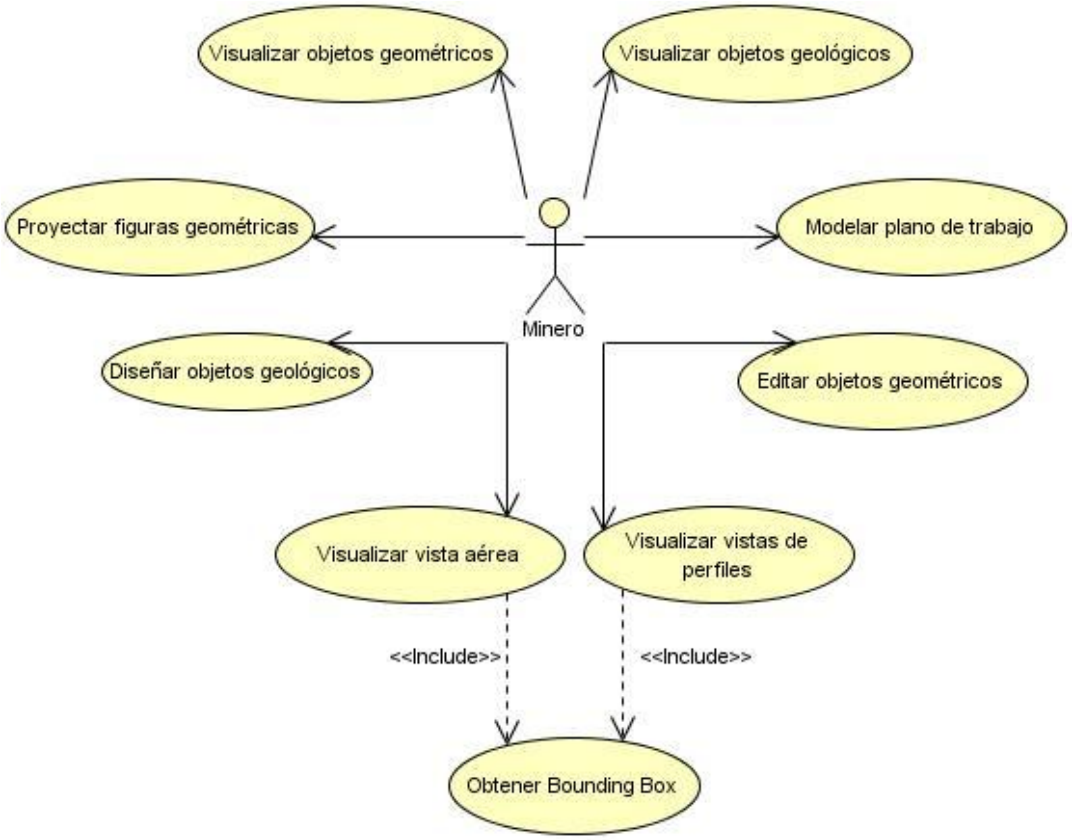

Figura 5: CU Arquitectónicamente significativos del Módulo de Visualización.

Subsistema del Módulo de Manipulación de Datos:

Exportar información: Permite exportar la información de los pozos que fue guardada en la base de datos a través de los ficheros.

Validar información: Este caso de uso permite detectar si la información tiene el formato correcto para luego proceder a trabajar con la misma.

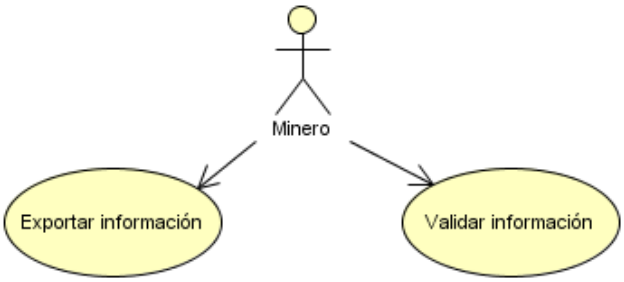

Figura 6: CU Arquitectónicamente significativos del Módulo de Manipulación de Datos.

2.4.2 VISTA LÓGICA

La vista lógica se encarga de describir la organización interna del sistema, siendo la base de la implementación. Describe las clases más importantes, su organización en paquetes y subsistemas, y la organización de estos subsistemas en capas. Esta descomposición no sólo se hace para potenciar el análisis funcional, sino también sirve para identificar mecanismos y elementos de diseño comunes a diversas partes del sistema. Principalmente apoya los requisitos funcionales, lo que el sistema debe brindar en términos de servicios a sus usuarios.

Visión general de la arquitectura – Alineamiento de paquetes, subsistemas y capas.

La Figura 7, representa la organización de las capas dentro de la capa cliente. Se utiliza para organizar este nivel el estilo en capas, “Presentación”, “Negocio”, “Datos”. Cada capa se dividió en 2 subcapas o capas más específicas, para lograr una mejor cohesión, organización y distribución de las funcionalidades del sistema.

1. **Capa de Presentación:**
    - **Componentes de Interfaz de Usuario:** Recoge los componentes, su declaración y su configuración base.
    - **Lógica de Interfaz de Usuario:** Recoge la implementación de los SIGNAL (Señales o eventos), SLOTS (Funcionalidades que ejecuta un bloque de intrusiones) y el procesamiento que se ejecute a nivel de interfaz, además de las validaciones.
2. **Capa de Negocio:**
    - **Control y ejecución (Patrón-Facade):** Recoge las clases y funcionalidades de control y

ejecución de las transacciones que llegan desde la capa de interfaz de usuario. Sirve como intermediario o mediado entre la lógica de presentación y la lógica de negocio. En este nivel se usará como interfaz principal el patrón fachada.

- **Objetos de negocio:** Recoge todas las clases, componentes, funcionalidades del negocio, reglas del negocio y entidades del negocio.

3. **Capa de Datos:**
    - **Lógica de acceso a datos (Patrón-Facade):** Recoge las clases y funcionalidades de acceso a los datos ya sea la información almacena en ficheros o en bases de datos. Sirve como intermediario o mediado entre la lógica de datos y la lógica de negocio. En este nivel se usará como interfaz principal el patrón fachada.
    - **Datos:** Recoge las especificaciones del almacenamiento de los datos, sea en ficheros o base de datos.

