UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
Facultad 6

TÍTULO: Desarrollo de paquete de Arquitecturas de Software para Sistemas de Información Geográfica de escritorio.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN INFORMÁTICA

AUTOR: Lewis Rodríguez Fuentes

TUTOR: Msc. David Silva Barrera

La Habana, 27 de Junio de 2011
"Año 53 de la Revolución"

La imaginación es más importante que la sabiduría

Albert Einstein

DEDICATORIA

Dedico especialmente este trabajo:

A Lissy: mi razón de ser, la imagen que en mis ojos arde.

A Emilia y Luis: las luces que guían mis pasos.

A Aimara: la estrella que ilumina mis noches.

AGRADECIMIENTOS

Fueron muchos los que siempre tuvieron sus manos extendidas a cualquier evento que invadiera mi vida. Fueron muchas voces las que siempre estuvieron presentes para los momentos de angustia o felicidad, para dar su apoyo o crítica constructiva. Y fue mucho el tiempo que dedicaron todas y cada una de esas personas para lograr llevarme hacia donde he llegado, para lograr hacer de mí quien hoy día soy.

Agradezco:

A nuestro comandante en jefe Fidel Castro Ruz, a Raúl Castro Ruz y a la gloriosa Revolución cubana.

A mi hermana que es mi razón de ser, la otra parte de mi yo interior y quien hace que me levante con más fuerzas cada día.

A mis padres, incansables guías de mis pasos en esta excursión por la vida y quienes les debo hoy día toda mi felicidad.

A mi amor, quien cuida en muchas noches mis sueños y en cada uno de ellos deposita un poquito de amor y esperanza. De igual manera a toda su familia, que ya es mía también, por haber sido en todo momento incondicional. Gracias por sus desvelos, ayuda y amor.

A mis hermanos de corazón y verdadera lealtad, quienes en todo momento me apoyaron y entregaron su confianza y cariño.

A mis abuelas y abuelos por su ejemplo de sacrificio y sabiduría. Por todas sus paciencias y enseñanzas de amor.

A toda mi familia pues de todos tengo en mí un pedacito de su sabiduría, amor y enseñanza.

A mi tutor quien me ayudó a formarme como profesional.

A los profesores y compañeros, que me dedicaron un pedacito de su tiempo y con ello una pisca de enseñanza y paciencia.

A todos aquellos que me han apoyado durante este largo andar…,

Muchísimas gracias.

DECLARACIÓN DE AUTORÍA

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

Para que así conste firmo la presente a los 27 días del mes de Junio del año 2011.

Lewis Rodríguez Fuentes	David Silva Barrera
Autor	Tutor

DATOS DE CONTACTO

Tutor: Msc. David Silva Barrera

Especialidad de graduación: Licenciado en Ciencias de la Computación

Categoría docente: Profesor Asistente

Categoría Científica: Máster en Ciencias

Años de graduado: 8

Correo electrónico: dsilva@uci.cu

Universidad de las Ciencias Informáticas, La Habana, Cuba

RESUMEN

En el presente trabajo se propuso el desarrollo de un paquete de arquitecturas de software según el nivel de importancia de las funcionalidades de representación geográfica, basado principalmente en la solución existente Quantum GIS. Para darle cumplimiento se definió los estilos y patrones arquitectónicos, así como el lenguaje de descripción de la arquitectura. Se describió algunas metodologías de desarrollo, lenguajes de programación, herramientas CASE y entornos de desarrollo integrados con el propósito de seleccionar el adecuado para el diseño del paquete arquitectónico a plantear. Se presentaron un conjunto de elementos que ayudaron a definir la funcionalidad del sistema, como es la descripción de atributos de calidad y los requisitos que tienen impacto sobre la arquitectura así como la herramienta de integración a utilizar para la comunicación con otros sistemas. Se muestran los casos de uso arquitectónicamente significativos, los cuales serán la base para el desarrollo de las demás vistas. En la vista lógica se realizó una estructura de paquetes, teniendo en cuenta el estilo arquitectónico y patrones utilizados. Las vistas: implementación y despliegue muestran la infraestructura de tecnologías que se necesita para el correcto funcionamiento del SIG, haciendo énfasis en sus principales componentes. Ambas vistas muestran la distribución física del sistema, en la primera desde el punto de vista de la implementación y la segunda según los nodos en los que se despliega. Finalmente se evaluó la arquitectura del software con el método ATAM, para obtener los posibles errores del diseño arquitectónico y el nivel de aceptación del paquete de arquitecturas.

PALABRAS CLAVES:

Arquitectura de Software (AS), GeoQ, paquete de Arquitectura de Software, Quantum GIS (QGIS), Sistema de Información Geográfica (SIG).

ABSTRACT

In this paper was purposed the development a software architectures' package based on the relevance's level of the geographical features, mainly based on the existing solution Quantum GIS. To comply so, it was defined architectural styles and patterns, as well as the architecture's description language. It described some development methodologies, programming languages, CASE tools and integrated development environments in order to select the appropriate package for architectural design. There were a number of factors that helped define the functionality of the system, such as the description of quality attributes and requirements that impact on the architecture and integration tool used to communicate with other systems. It shows the architecturally significant use cases, which will be the basis for the development of other views. The logical view has a package's structure, taking into account the architectural style and patterns used. Views: implementation and deployment show the infrastructure needed for the proper functioning of GIS, emphasizing its main components. Both views show the physical distribution system, the former from the point of view of implementation and the latter as the nodes on which it is deployed. In the evaluation of software architecture is used the ATAM method to obtain the architectural design's error and the level of acceptance packet architectures.

KEY WORDS:

Geographic Information System (GIS), GeoQ, Quantum GIS (QGIS), Software Architecture (AS), Software Architecture package.

FIGURAS Y TABLAS

Figuras

Tablas

ÍNDICE

INTRODUCCIÓN

La distribución espacial de objetos es inherente a los fenómenos, tanto los naturales como los artificiales, que puedan ocurrir sobre la corteza terrestre. Desde el surgimiento de las necesidades más remotas del hombre en cuanto al conocimiento geográfico, como conocer la distancia entre dos lugares, el tiempo de recorrido y la ubicación de los servicios sociales; este ha organizado de alguna forma la información espacial, ya sea marcando los lugares más importantes o realizando dibujos y trazos que le ayudarían a reconocer más tarde el territorio y poder ejercer decisiones sobre este según lo que había obtenido.