Módulos Básicos:

- **Configuración:** Conjunto de clases encargada de realizar todo lo concerniente a la configuración del área de trabajo, idioma de la aplicación, conexiones a base de datos y módulos.
- **Tratamiento de Excepciones:** Conjunto de clases encargada de llevar el control y el tratamiento de las excepciones que ocurren en tiempo de ejecución.
- **Internacionalización:** Encargado de manipular y mantener la configuración de idioma de la plataforma (Español, Inglés, otros idiomas).
- **Seguridad:** Encargado de llevar el control y seguimiento de los aspectos de seguridad que soporte el sistema sobre los elementos de configuración y los datos.

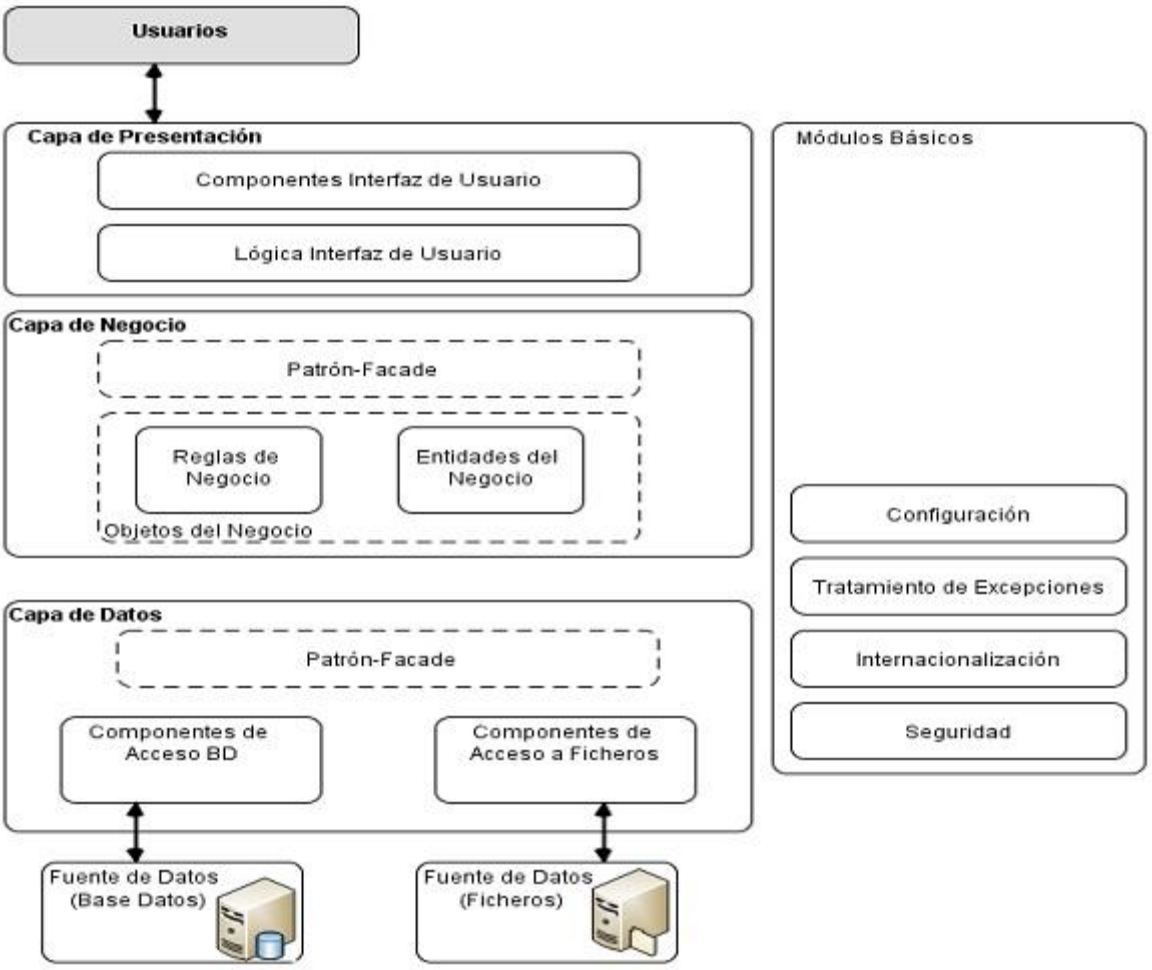

Figura 7: Organización de las capas y sus elementos más significativos.

Descomposición en Módulos.

Para descomponer los módulos del sistema, después de elegir la organización general del sistema, es necesario decidir cómo dividir el sistema en módulos.

Se utilizará la siguiente estrategia:

- Descomposición orientada a objetos:
  - ✓ El sistema se estructura en un conjunto de objetos débilmente acoplados con interfaces bien definidas.

- ✓ Cuando se implementan, se utiliza algún modelo de control para coordinar las operaciones de los objetos.

Estructura del Sistema en Módulos.

Cada módulo va a estar constituido por un conjunto de librerías dinámicas con sus interfaces bien definidas, desarrolladas por terceras empresas (Librerías QT-C++, VTK 3D y 2D, drivers de Base de datos) o por el equipo de desarrollo del proyecto Minería.

- Framework QT-C++: Contiene las librerías de QT.
- Visualización: Se sustenta en las librerías VTK 3D y 2D para realizar la graficación de los elementos y objetos.
- Geoestadística, Estimación, Diseño de Mina, Leyendas – Colores y Topografía.
- CORE: Tratamiento de excepciones, i18n (Internacionalización del idioma), Configuración y Seguridad.
- ORM-DataBase (Mapeo de Objeto Relacional-Base de Datos): Encargado del trabajo dinámico del acceso a los datos sobre servidores de bases de datos.
- FOM-File: Encargado de la manipulación de información a través de ficheros.

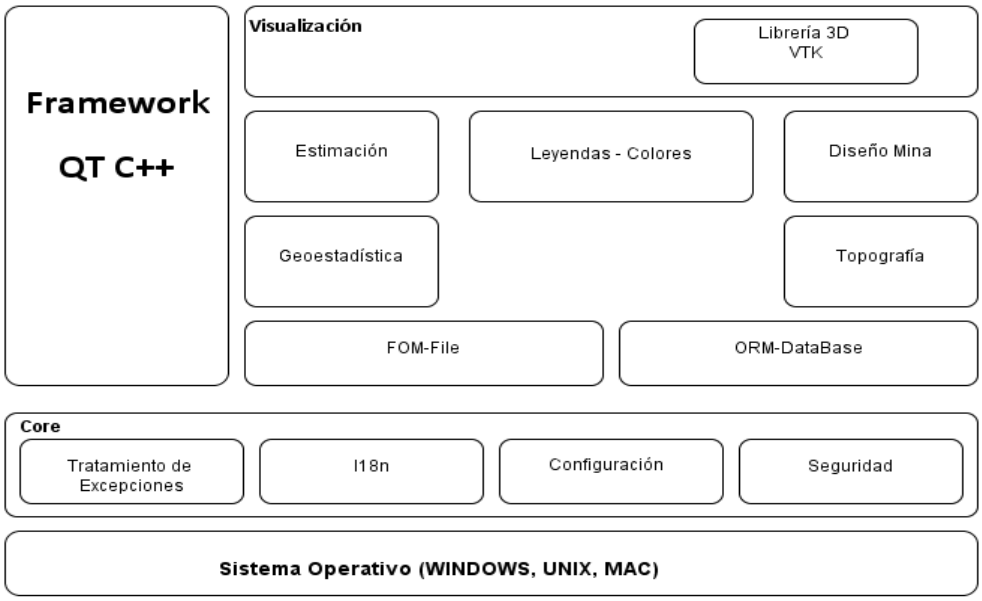

Figura 8: Organización de los módulos.

2.4.3 VISTA DE PROCESOS

La Vista de Procesos 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. Se centra por tanto en la concurrencia y distribución de procesos.

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

2.4.4 VISTA DE DESPLIEGUE

La vista de despliegue, muestra la distribución física entre los componentes hardware y software del sistema (ordenadores, dispositivos) y sus conexiones. Es un conjunto de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes software, objetos y procesos.

Teniendo en cuenta la infraestructura tecnológica del Proyecto Minería, el diagrama de despliegue de la aplicación dará una medida de la tecnología necesaria para el correcto funcionamiento del sistema en

desarrollo. Además propone la distribución física de los elementos que lo conforman, pues representa cómo estarán distribuidos y cómo se satisfacen los requisitos no funcionales.

La siguiente figura muestra como está diseñado el diagrama:

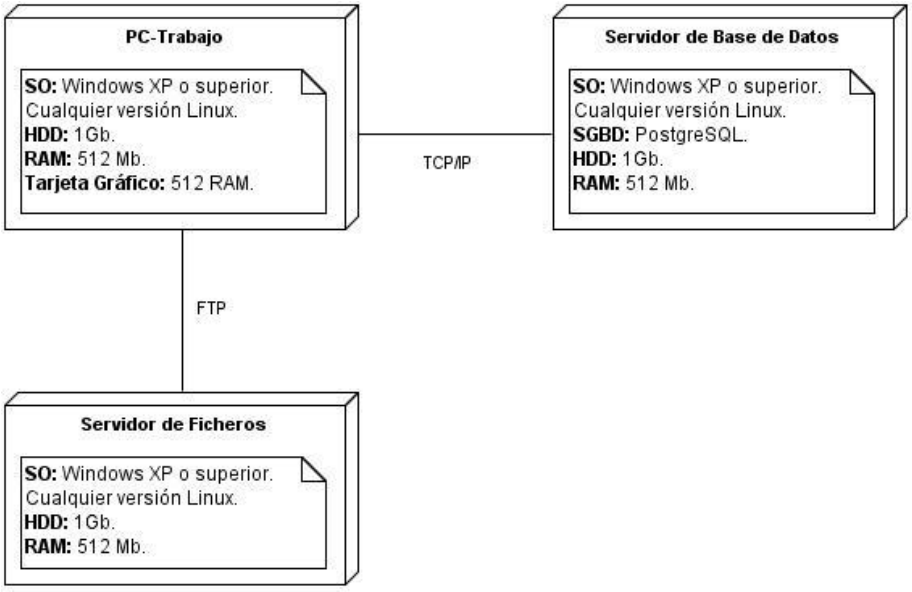

Figura 9: Diagrama de despliegue.

A continuación se describirán los elementos (nodos), y el protocolo de comunicación:

Nodos:

- **PC-Trabajo:** A través de este nodo será la interacción de los usuarios con el sistema. Existirán tantos nodos como este según entienda el cliente.
- **Servidor de Base de Datos:** Este nodo contendrá la base de datos del sistema, así como toda la información que se maneja en el mismo con el SGBD PostgreSQL.
- **Servidor de Ficheros:** Este nodo contendrá la información de los pozos de sondeo que se maneja en el sistema.

Protocolo de comunicación:

- TCP/IP: La familia de protocolos de internet TCP/IP, hace referencia a los dos protocolos más importantes que la componen. El protocolo de control de transmisión (TCP) garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron. EL protocolo de Internet (IP) es no orientado a conexión usado tanto por el origen como por el destino para la comunicación de datos a través de una red. (Babylon, 2008)

2.4.5 VISTA DE IMPLEMENTACIÓN

La vista de Implementación, muestra las organizaciones y dependencias lógicas entre componentes software, sean estos componentes de código fuente, binarios o ejecutables. En este diagrama se tiene en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización, las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo.

Con la representación de esta vista se ayuda a los desarrolladores a visualizar el camino de la implementación y permite tomar decisiones respecto a las tareas del desarrollo. Los paquetes definidos para cada uno de los subsistemas y módulos de la aplicación son:

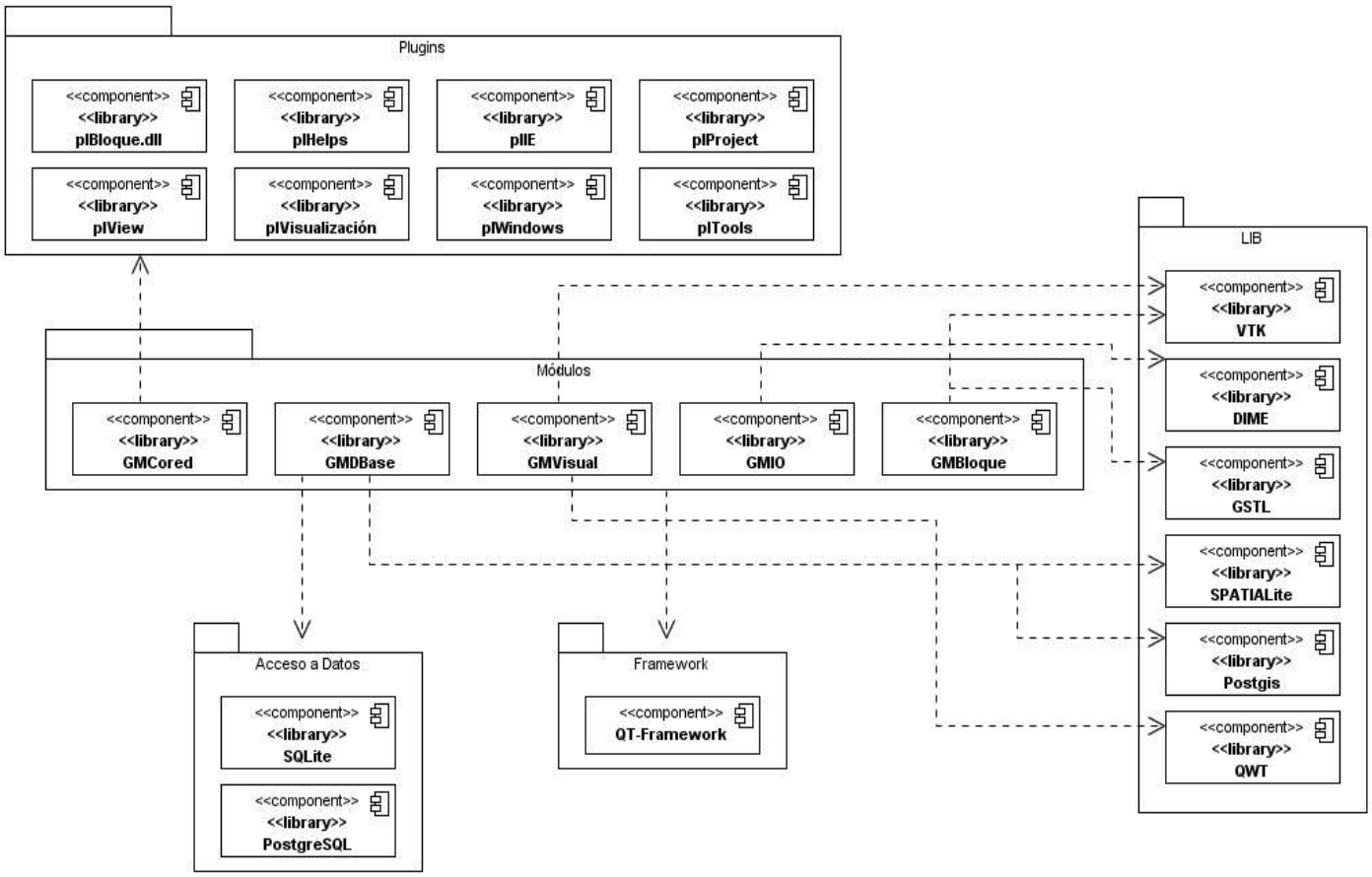

Figura 10: Diagrama de Implementación.

A continuación se describe cada uno de los paquetes representados en el diagrama de implementación:

- Plugins: Estos plugins se utilizan específicamente para agregarle diferentes funcionalidades a la aplicación. Son desarrollados por implementadores del proyecto.
- Módulos: Para la implementación, el sistema deberá dividirse en cada uno de los módulos representados.
- LIB: Librerías básicas que se irán utilizando de acuerdo a las necesidades de cada uno de los módulos. Estas librerías sirven de apoyo o complemento a los diferentes módulos.
- Acceso a Datos: Este paquete contendrá las librerías de los SGBD SQLite y PostgreSQL.

- Framework: Paquete de clases QT a utilizar para el desarrollo del sistema.

2.5 CONCLUSIONES PARCIALES

En este capítulo quedó finalmente establecida una arquitectura candidata para el sistema GeolMin. Esta propuesta se describe mediante las vistas de la arquitectura. Se establecieron los requisitos no funcionales que debe cumplir el sistema, basados en las diferentes tecnologías, herramientas y licencias libres con las cuales se justifica un desarrollo ágil y de calidad.

Se contemplan además un conjunto de elementos que ayudan a definir la funcionalidad del sistema, como es la vista de casos de uso, que representa los más significativos desde el punto de vista arquitectónico. En la vista lógica se realizó una estructura de paquetes teniendo en cuenta el estilo arquitectónico y patrones utilizados; lo que permite mostrar cómo la funcionalidad es diseñada en el interior del sistema, en términos de la estructura estática y comportamiento dinámico del sistema. La vista de despliegue muestra la infraestructura de tecnologías que se necesita para el correcto funcionamiento del sistema GeolMin.

CAPÍTULO #3

VALIDACIÓN DE LA PROPUESTA DE ARQUITECTURA

El presente capítulo está enfocado a la evaluación de la arquitectura propuesta descrita en el capítulo 2. Analizando los resultados obtenidos de acuerdo con las técnicas y métodos de evaluación de arquitecturas de software, se busca que la propuesta sea funcional y cumpla con los atributos de calidad requeridos.

El objetivo de evaluar una arquitectura es saber si puede habilitar los requerimientos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los stakeholders. (Procedimiento para la Evaluación de Arquitecturas de Software basadas en Componentes, 2009)

El propósito de realizar evaluaciones a la arquitectura, es para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante, verificar que los requerimientos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad. (Procedimiento para la Evaluación de Arquitecturas de Software basadas en Componentes, 2009)

3.1 CUALIDADES PARA EVALUAR UNA ARQUITECTURA

Para evaluar una arquitectura es necesario saber que cualidades de esta se desea evaluar realmente. A estas cualidades también se les llama atributos de calidad.

Modelo ISO/IEC 9126 (adaptado para arquitecturas de software).

El estándar ISO/IEC 9126 ha sido desarrollado en un intento de identificar los atributos clave de calidad para un producto de software (Pressman, 2002). Referente a este estándar Losavio et al. (2003) propone una adaptación de este modelo para efectos de la evaluación de arquitecturas de software. Este se basa en los atributos de calidad que se relacionan directamente con la arquitectura: funcionalidad, confiabilidad, eficiencia, mantenibilidad y portabilidad, los cuales se representan en la siguiente tabla:

Validación de la propuesta de arquitectura

Característica	Subcaracterística	Elementos de tipo arquitectónico
Funcionalidad	Adecuación	Refinamiento de los diagramas de secuencia.
	Exactitud	Identificación de los componentes con las funciones responsables de los cálculos.
	Interoperabilidad	Identificación de conectores de comunicación con sistemas externos.
	Seguridad	Mecanismos o dispositivos que realizan explícitamente la tarea.
Confiabilidad	Tolerancia a fallas	Existencia de mecanismos o dispositivos de software para manejar excepciones.
	Recuperabilidad	Existencia de mecanismos o dispositivos de software para reestablecer el nivel de desempeño y recuperar datos.
Eficiencia	Desempeño	Componentes involucrados en un flujo de ejecución para una funcionalidad.
	Utilización de recursos	Relación de los componentes en términos de espacio y tiempo.
Mantenibilidad	Acoplamiento	Interacciones entre componentes.
	Modularidad	Número de componentes que dependen de un componente.