Así, el creciente desarrollo en las civilizaciones fue creando una mayor necesidad de georreferenciar[1] todo lo que les rodeaba para conocer el espacio que ocupaba en el terreno. Ello contribuyó al surgimiento de los mapas, lo cual unido a la evolución misma de la cartografía contribuyeron posteriormente a análisis más profundos de la conceptualización del entorno geográfico. Consecutivamente con la ayuda del desarrollo continuo de las Tecnologías de la Información y las Comunicaciones (TIC) para mejorar el proceso, surgen los Sistemas de Información Geográficos (SIG, GIS por sus siglas en inglés) como una solución mejor elaborada y con más funcionalidades.

Los SIG son sistemas de información que se utilizan para el análisis, manipulación, obtención o visualización de datos geográficos. Ellos permiten la representación, en un mapa digital, de una región con todos sus datos georreferenciados que brindarán información al usuario que los consulte. Los mismos cuentan con herramientas dotadas de grandes capacidades de procesamiento alfanumérico y gráfico, que poseen aplicaciones y procedimientos capaces de realizar la captura, el almacenamiento, el análisis y exposición de los datos representados.

En un proyecto de desarrollo de sistemas de información geográfica el peso de los elementos que lo componen ha ido cambiando, radicando su mayor importancia en los datos geográficos, pues estos se hacen cada vez más necesarios y son los que consumen actualmente la mayor parte de las inversiones en términos económicos y de tiempo. Numerosos han sido los sistemas de información geográfica desarrollados internacionalmente; GeoMedia, ArcView, ERDAS, GRASS, QGIS, son algunos de ellos, los cuales implementan variadas alternativas de arquitectura de software para su concepción.

[1] Incluir su posición en el espacio utilizando un sistema de coordenadas estandarizado resultado de una proyección cartográfica.

La arquitectura es la base y organización fundamental de todo proyecto de software y está principalmente centrada en los componentes de la aplicación, así como las relaciones entre ellos y los principios que rigen su diseño y posterior desarrollo. Una definición de la Arquitectura del Software (AS) la expresa la IEEE Std 1471-2000 que plantea: La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución. **(IEEE 2000)**

La Arquitectura de Software, a grandes rasgos, es una vista del sistema que incluye los componentes principales del mismo. La conducta de esos componentes se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. Además la vista arquitectónica es una vista abstracta, que aporta el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones, según Clements cuando expresa una de las definiciones más reconocidas. Y su encause principal es dotar de conceptos y un lenguaje común a los equipos participantes en un proyecto y para lograrlo, construir abstracciones materializándolas en forma de diagramas. Además de aportar elementos que ayuden a la toma de decisiones durante el desarrollo del software.

El departamento de Geoinformática[2] y Señales Digitales (GEySED) de la Universidad de las Ciencias Informáticas (UCI), se propone el desarrollo de un SIG para plataforma de escritorio que pueda ofrecerse al país como un producto completo y robusto. Con este propósito el proyecto SIG-Desktop se propone realizar el diseño de un paquete de Arquitectura de Software de la plataforma de desarrollo que pueda ser ajustable a diferentes necesidades de negocio, teniendo en cuenta el nivel de importancia de las funcionalidades de representación geográfica. Basado principalmente en la solución existente: Quantum GIS. QGIS es un sistema respaldado por una gran comunidad, mas hoy día no está orientado hacia otras áreas o negocios, pues no reconoce la existencia de otros sistemas. Ello se debe principalmente a que su arquitectura no está concebida para la comunicación con otras aplicaciones, y se basa únicamente en el trabajo con datos cartográficos.

Aún, cuando el número de SIG crece cada día en el mercado internacional, la mayoría de ellos no permiten la integración con otros sistemas existentes que puedan prescindir de sus servicios. Tal desventaja se hace grandemente notable cuando una aplicación necesita trabajar con datos geográficos pero su negocio no está orientado completamente al trabajo con los mapas, o de igual

[2] Es el uso de las matemáticas y las técnicas informáticas para resolver problemas geográficos utilizando programas informáticos y modelos matemáticos.

forma cuando se necesita representar información socio-económica en un mapa, y dicha información la suministra un software ajeno al sistema de información geográfica.

En la actual investigación se presenta un paquete de arquitectura de software para dar solución a la demanda de los sistemas de información geográficos, donde dicho conjunto de líneas base sirva de apoyo para orientar la construcción de los SIG que puedan integrarse a uno o varios negocios, lo cual permitirá que estos sistemas sean extensibles y flexibles. Dada la importancia que tiene la AS dentro del desarrollo de software de aplicaciones SIG se determina como **problema a resolver**: La Arquitectura de Software de QGIS no está orientada a ningún negocio ni reconoce la existencia de otras aplicaciones informáticas de negocio. Se define entonces como **objeto de estudio**: Las Arquitecturas de software de aplicaciones de escritorio y como **campo de acción**: Las Arquitecturas de software para soluciones informáticas de escritorio con funcionalidades de representación geográfica.

Se tiene entonces como **hipótesis:** La creación de un paquete de Arquitectura de Software que establezca las bases para la creación de SIG-Escritorio orientado a diferentes negocios, permitirá el reconocimiento de otras aplicaciones de la Informática para atender negocios con datos geoespaciales.

Para responder al problema de investigación se define como **objetivo general de la investigación:** Desarrollar un paquete de Arquitecturas de Software ajustadas al nivel de importancia de funcionalidades de representación geográfica en aplicaciones de escritorio, teniendo como **tareas de la investigación:**

1. Evaluar las funcionalidades de representación geográfica de diferentes soluciones informáticas existentes.
2. Caracterizar el estado del arte de temas de arquitectura de software en aplicaciones de escritorio y software libre reutilizable y extensible.
3. Caracterizar la Arquitectura de Software de la solución Quantum GIS (QGIS).
4. Desarrollar la Arquitectura de Software ajustada a cada uno de los niveles identificados.
5. Evaluar la Arquitectura de Software.

Para la realización del presente trabajo de diploma se pusieron en práctica varios métodos de la investigación: **Teóricos** y **Empíricos.** Dentro de los métodos teóricos utilizados se encuentra el **Analítico-Sintético** ya que fue necesario el análisis de las teorías y los documentos referenciados con el objetivo de extraer los elementos más importantes que se relacionan con el objeto de estudio planteado, y de plasmar la información encontrada en las diferentes bibliografías con el fin de lograr una alta comprensión del contenido. La necesidad de evaluar la trayectoria real de un

conjunto de AS existentes en el mundo de los SIG de escritorio, así como su evolución y desarrollo, con el fin de constatar teóricamente cómo han evolucionado y la vez el estudio de los estilos, patrones y lenguajes de descripción arquitectónicos de dichas arquitecturas de software en busca del más idóneo para la solución a brindar, hizo tangible el uso del **Análisis Histórico-Lógico.** Para la representación de los diagramas se hace uso de la **Modelación,** método que orienta al investigador cómo realizar los modelos que se establecen en la presente investigación.

Se emplean además otros métodos tales como la **Observación** y las **Entrevistas**, ambas pertenecientes a los métodos empíricos. El primero se evidencia en la observación a SIG de escritorio en funcionamiento para clasificar y analizar los efectos de la interacción de su arquitectura con ellos. Se realizaron además entrevistas a especialistas en el tema de AS y a arquitectos, desarrolladores de plataformas de sistemas de información geográfica, con el fin de que constituya un medio para el conocimiento cualitativo de la AS y su implementación en los SIG de escritorio.

Estructura y contenido. El documento está compuesto por tres capítulos, que incluyen aspectos relacionados con el trabajo investigativo realizado, además de la Introducción, Conclusiones de la investigación, Recomendaciones y Referencias Bibliográficas utilizadas, así como el Glosario de Términos y varios Anexos como material complementario para la comprensión del trabajo desarrollado.

En el ***Capítulo I.*** *Fundamentación Teórica.* Se hace un estudio del arte de la arquitectura de software y los sistemas de información geográficos, y se describen algunos conceptos fundamentales de la arquitectura a proponer.

El ***Capítulo II.*** *Descripción de la base arquitectónica*. Está dirigido a la muestra de cada uno de los modelos para describir la base de la solución a proponer, así como los atributos de calidad y los requisitos que tienen impacto en la arquitectura.

El ***Capítulo III.*** *Descripción del paquete de* arquitectura. Muestra las descripciones de cada una de las arquitecturas de software según los enfoques arquitectónicos, alto, medio, bajo. Basado principalmente en el capítulo 2 y orientado por el capítulo 1.

En el ***Capítulo IV.*** *Evaluación de la arquitectura.* Se realizará la evaluación de la arquitectura propuesta en el segundo capítulo y así conocer si se pueden habilitar los requisitos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los usuarios finales.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

1.1 Introducción

Este capítulo constituye una introducción a los elementos fundamentales de los Sistemas de Información Geográfica y a la Arquitectura de Software, con el propósito de brindar una visión de conjunto con una estructura coherente. Se establece el papel de la AS en relación con la estrategia arquitectónica de las herramientas SIG, los patrones de diseño así como el lenguaje de descripción de la arquitectura de software (por sus siglas en inglés ADL) y los modelos para describirla. De igual manera se incluye un estudio del estado del arte de la arquitectura de software en los sistemas de información geográfica de escritorio, a nivel internacional, nacional y de la Universidad. Se presentan además las tendencias, técnicas, tecnologías y metodologías usadas para la solución a proponer.

1.2 Referentes Teóricos de los SIG

Como expresión general, se puede plantear que un *Sistema de Información* (SI) radica en la unión de información en formato digital y herramientas informáticas, programas, para su posterior análisis con unos objetivos concretos dentro de una organización (empresa, administración, etc.). Un SIG es un caso particular de SI en el que la información aparece georreferenciada en mapas digitales, que como se había expresado anteriormente, no son más que SI utilizados como herramientas para el análisis, la consulta, la manipulación y el despliegue de información geográfica.

“Un SIG se define como un conjunto de métodos, herramientas y datos que están diseñados para actuar coordinada y lógicamente para capturar, almacenar, analizar, transformar y presentar toda la información geográfica y de sus atributos con el fin de satisfacer múltiples propósitos. Los SIG son una tecnología que permite gestionar y analizar la información espacial, y que surgió como resultado de la necesidad de disponer rápidamente de información para resolver problemas y contestar a preguntas de modo inmediato.” **(GEOENSEÑANZA 2006)**

Posibilitan representar en una computadora información espacial como redes de carreteras, tipos de suelo, límites entre ciudades y atributos de naturaleza descriptiva o estadística (información no espacial) como el tamaño de la población, volumen de tráfico entre otros y consultar las propiedades e información de los objetos definidos en él.

Los sistemas de información geográfica han sido consecuencia del automatismo de pesadas tareas de producción cartográfica ligadas a los sistemas digitales y evolucionando propiamente desde los años 1960 hasta nuestros días.

Durante los años 1960 y 1970 se comenzó a aplicar la tecnología del computador digital al desarrollo de tecnologías automatizadas. Excluyendo cambios estructurales en el manejo de la información, la mayoría de los programas estuvieron dirigidos hacia la automatización del trabajo cartográfico; algunos pocos exploraron nuevos métodos para el manejo de información espacial, y se siguieron básicamente dos tendencias: **(ALVARO DE J. CARMONA)**

- Producción automática de dibujos con un alto nivel de calidad pictórica, CAD[3]
- Producción de información basada en el análisis espacial pero con el costo de una baja calidad gráfica, SIG.

Al comenzar el 1970 se desarrollaron algoritmos que permitían generar las posiciones relativas mediante topología en capas (Layer en inglés). Esta técnica, llamada modelo orientado a capas, se impuso durante los 80′s y aun perdura en muchos estudios pues es la técnica más práctica y comercialmente distribuida.

En 1985 los ingleses crearon el modelo orientado a objetos donde se considera el "paisaje tal como lo es realmente": todo se conforma de partes y las partes se integran y forman objetos. [12]

Consecutivamente se fue implementando el continuo desarrollo de los SIGs, los cuales están fundamentados con la informática, lo que posibilitó que según el avance de esta última fuera el proceso de mejora de los sistemas de información geográficos.

1.2.1 Enfoques de funcionalidades SIG

Los sistemas de información geográficas, como herramientas muy importantes en la toma de decisiones, son muy utilizados en disímiles centros a manera de apoyo al negocio en el que se desarrollan, ya sea en industrias petroleras, en la defensa nacional de un país u otros ámbitos en los que estos sistemas, de alguna forma u otra, sean necesarios. Para resaltar y entender la importancia de las funcionalidades de los SIG, se propone dividirlos en tres categorías:

Nivel de importancia alto (100% SIG, sin negocio): Sistema de Información Geográfica con la mayoría de sus funcionalidades y dedicado única y exclusivamente al trabajo con datos cartográficos. No permite la comunicación con otro sistema o negocio. Este es el estado actual de QGIS.

Nivel de importancia medio (50% SIG, 50% negocio): El SIG se fusiona con otro sistema informático como parte importante donde va a existir una amplia comunicación entre ambos

[3] Diseño asistido por ordenador (Computer Aided Design), software utilizado para el diseño de precisión.

negocios. No es necesario que el SIG posea todas sus funcionalidades, pero sí las más importantes necesarias para el sistema con el que se comunica.