Portabilidad	Adaptabilidad	Presencia de mecanismos de adaptación.
	Instalabilidad	Presencia de mecanismos de instalación.
	Coexistencia	Presencia de mecanismos que faciliten la coexistencia.
	Reemplazabilidad	Lista de componentes reemplazables para cada componente.

Tabla 1: Atributos de calidad planteados por Losavio et al. (2003).

Después de analizar cada atributo y sub-atributo, se puede identificar en base a los requisitos no funcionales, cuáles de estos atributos deberán estar presentes en el árbol de utilidad, su prioridad y los escenarios donde se manifiestan.

Luego de conocer los atributos de calidad a medir en la arquitectura, todavía no se está en condiciones de medir la calidad de una arquitectura si no se utiliza alguna técnica existente para estas evaluaciones.

3.2 TÉCNICAS DE EVALUACIÓN

Existen un grupo de técnicas para evaluar la arquitectura que se clasifican en cualitativas y cuantitativas, como lo muestra la siguiente figura. (Procedimiento para la Evaluación de Arquitecturas de Software basadas en Componentes, 2009)

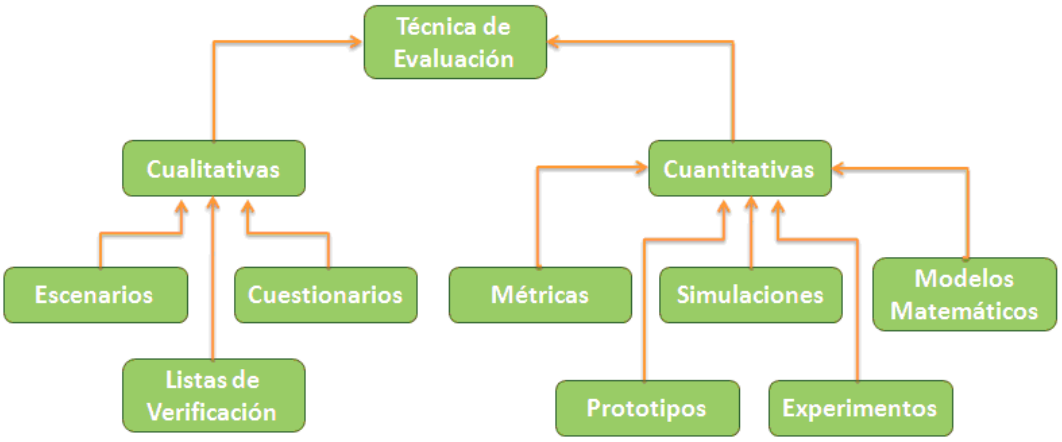

Figura 11: Técnicas de Evaluación.

Generalmente las técnicas de evaluación cualitativas son utilizadas cuando la arquitectura está en construcción, mientras que las técnicas de evaluación cuantitativas, se usan cuando la arquitectura ya ha sido implantada. Estos tipos de técnicas posibilitan evaluar una arquitectura y establecer comparaciones entre arquitecturas candidatas para determinar cuál satisface más un atributo de calidad específico.

De las técnicas de validación mencionadas, se utilizará la evaluación cualitativa, específicamente la evaluación basada en escenarios. Debido a que el sistema se encuentra en desarrollo y la arquitectura propuesta está en construcción. A continuación se describe en qué consiste la evaluación basada en escenarios y los elementos relacionados:

Evaluación basada en escenarios:

Un escenario consta de tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. El contexto describe qué sucede en el sistema al momento del estímulo. La respuesta describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Dentro de la evaluación por escenarios se utiliza lo que se conoce como el árbol de utilidad (*Utility Tree, en inglés).* (Kazmar, 2001)

Árbol de utilidad:

El Árbol de utilidad es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle, el

nivel de prioridad de cada uno. Identifica los atributos de calidad más importantes para un proyecto particular. El Árbol de utilidad contiene como nodo raíz la utilidad general del sistema. Los atributos de calidad asociados al mismo conforman el segundo nivel del árbol los cuales se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle prioridad a cada atributo considerado. (Camacho, 2004)

Al definir la técnica de evaluación y los atributos de calidad por los que será evaluada la arquitectura, solo queda definir porque método se realizará dicha evaluación.

3.3 MÉTODOS DE EVALUACIÓN DE ARQUITECTURAS

Los métodos de evaluación de arquitecturas permiten identificar los conflictos entre las arquitecturas y las soluciones que se desarrollan. Existen varios métodos que ayudan a esta identificación, entre estos se encuentran *Architecture Trade-off Analysis Method* (ATAM), *Software Architecture Analysis Method* (SAAM) y *Active Review for Intermediate Designs* (ARID).

3.3.1 SAAM.

El método de análisis de arquitecturas de software (SAAM[7], por sus siglas en inglés) es uno de los primeros que fue ampliamente promulgado y documentado. Originalmente fue creado para el análisis de modificabilidad de la arquitectura, pero ha demostrado ser muy útil para evaluar ciertos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad. (Kazmar, y otros, 2001)
La evaluación de este método está compuesta por seis (6) pasos, los cuales son:

1. Desarrollo de escenarios.
2. Descripción de la arquitectura.
3. Clasificación y asignación de la prioridad de los escenarios.
4. Evaluación individual de los escenarios indirectos.
5. Evaluación de la interacción entre los escenarios.
6. Creación de la evaluación global.

[7] Método de análisis de arquitectura de software.

Dependiendo del objetivo de la evaluación será el resultado de la misma. Por ejemplo, si el objetivo es evaluar una sola arquitectura, este método enumera los lugares más vulnerables a fallos dentro de la misma, en términos de los requerimientos de modificabilidad. Para el caso de que se deseen evaluar varias arquitecturas con el fin de seleccionar una que satisfaga mejor los requerimientos de calidad y con la menor cantidad de modificaciones posibles.