Nivel de importancia bajo (20% SIG, 80% negocio): El SIG se comunica con un sistema informático, donde el primero no es de mucha importancia o relevancia en el negocio. En esta calificación el sistema de información geográfica no posee más funcionalidades que las necesarias para el sistema con el que establece la comunicación, en este caso con mucho menos características que la del nivel medio, lo que lo deja en las funciones básicas.

Como QGIS no cuenta con ningún tipo de integración, de ahí la importancia de crear arquitecturas de software para sistemas de información geográfica según el nivel de importancia que tengan o atribuyan a un proyecto o producto.

1.3 Referentes Teóricos de la AS. Elementos de la Arquitectura de Software

La Arquitectura de Software (AS), organización fundamental de un sistema, está diseminada en el mundo con una gran polarización en el consenso conceptual. Tiene un alto grado de integración con otras disciplinas del desarrollo de software y un alto impacto en los resultados de los productos. Se centra en los requisitos no funcionales que se satisfacen mediante diseños y modelos de la aplicación. Es representada a través de sus componentes y la relación entre ellos, además del entorno y los principios que orientan su diseño y programación. Sus inicios datan desde 1960, con las tempranas observaciones de Edsger Dijkstra, David Parnas y Fred Brooks.

En sus inicios el surgimiento de la construcción de software era desorganizado y se basaba solo en la implementación. Para 1968, Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda, propone que se estableciera una estructuración correcta de los sistemas de software antes de lanzarse a programar. Dijkstra sostenía que las ciencias de la computación eran una rama aplicada de las matemáticas y sugería seguir pasos formales para descomponer problemas mayores. **(DIJKSTRA)**

En 1975, Frederick Phillips Brooks Jr. diseñador del sistema operativo OS/360, utilizaba el concepto de arquitectura del sistema para designar *la especificación completa y detallada de la interfaz de usuario*. Y además, distinguía entre arquitectura e implementación; mientras que la primera decía *qué hacer*, la segunda se ocupaba del *cómo hacerlo*. Durante algún tiempo la AS quedó congelada, hasta 1992 cuando Dewayne Perry y Alexander Wolf publicaron un artículo bajo el nombre “Foundations for the study of software architecture”, donde exponen lo que sería la década de 1990 para la arquitectura de software.

“La década de 1990, creemos, será la década de la arquitectura de software. Usamos el término *arquitectura*, en contraste de *diseño*, para evocar nociones de codificación, de abstracción, de las normas, de formación (de arquitectos de software), y de estilo.” **(DEWAYNE E. PERRY 1992)**

En 1996, Shaw y Garlan en su libro *Software Architecture. Perspectives of an Emerging Discipline* plantean que en esta disciplina resultan muy importantes los elementos como: los protocolos para la comunicación, la sincronización, y el acceso a los datos; la asignación de la funcionalidad para diseñar elementos; la distribución física; la composición de los elementos del diseño; el escalamiento y el funcionamiento; y la selección entre alternativas del diseño.

1.3.1 Definiciones de AS

Hoy día existen numerosos conceptos en el plano detallado de la AS, pero todos se articulan alrededor de algunos principios esenciales: guía todo el proceso de desarrollo de los sistemas informáticos de software y debe garantizar que sean cumplidos los requisitos funcionales y no funcionales descritos por el cliente. La misma se refina durante todo el ciclo de vida de los proyectos y contribuye a que no excedan los límites de costo y tiempo establecidos en ella. Se puede manejar además como un conjunto de decisiones a tener en cuenta en el diseño de un proyecto, las cuales deben estar correctamente planteadas, de no ser así se podría anular todo un producto. Dos de los conceptos más reconocidos o usados son los mencionados por la IEEE Std 1471-2000 y el de Clements expresados con anterioridad y a los cuales se ajusta presente investigación, además de otros como:

...“Arquitectura de software es el estudio de la estructura a gran escala y el rendimiento de los sistemas de software. Aspectos importantes de la arquitectura de un sistema incluye la división de funciones entre los módulos del sistema, los medios de comunicación entre módulos, y la representación de la información compartida.” **(YAMILA VIGIL REGALADO 2008)**

… “Arquitectura es un nivel de diseño que hace foco en aspectos más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema" **(DAVID GARLAN 1994)**

1.3.2 Estilos arquitectónicos

Los estilos arquitectónicos se pueden definir como la categorización de un sistema además de describir sistemas completos. Plantean de manera global la solución a determinados problemas con un alto nivel de abstracción y definen de igual manera la arquitectura a aplicar. Brindan una estructura en cómo deben estar organizada la aplicación en cuanto a los componentes, los

conectores, las configuraciones y las restricciones de estos elementos y las relaciones que pueden existir entre ellos.

Algunas definiciones por autores dedicados al tema son:

"... una familia de sistemas de software en términos de un patrón de organización estructural, que define un vocabulario de componentes y tipos de conectores y un conjunto de restricciones de cómo pueden ser combinadas. Para muchos estilos puede existir uno o más modelos semánticos que especifiquen cómo determinar las propiedades generales del sistema partiendo de las propiedades de sus partes." es la definición que le dan Shaw y Garlan **(DAVID GARLAN 1994)**

"…expresan esquemas de organización estructural fundamentales para los sistemas de software. Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen guías y lineamientos para organizar las relaciones entre ellos." **(CARLOS REYNOSO 2004)**

Por ende se puede decir que un estilo arquitectónico es una transformación impuesta al diseño de todo un sistema con el objetivo de establecer una estructura para todos los componentes de dicho sistema.

El paquete de AS que se propone para el Sistema de Información Geográfica de escritorio GeoQ está fundamentado por el estilo arquitectónico orientado a objetos (AOO), aunque también hace uso de la arquitectura basada en componentes (ABC) ya que se requiere una arquitectura de sistema flexible y fácil de personalizar. Ello permite que se le integren plugin que permitan más funcionalidades al sistema según sean necesarias. A continuación sus descripciones:

Arquitecturas Orientadas a Objetos: También reconocido por algunos autores como Arquitecturas Basadas en Objetos u Organización Orientada a Objetos. En este estilo los componentes son los objetos o más bien instancias de los tipos de dato abstractos. Estos componentes del sistema encapsulan datos y operaciones que deben usarse para manipular dichos datos. La comunicación y coordinación entre esos componentes se realiza mediante envío de mensajes. Este es un sistema donde se enfatiza el empaquetamiento entre datos y operaciones que permiten manipular y acceder a dichos datos. **(PRESSMAN 2005)**

Como se explica, la misma estructura el sistema en un conjunto de objetos débilmente acoplados y con interfaces bien definidas donde los objetos hacen llamadas a las funciones de otros objetos. Una organización orientada a objetos está relacionada con las clases de objetos, sus atributos y sus operaciones. Cuando se implementa, los objetos se crean a partir de estas clases y se usan

algunos modelos de control para coordinar las operaciones de dichos objetos. **(SOMMERVILLE 2005)**

Ello se traduce en la descomposición del diseño en componentes prácticos o lógicos que exponen interfaces de comunicación bien definidas. La misma no se enfoca en asuntos específicos de los objetos. Describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema.

Las ventajas de este estilo radican en que como los objetos están débilmente acoplados, la modificación de ellos puede efectuarse sin comprometer a los otros. Los mismos son representaciones de entidades del mundo real lo que posibilita que la estructura del sistema sea fácilmente comprensible; y ello a la vez permite la reutilización de dichos objetos dado que las entidades en el mundo real se usan en sistemas diferentes.

Arquitectura Basada en Componentes: Esta arquitectura se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas. Esto provee un nivel de abstracción mayor que los principios de orientación por objetos y no se enfoca en asuntos específicos de los objetos como los protocolos de comunicación y la forma como se comparte el estado **(SIVARAMAKRISHNAN SOMASEGAR 2009)**. Es decir, esta descomposición posibilita la transformación de los componentes pre-existentes en piezas más grandes de un software. Es necesario conocer que es un componente de software para su mejor comprensión, y el mismo se puede definir como algo que puede ser utilizado como una caja negra, de manera que se tiene una especificación externa general, y esta es completamente independiente de la especificación interna. Es en sí un objeto de software diseñado para cumplir con cierto propósito. Un componente debe poseer principios fundamentales como: *Reusable, Sin contexto específico, Extensible, Encapsulado e Independiente.*

Entre los principales beneficios de este estilo, se encuentran la facilidad de instalación que permite reemplazar una nueva versión del sistema por la existente sin impacto en el mismo o en sus componentes. El uso de componentes de terceros permite distribuir el costo del desarrollo y del mantenimiento, reduciendo el mismo considerablemente. Otra de las ganancias más notables con este estilo es la facilidad del desarrollo ya que al igual que AOO los componentes implementan un interface bien definida lo que permite el desarrollo sin impactar otras partes del sistema. Otro de los más significativos está muy ligado a uno de los principios, la reusabilidad, pues los componentes pueden ser usados para distribuir el desarrollo y el mantenimiento entre múltiples aplicaciones y sistemas. La mitigación de complejidad técnica constituye otro de los beneficios de este estilo, consiste en la mitigación de la complejidad por medio del uso de contenedores de componentes y sus servicios.

Es importante señalar que aunque este estilo tiene sus similitudes al estilo AOO, el primero proporciona un mayor nivel de abstracción que los principios de diseño del segundo.

1.3.3 Patrones

Un patrón arquitectónico, al igual que un estilo, impone una transformación en el diseño de una arquitectura **(PRESSMAN 2005)**, es por ello que ambas terminologías tienden a usarse en el mismo contexto. Sin embargo se podría decir que los estilos están vinculados a la forma más general en que está organizado un sistema de software, independientemente de las consideraciones más específicas, es decir que el estilo está asociado a formas generales de organización. Mientras que los patrones estarán asociados a formas más concretas, que tienen que ver con la especialización que adoptan los objetos y clases de acuerdo al tipo de aplicación o entorno tecnológico, mas ellos difieren en varios elementos. Desde la perspectiva de los patrones:

a) Se concentran en aspectos específicos y no en toda la arquitectura por lo que su alcance es menor.
b) Impone reglas sobre la arquitectura definiendo la manera en la que el software manejará determinada funcionalidad al nivel de la infraestructura.
c) Abarcan aspectos específicos del comportamiento dentro de la arquitectura.

Aunque existan sus diferencias (Anexo 1), los patrones van muy ligados a los estilos arquitectónicos, pues ambos determinan la forma de la estructura general de un sistema y plantean un lenguaje común a una comunidad de desarrolladores para comunicar experiencia sobre los problemas y sus soluciones.

Tal y como se había expuesto, un patrón se puede definir como una solución a determinados problemas de diseño que aparecen con frecuencia y pueden ser reutilizados tantas veces sea necesario. Generalmente se documentan en plantillas estándares las cuales los describen asignándoles un nombre, un resumen del problema y la razón que lo hacen surgir y una solución en términos de colaboración de clases participantes e interacción entre objetos de esas clases.

Existe un relación intrínseca entre los niveles de abstracción de los estilos arquitectónicos, los patrones arquitectónicos y los patrones de diseño, lo que permite la comprensión del planteamiento de Buschmann **(SOMMERVILLE 2005)**, que proponen el desarrollo de arquitecturas de software como un sistema de patrones, y distintos niveles de abstracción:

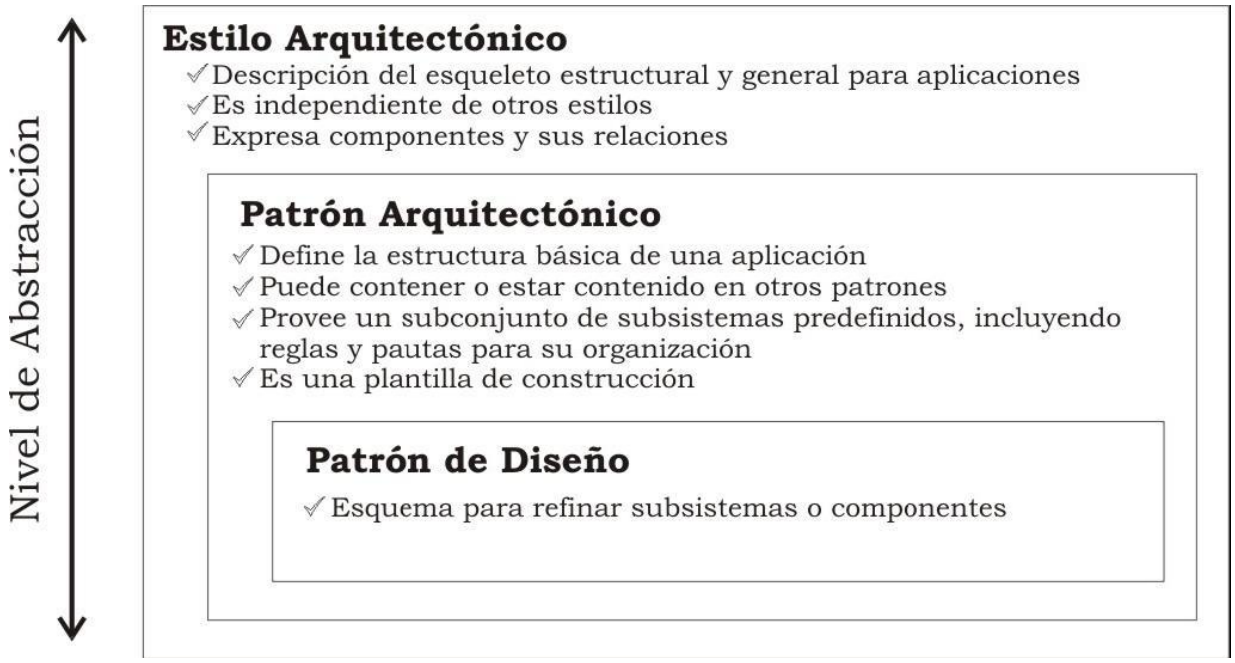

Ilustración 1. Relación de abstracción entre estilo arquitectónico, patrón arquitectónico y patrón de diseño