3.3.2 ARID.

El método de Análisis de Diseños Intermedios *(Active Reviews for Intermediate Designs, en inglés)* es conveniente para realizar revisiones parciales en etapas tempranas del desarrollo. Es conveniente saber si el diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura. ARID es un híbrido entre Active Design Review (ADR) y ATAM. ADR es utilizado para la evaluación de diseños detallados de unidades de software como los componentes o módulos.

En el caso de ADR proporciona una documentación detallada del diseño y completan cuestionarios. En el caso de ATAM se realiza una evaluación de la arquitectura en su conjunto. Por esta combinación de filosofías surge ARID para la evaluación temprana de arquitecturas de software. El método de evaluación comprende nueve pasos divididos en dos fases.

Fase 1: Actividades previas.

1. Identificación de los encargos de la revisión.
2. Preparar el informe de diseño.
3. Preparar los escenarios bases.
4. Preparar los materiales.

Fase 2: Revisión.

5. Presentación del ARID.
6. Presentación del diseño.
7. Lluvia de ideas y establecimiento de prioridad de escenarios.
8. Aplicación de los escenarios.
9. Resumen.

Por las características de los métodos de evaluación de arquitectura explicados anteriormente, se decide utilizar ATAM. Debido a que permite realizar la evaluación temprana de los diseños de la arquitectura. Teniendo en cuenta las condiciones y atributos de calidad del sistema GeolMin. Se considera que la evaluación de la arquitectura propuesta mediante este método sería más eficiente que otros.

3.3.3 ATAM.

EL método de identificación de riesgos para la arquitectura (ATAM[8], por sus siglas en inglés) está centrado en tres aéreas distintas, el estilo arquitectónico, el análisis de los atributos de calidad y el método SAAM. Su nombre surge del hecho de que revela la forma específica en que una arquitectura cumple con algunos atributos de calidad, así como la interacción de estos con otros. (Kazmar, y otros, 2001)

La metodología de ATAM comprende nueve pasos divididos en cuatro fases, los cuales se describirán a continuación. (Clements, 2007)

Fase 1: Presentación.	
Paso 1. Presentación del ATAM.	El equipo de evaluación presenta un panorama general de los pasos de ATAM y trata de establecer las expectativas de los resultados del proceso.
Paso 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. Se presenta brevemente el negocio y el contexto de la arquitectura.
Paso 3. Presentación de la arquitectura.	El arquitecto presenta un panorama de la arquitectura, describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio.

[8] Architecture Tradeoff Analysis en inglés. Método de identificación de riesgos para la arquitectura.

Fase 2: Investigación y Análisis.	
Paso 4. Identificación de los enfoques arquitectónicos.	El equipo de evaluación y el arquitecto deben detallar los planteamientos arquitectónicos descubiertos en el paso anterior. Estos elementos son detectados, pero no analizados.
Paso 5. Generación del Utility Tree.	Se identifican, priorizan, definen y refinan los atributos de calidad más importantes en un formato de árbol de utilidad y que engloban la "utilidad" del sistema, especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.
Paso 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. Se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance. Se utilizan las preguntas similares a las presentadas en el paso 5.
Fase 3: Pruebas.	
Paso 7. Lluvia de ideas y establecimiento de prioridad de escenarios.	Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios y se le da prioridad a los escenarios sobre la base de su importancia relativa.
Paso 8. Análisis de los enfoques	Este paso repite las actividades del paso 6, haciendo

arquitectónicos.	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.	
Paso 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. El equipo de evaluación recapitula los pasos de ATAM, resultados, y recomendaciones.

Tabla 2: Pasos del Método de Evaluación ATAM.

Al finalizar el desarrollo del método se obtienen los siguientes resultados:

- El árbol de utilidad.
- El conjunto de escenarios priorizados.
- Los riesgos descubiertos.
- Los no riesgos documentados.
- Los puntos de sensibilidad encontrados.
- El conjunto de preguntas basadas en los atributos.

3.4 EVALUACIÓN DE LA ARQUITECTURA

Al analizar los métodos descritos anteriormente se escoge el método ATAM para evaluar la arquitectura propuesta. Esta selección se basa en la posibilidad de identificar riesgos, puntos de sensibilidad, y puntos de equilibrio en el análisis de la arquitectura. Lo que permite conocer si la propuesta de arquitectura da cumplimiento a los objetivos trazados.

De las 4 Fases y 9 Pasos que define el ATAM, se presenta continuación el árbol de utilidad, con los atributos de calidad en base a los requisitos no funcionales, definidos como significativos para el sistema en general. Se realiza además el análisis de los escenarios de mayor prioridad arquitectónica. Sólo se exponen estos dos pasos por ser identificados como los más importantes en el proceso de evaluación.

Generación del Árbol de Utilidad: Se obtienen los atributos de calidad que engloban la "utilidad" del sistema especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.

El árbol de utilidad generado se toma como un plan para el resto de la evaluación que realiza el método. Indica además al equipo evaluador dónde ocupar su tiempo y dónde probar aproximaciones y riesgos arquitectónicos.

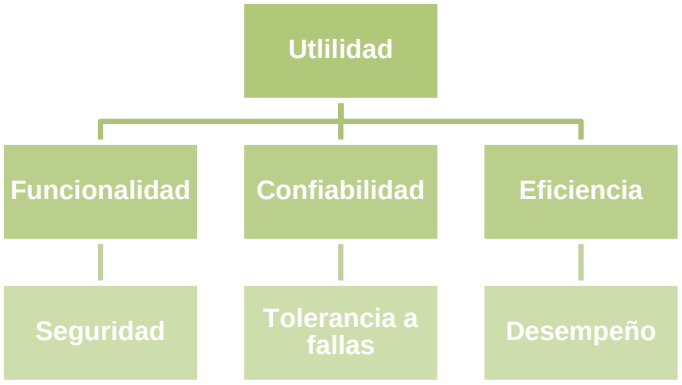

Figura 12: Árbol de Utilidad.

Luego de conformar el árbol de utilidad, se muestra el análisis realizado a los escenarios más importantes y las propuestas arquitectónicas que cumplen con estos escenarios. Teniendo en cuenta el árbol de utilidades se les asigna prioridad a cada uno de ellos. Se describe para cada uno el atributo, estímulo, ambiente en el que se encuentra el sistema, la respuesta y una explicación de cómo responde la arquitectura al estímulo a través de la correspondiente decisión.

Los escenarios asociados al árbol de utilidad son:

Escenario # 1	Escenario: La información manejada por el sistema deberá estar protegida de acceso no autorizado y divulgación.
Atributo(s)	**Funcionalidad-Seguridad**
Ambiente	Operación Normal
Estímulo	Usuario que no cumple con los requisitos de

	acceso intenta acceder a la información del sistema guardada.
Respuesta	Acceso denegado si no tiene la clave correcta.
Decisiones arquitectónicas	En la clase de control de la seguridad se implementa un método Seguridad que establece que el usuario no puede acceder a la información guardada en el sistema sin la clave correcta.

Tabla 3: Escenario #1.

Escenario # 2	Escenario: La probabilidad de que el sistema funcione o desarrolle las funciones de trabajo con los datos geológicos en el terreno debe ser alta.
Atributo(s)	**Confiabilidad-Tolerancia a fallas**
Ambiente	Operación Normal
Estímulo	Error al activar la aplicación o realizar alguna funcionalidad.
Respuesta	No se puede trabajar con la aplicación.
Decisiones arquitectónicas	Al no poder trabajar con la aplicación luego de un fallo ocurrido, este será corregido por el equipo de desarrollo que designe el líder del proyecto en un período menor a un mes.

Tabla 4: Escenario #2.

Escenario # 3	Escenario: La velocidad de procesamiento de la información, la actualización y la recuperación dependerán de la cantidad de información que tenga que procesar la aplicación.
Atributo(s)	**Eficiencia-Desempeño**
Ambiente	Operación Normal

Estímulo	Se procesa gran cantidad de datos geológicos en forma de imágenes.
Respuesta	La aplicación se demora en procesar la información.
Decisiones arquitectónicas	Se debe revisar el cumplimiento de los requisitos de hardware que ayuda al procesado de imágenes.

Tabla 5: Escenario #3.

Basado en la información recolectada y haciendo uso del método de evaluación ATAM se resume y presenta; las salidas más importantes que son:

- El árbol de utilidad.
- El conjunto de escenarios priorizados.
- Los riesgos descubiertos. *(ver Anexo 1)*.
- Los no riesgos descubiertos. *(ver Anexo 2)*.
- Los puntos de sensibilidad encontrados. *(ver Anexo 3)*.

Se propone realizar una posterior evaluación a la arquitectura de forma detallada, siguiendo todas las fases y pasos definidos por el método. Debido a que la evaluación realizada hasta el momento se efectuó con los atributos de calidad, en base a los requisitos no funcionales identificados como significativos para el correcto funcionamiento del sistema. Por lo que se logra la meta del equipo de evaluación de estar convencidos que la propuesta instanciada en la arquitectura que se está evaluando es la apropiada para satisfacer los requerimientos de un atributo específico.

3.5 CONCLUSIONES PARCIALES

Al realizarse la propuesta de la evaluación de la arquitectura por el método ATAM, como parte del proceso de evaluación temprana, se aplicó la técnica basada en escenarios y el instrumento árbol de utilidad. Se identificaron 3 escenarios, los cuales arrojaron un riesgo, dos no riesgos y dos puntos sensibles, demostrándose que la propuesta de arquitectura da cumplimiento al objetivo trazado.

CONCLUSIONES GENERALES

El desarrollo de la arquitectura de software es una de las etapas fundamentales en el desarrollo de software. Aquí es donde los profesionales aportan todos sus conocimientos, creatividad y experiencia para crear la mejor propuesta de solución que se dará al cliente, que cumpla con los requisitos funcionales y no funcionales establecidos para el sistema. Es de vital importancia que todo sistema de software esté respaldado con una arquitectura sólida, que facilite el entendimiento del mismo, que sea capaz de organizar el desarrollo, fomentar la reutilización y hacer evolucionar el sistema. Luego de culminada la realización del presente trabajo y analizado todo el contenido expuesto anteriormente, se arriban a las siguientes conclusiones:

- Mediante la descripción de la arquitectura se documenta la línea base y se representan los principios que guían la evolución de esta para el desarrollo del producto con un enfoque basado en tecnologías libres.
- Con la aplicación correcta de los patrones y estilos arquitectónicos a los elementos estructurales del sistema se obtiene una arquitectura robusta para el sistema GeolMin.
- La evaluación de la arquitectura mediante el método ATAM, demuestra cómo se potencian los atributos de calidad propuestos como objetivos.

RECOMENDACIONES

De forma general, el diseño arquitectónico propuesto describe toda una serie de aspectos significativos referentes al desarrollo del sistema minero GeolMin. No obstante, como parte de un proceso de mejoras continuas se proponen las siguientes recomendaciones:

- Debido a que el sistema GeolMin se encuentra en desarrollo, se recomienda mantener la propuesta de arquitectura en constante refinamiento con el propósito de entregar la documentación asociada a la arquitectura completa y en correspondencia con las nuevas funcionalidades que se le agreguen al sistema.
- Realizar una evaluación detallada a la arquitectura propuesta, considerando todos los pasos que define el método de evaluación seleccionado ATAM.

BIBLIOGRAFÍA CITADA

Babylon. 2008. Babylon. [En línea] 2008. [Citado el: 28 de Mayo de 2011.] http://www.babylon.com/definition/TCP_IP/Spanish.

Bosh, J. 2000. *Design & Use of software architecture.* Addison Wesley : s.n., 2000.

Burbeck, Steve. 2006. *Application programming in Smalltalk-80: How to use Model-View-Controller (MVC).* 2006.

Camacho, E., Cordeso, F. y Nuñez, G. 2004. *Arquitectura de Software.* 2004.

Clements. 1996. *A field guide to Boxology: Preliminary classification. .* 1996.

Clements, Paul, y otros. 2007. *ATAM: Method for Architecture Evaluation.* Pittsburgh : s.n., 2007.

Garlan, David. 1994. *An Introdduction to Software Architecture.* 1994.

IBM. 2003. *Ayuda del Rational Rose.* 2003.

IGAC. 2009. 2009.

ISO. 2007. *Standarization International Organization .* 2007.

Kazmar, R., Clement, P. y Cleint, M. 2001. *Evaluating Software architectrues. Methods and cases studies.* Addison Wesley : s.n., 2001.