Los patrones definidos para el diseño e implementación de la solución propuesta son: Orientado a Objeto y Orientado a Componentes, ambos coinciden con los estilos arquitectónicos seleccionados y descritos. Los mismos cumplen iguales principios que los estilos pero a un nivel más determinante en la arquitectura, puesto que estarán asociados a formas más concretas y a la especialización que adoptan los objetos y clases.

Como expresan los autores de Pattern-Oriented Software Architecture (POSA), los patrones de diseño son los patrones de mediana escala, siendo un poco más específicos que los patrones arquitectónicos y tienden a ser independientes de un lenguaje o paradigma de programación. La aplicación de un modelo de diseño no tiene ningún efecto sobre los aspectos fundamentales de la estructura de un sistema de software, pero puede tener una fuerte influencia en la arquitectura de un subsistema **(MARCA HUALLPARA HUGO MICHAEL)**. Los patrones de diseño además de ser soluciones a problemas comunes de diseño en un determinado contexto y estar presentes en el desarrollo de software; permiten formalizar un vocabulario común entre los diseñadores del sistema y estandarizar el modo en que se realiza el diseño. Su utilización posibilita entender, mantener y ampliar el sistema de manera fácil, así como contribuir a la reutilización y diseño de componentes de software, a la organización del código, a la flexibilidad y extensibilidad, y la facilidad de realizar cambios en el sistema.

Los patrones de diseño identificados para el desarrollo del SIG son:

GRAPS (General Responsability Assignment Software Patterns)

a) Experto (Expert): La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase que contiene toda la información necesaria para realizar la labor que tiene encomendada.
b) Creador (Creator): Es el que crea y guía la asignación de responsabilidades relacionadas con la creación de objetos. Se asigna la responsabilidad de que una clase B cree un objeto de la clase A. Esta tarea es muy frecuente en los sistemas orientados a objetos.
c) Controlador (Controller): Es un evento generado por actores externos. Se asocian con operaciones del sistema como respuestas a los eventos del sistema, tal como se relacionan los mensajes y los métodos. Un controlador delega en otros objetos el trabajo que se necesita hacer y coordina o controla la actividad. No realiza mucho trabajo por sí mismo.

GoF (Gans of Four)

a) Patrones Creacionales (Creational Patterns)

1. Factoría Abstracta (Abstract Factory): Abastecer una interfaz con el fin de crear familias de objetos, que dependen entre sí o estén relacionados, sin especificar sus clases concretas. Abstraer algunas clases que se instancian y encapsular la funcionalidad que se desea. Es la vía de comunicación hacia las clases concretas instanciadas.
2. Método Factoría (Factory Method): Utilizar la abstracción de clases para crear y relacionar objetos sin tener en cuenta de qué clase proviene. Permite definir una interfaz para crear un objeto y posibilita que sean las subclases quienes decidan que clases instanciar. Permite que una clase delegue en sus subclases la creación de objetos (Constructor Virtual).

b) Patrones Estructurales (Structural Patterns)

1. Bridge - Handle/Body: Separar una abstracción de su implementación de forma que ambas pueden evolucionar por separado y así la abstracción pueda tener varias implementaciones posibles. Es decir, una clase abstracta define la interfaz y clases concretas la implementa de diferentes formas. El camino habitual para acomodarlas es a través del uso de la herencia.
2. Fachada (Facade): Proporcionar una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar. Reduce la dependencia entre clases, ya que ofrece un punto de acceso al resto de las clases, si estas cambian o se sustituyen por otras solo hay que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones cliente.

c) Patrones de Comportamiento (Behavioral Patterns)

1. Chain of Responsability: Delegar responsabilidades a una clase especializada. Permite distribuir la ejecución repartiendo las responsabilidades y constituye la forma básica al realizar un Diseño Orientado a Objetos, por lo que su uso es de forma intuitiva en este tipo de diseño.

1.3.4 Lenguajes de descripción de la arquitectura (ADLs)

Lenguaje de Descripción de la Arquitectura, por sus siglas en inglés ADL, es un término que se ha utilizado mucho en el contexto de diseño de software, los cuales son muy usados para el análisis, la representación y las pruebas de la arquitectura de software con el fin de determinar la consistencia arquitectónica. Los mismos capturan las especificaciones de comportamiento de los componentes y las interacciones que comprenden dicha arquitectura para su posterior descripción. Estos lenguajes se utilizan para expresar formalmente una vía para la representación de la arquitectura, además de permitir la observación en cuanto a la integridad, el rendimiento, la coherencia y la ambigüedad de una AS.

"Los ADLs permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento." **(OSVALDO DÍAZ VERDECIA 2009)**

Pese a la variedad de lenguajes para describir la arquitectura de software (Anexo 4), se escoge a UML como lenguaje descriptor, de manera que aunque el mismo no sea categorizado como ADL porque carece del concepto de estilo y sus definiciones de arquitectura no guardan relación con lo que ella significa en el campo de los ADL; es conocido en la industria y soportado por diversas herramientas y metodologías de desarrollo, siendo adoptado como notación estándar por empresas productoras de aplicaciones informáticas a nivel mundial. Posee como principal ventaja que puede ser usado en todas las etapas de desarrollo de un sistema y su representación gráfica permite ser usada para la comunicación con los usuarios. Como características a destacar están: que es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema, y que ofrece un estándar para describir un modelo del sistema, incluyendo aspectos conceptuales tales como: procesos de negocio, funciones del sistema, esquemas de bases de datos y componentes reutilizables.

1.3.5 Modelos para describir la arquitectura de software

Como se expresa anteriormente la AS debe describir diferentes aspectos del software para la que fue concedida y cada uno de estos aspectos se describe de una manera más perceptible por medio de la utilización de modelos o vistas. Los modelos para la descripción de la arquitectura de software son diagramas creados utilizando estándares disponibles, en los que la preocupación principal es mostrar un conjunto específico de los factores inherentes a la estructura y el diseño de

un sistema, con la información sobre las interfaces y las interacciones dinámicas entre los subsistemas. Igualmente son utilizados por los arquitectos de software para comunicarse con los demás y retroalimentar sus conocimientos.

Cada uno de estos modelos constituye una descripción parcial de una misma arquitectura y es deseable que exista cierto solapamiento entre ellos, asegurando que todas las vistas sean coherentes entre sí.

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

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

Estas vistas se pueden expresar a través de diagramas de flujos de datos, diagramas de estado, etc. Los cuales constituyen los lenguajes de expresión de los modelos o vistas, y que por ende solo son apropiados para estas. En la presente investigación se estará trabajando sobre el lenguaje unificado de modelado (UML, por sus siglas en inglés: Unified Modeling Language).

1.4 Arquitectura de software en la metodología utilizada

Las metodologías de software definen Quién debe hacer Qué, Cuándo y Cómo debe hacerlo para obtener los distintos productos en un determinado proceso de software. Las mismas guían a los desarrolladores en el diseño e implementación del producto. Existen numerosas propuestas metodológicas (Anexo 3) que inciden en distintas dimensiones del proceso de desarrollo, estas se pueden clasificar en dos grupos:

- Metodologías Ágiles o Ligeras: Están orientadas a la interacción permanente con el cliente y el desarrollo incremental del software que implementa. Muestra versiones parcialmente funcionales del software al consumidor en cortos periodos de tiempo para que pueda evaluar y sugerir cambios en el producto según se va desarrollando.

- Metodologías Tradicionales o Pesadas: Están orientadas al control de los procesos, estableciendo rigurosamente las actividades a desarrollar, herramientas a utilizar y notaciones que se usarán.

Como se expresa ambos grupo o clasificaciones poseen notables diferencias (Anexo 2) y por ende responden a diferentes guías de desarrollo que se utilizan según la complejidad del producto a realizar y lo que exija el usuario final.

1.4.1 Microsoft Solutions Framework (MSF)

Incluye una serie de conceptos, modelos y prácticas de usos que apoyan a los equipos de desarrollo, estas últimas controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. Se centra en los modelos de proceso y de equipo asumiendo que no son prioridades las elecciones tecnológicas. Todo ello se traduce a que MSF se compone de principios, modelos y disciplinas.

Entre los modelos se encuentran:

- El modelo de equipo incita a poseer la agilidad para hacer frente a nuevos retos y cambios haciendo partícipe a todo el equipo en las decisiones fundamentales.

- El modelo de proceso mediante la estrategia iterativa en la construcción de los productos del proyecto, proporciona una visión más clara del estado de los mismos en cada etapa.

Las disciplinas que rigen a MSF son:

- Gestión de proyectos: Describe el rol de la gestión del proyecto dentro del modelo de equipo de MSF, que permite alta escalabilidad tanto en proyectos pequeños como largos y complejos.

- Control de riesgos: Ayuda al equipo a establecer las prioridades, a tomar las decisiones estratégicas correctas y a controlar las incidencias que puedan surgir.

- Control de cambios: Los cambios deben considerarse como riesgos inherentes. Estos cambios deben registrarse.

Siendo MSF una metodología que propone que la tecnología está en un segundo plano, no es la más propicia para el desarrollo de GeoQ puesto que además de los datos la tecnología es un factor muy importante en el desarrollo de un SIG.

1.4.2 Programación Extrema (XP)

Metodología ágil que está basada en la simplicidad de las soluciones que se implementan, la comunicación entre los integrantes del proyecto y la retroalimentación del cliente y el equipo de desarrollo. Diseñada para proyectos de corto plazo y su objetivo principal es la programación

rápida. Se centra en la reutilización de código, para lo cual se crean patrones o modelos estándares lo que la hace más flexible al cambio.

Posee 5 principios básicos que le dan cumplimiento a los basamentos que lo soportan: realimentación rápida, asumir la simplicidad, cambio incremental, adherirse al cambio y trabajo de alta calidad. Estos principios así como la práctica son de sentido común pero llevadas al extremo, de ahí su nombre.

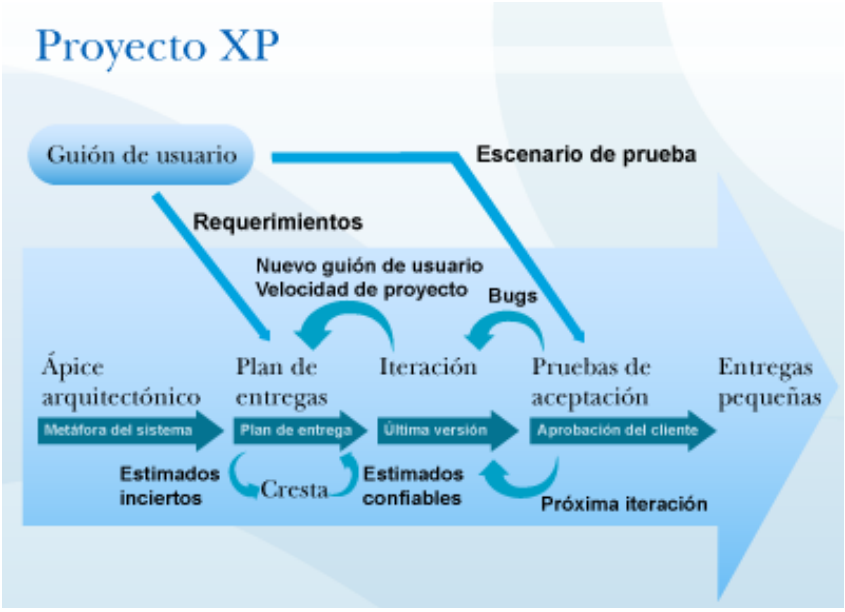

Ilustración 2. Flujos de trabajos en un proyecto XP

De igual manera eXtreme Programming no es factible para el proyecto SIG-DESKTOP puesto que como metodología ágil solo se compromete con pequeñas soluciones y requiere una constante interacción con el cliente, lo que no es posible si este último no se encuentra en la misma nación, además, no define una arquitectura temprana y en cambio la va reconstruyendo en consecuencia de las versiones vs requisitos que va presentando el usuario final, lo que para proyectos de gran escala sería un costoso gasto de tiempo y recurso.

1.4.3 Rational Unified Process (RUP[4])

Debido a la complejidad y alcance de la propuesta del software a realizar, la investigación se desarrollará sobre la metodología de desarrollo RUP ya que es la metodología más acorde, dada la magnitud del proyecto, además de ser una metodología bien documentada y ejemplificada. Su ciclo de vida se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y ser iterativo e incremental.

[4] Proceso Unificado de Desarrollo de Software es una metodología de desarrollo para proyectos de gran envergadura, por sus siglas en inglés: Rational Unified Process.