Kruchten, Philippe B. 1995. *Architectural blueprints: The 4+1 view model of architecture.* s.l. : IEEE Software, 1995.

Larman, Craig. 2004. *UML y Patrones.* La Habana : Felix Varela, 2004.

Leyet. 2007. *Minerales.* 2007.

Manual de usuario PostgreSQL.

Perera, Jose Raúl. 2007. *ARQUITECTURA DE SOFTWARE PARA SISTEMA GESTION DE INVENTARIOS.* Ciudad Habana : s.n., 2007.

PostgreSQL, Manual. *Manual de usuario PostgreSQL.*

Pressman, R. S. 2001. *Ingeniería de Software "Un enfoque práctico".* 2001.

Procedimiento para la Evaluación de Arquitecturas de Software basadas en Componentes. **Carrascoso Puebla, Yoan Arlet, Chaviano Gómez, Enrique y Céspedes Vega, Anisleydi. 2009.** La Habana : s.n., 2009. 08.

Reynoso, Billy. 2004. *Introducción a la Arquitectura de Software.* 2004.

Rondón, Yoandri Quintana. 2009. *Quintana. Desarrollo de la arquitectura del proyecto Captura .* Habana : s.n., 2009.

Sánchez Mendoza, María A. 2004. *Metodologías de desarrollo de software.* Perú : s.n., 2004.

BIBLIOGRAFÍA CONSULTADA

Cueva Lovelle, Juan Manuel. Tecnología de Objetos: Patrones de Diseño. 2004.

Pressman. 2001. Ingeniería del software Un enfoque practico. s.l. : Mcgraw-hill , 2001. 8448132149.

Clements, Paul. A Survey of Architecture Description Languages. . Alemania : s.n., 1996.

IEEE Std, 1471-2000. Recommended Practice for Architectural Description of Software-Intensive Systems. 2000.

Babylon. 2008. Babylon. http://www.babylon.com/definition/TCP_IP/Spanish.

Bosh, J. 2000. Design & Use of software architecture. Addison Wesley : s.n., 2000.

Burbeck, Steve. 2006. Application programming in Smalltalk-80: How to use Model-View-Controller (MVC). 2006.

Camacho, E., Cordeso, F. y Nuñez, G. 2004. Arquitectura de Software. 2004.

Clements. 1996. A field guide to Boxology: Preliminary classification. . 1996.

Clements, Paul, y otros. 2007. ATAM: Method for Architecture Evaluation. Pittsburgh : s.n., 2007.

Garlan, David. 1994. An Introdduction to Software Architecture. 1994.

IBM. 2003. Ayuda del Rational Rose. 2003.

IGAC. 2009. 2009.

ISO. 2007. Standarization International Organization . 2007.

Kazmar, R., Clement, P. y Cleint, M. 2001. Evaluating Software architectrues. Methods and cases studies. Addison Wesley : s.n., 2001.

Kruchten, Philippe B. 1995. Architectural blueprints: The 4+1 view model of architecture. s.l. : IEEE Software, 1995.

Larman, Craig. 2004. UML y Patrones. La Habana : Felix Varela, 2004.

Leyet. 2007. Minerales. 2007.

Perera, Jose Raúl. 2007. ARQUITECTURA DE SOFTWARE PARA SISTEMA GESTION DE INVENTARIOS. Ciudad Habana : s.n., 2007.

Pressman, R. S. 2001. Ingeniería de Software "Un enfoque práctico". 2001.

Procedimiento para la Evaluación de Arquitecturas de Software basadas en Componentes. **Carrascoso Puebla, Yoan Arlet, Chaviano Gómez, Enrique y Céspedes Vega, Anisleydi. 2009.** La Habana : s.n., 2009. 08.

Reynoso, Billy. 2004. Introducción a la Arquitectura de Software. 2004.

Rondón, Yoandri Quintana. 2009. Quintana. Desarrollo de la arquitectura del proyecto Captura . Habana : s.n., 2009.

Sánchez Mendoza, María A. 2004. Metodologías de desarrollo de software. Perú : s.n., 2004.

ANEXOS

Anexo 1. Salidas del método de evaluación ATAM: Riesgos.

R1: Si no se cumplen los requisitos no funcionales de hardware, el sistema puede presentar demora en realizar el procesado de la información.

Anexo 2. Salidas del método de evaluación ATAM: No Riesgos.

NR1: Definir una clase o método Seguridad, asegura que la información esté protegida del acceso no autorizado.

NR2: Si el sistema presentara algún desperfecto, el equipo de mantenimiento estará encargado de corregir el error en un período menor a un mes.

Anexo 3. Salidas del método de evaluación ATAM: Puntos de sensibilidad.

P1: Desarrollar una clase o método Seguridad, representa un punto de sensibilidad para lograr el aseguramiento de la información.

P2: Normalizar la Base de Datos constituye un punto de sensibilidad para lograr que el tiempo de respuesta y procesado de la información a las peticiones realizadas por los usuarios sean mínimos.

GLOSARIO DE TÉRMINOS

Software: Son los programas usados para dirigir las funciones del sistema de computación.

Arquitectura de Software: Es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación.

Base de Datos: Es una colección de información organizada de tal forma que un programa de ordenador pueda seleccionar.

Software Libre: Llamado también software de dominio público, es el que se ofrece sin costo alguno.

UML: Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad.

Caso de Uso: Es una descripción de un conjunto de secuencias de acciones, incluyendo variaciones, que un sistema lleva a cabo y que conducen a un resultado observable de interés para un actor determinado.

Framework: Esquema (un esqueleto) para el desarrollo y/o la implementación de una aplicación.

Herramienta: Software que se utiliza para automatizar las actividades definidas en los procesos.

Herramientas CASE: Conjunto de aplicaciones informáticas orientadas al incremento de la productividad en el desarrollo de software, las siglas CASE vienen dadas por su nombre en inglés *Computer Arded Software Engineering* que se conoce como Ingeniería de Software Asistida por Computadoras.

Stakeholders: Personas u organizaciones que están activamente implicadas en el negocio, ya sea porque participan en él o porque sus intereses se ven afectados con los resultados del proyecto.

Programación Orientada a Objetos (POO): Es un paradigma de programación que define los programas en términos de clases de objetos, que son entidades que combinan *estado* (datos), *comportamiento* (procedimientos o métodos) e *identidad* (propiedad del objeto que lo diferencia del resto). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Estos métodos están pensados para hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.