La familia de productos RUP proporciona un número de herramientas que permiten y simplifican la definición, configuración y personalización del proceso de ingeniería y proporciona un número de vistas que se centran en diferentes grupos de profesionales de la ingeniería de software.

RUP está constituido por 9 flujos de trabajo, los 6 primeros son conocidos como flujos de ingeniería y los 3 últimos de apoyo. Todos tienen lugar sobre 4 fases de desarrollo.

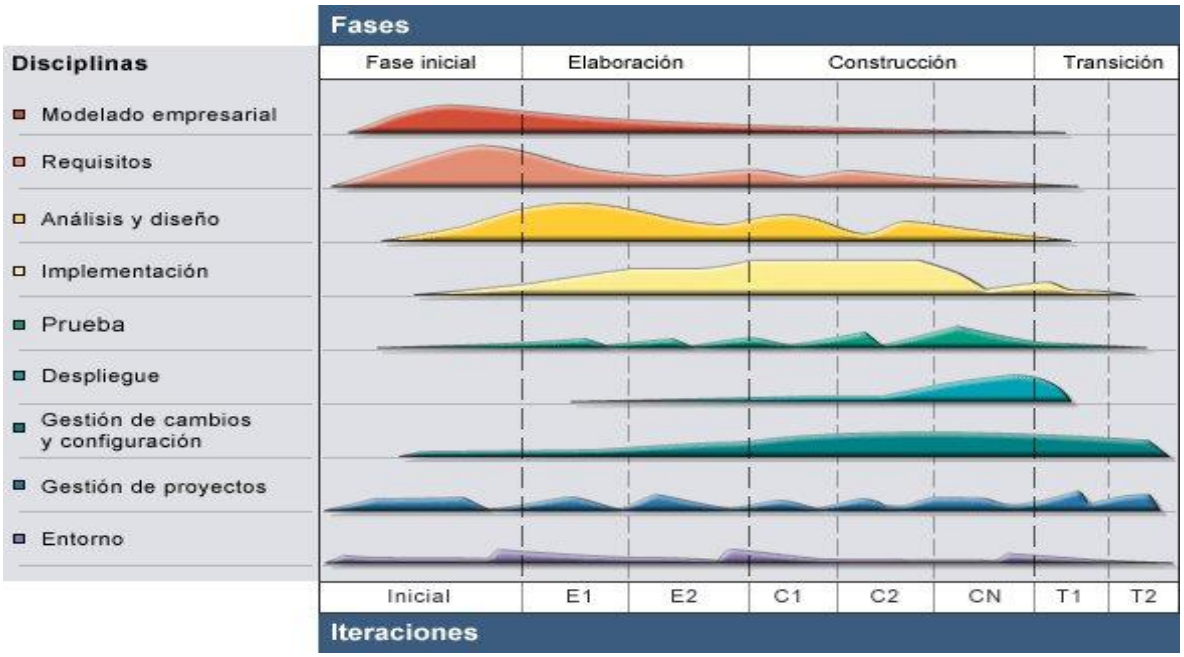

Ilustración 3. Flujos y fases de trabajos de RUP

Centrado en la arquitectura es un concepto que RUP maneja como la implicación de los elementos más significativos del sistema. Está influenciada por las plataformas de *software*, los sistemas operativos, los sistemas de gestión de bases de datos, los sistemas heredados y los requisitos no funcionales. Se representa mediante varias vistas que se centran en aspectos concretos.

"RUP es un proceso configurable. Un simple no es adecuado para todo el desarrollo de software. Se ajusta a pequeños equipos de desarrollo así como a las grandes organizaciones de desarrollo. Se basa en una arquitectura de proceso simple y transparente que permite una uniformidad en toda una familia de procesos. Sin embargo, se pueden variar para adaptarse a diferentes situaciones. Contiene un kit de desarrollo, proporcionando soporte para configurar el proceso para adaptarse a las necesidades de una organización determinada." **(JAMES RUMBAUGH)**

1.5 Selección y descripción de las tecnologías a utilizar

Para el desarrollo de un sistema de información geográfica se han de necesitar varias tecnologías y herramientas que le den soporte a dicho sistema y aseguren su perfecto funcionamiento junto a la satisfacción del cliente o usuario final. Dichas tecnologías serán libres bajo licencia GPL[5] implementadas y desarrolladas para Sistemas de Información Geográfica en entornos de escritorio.

1.5.1 Entornos de Desarrollo Integrado (IDE)

Los Entornos de Desarrollo Integrado, o en inglés Integrated Development Environments (IDE), son ambientes para escribir la lógica y el diseño de interfaces de aplicaciones; es decir son programas compuestos por un conjunto de herramientas que le facilitan a un programador desarrollar un determinado software. Estos entornos de programación son empaquetados como un programa de aplicación y consisten en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Pueden dedicarse en exclusiva a un sólo lenguaje de programación o bien para varios y han de ser aplicaciones por sí solas o parte de otros sistemas.

Eclipse: IDE de código abierto y multiplataforma muy utilizado principalmente para desarrollar aplicaciones en Java. Posee una gran comunidad de usuarios nombrada de igual modo la cual extiende constantemente las áreas de aplicación cubiertas.

El mismo provee al implementador frameworks con diversas funcionalidades útiles para el desarrollo de aplicaciones gráficas, aplicaciones web y definición y manipulación de modelos de software. Así como es poseedor de un excelente editor de texto capaz de resaltar las sintaxis, la compilación la realiza en tiempo real. Emplea módulos (plugin) para proporcionar funcionalidades a plataformas de escritorio, a diferencias de otros IDE que son monolíticos. Ello le permite extenderse hacia el uso de otros lenguajes de programación tales como C++ y Phyton. Para el trabajo con estos lenguajes se les integran los módulos: CDT para aplicaciones programadas con C++ y PyDev para aquellas que contengan códigos en phyton.

QT-Creator: IDE multiplataforma que permite desarrollar aplicaciones implementadas con el lenguaje de programación C++ de manera sencilla y rápida. Es soportado principalmente por Windows XP y Vista, Linux y Mac OS X. Como características principales posee:

a) Editor avanzado para C++ y JavaScript.
b) Herramientas para la administración y construcción de proyectos.

[5] Licencia Pública General de GNU o más conocida por su nombre en inglés GNU General Public License.

c) Completado automático.
d) Depurador Visual.
e) Soporte para el control de versiones.
f) Simulador para interfaces de usuarios móviles.
g) Soporte para objetivos de escritorios y portátiles.

Este entorno de desarrollo permite crear aplicaciones de escritorio y plataformas de dispositivos móviles para diversos SO haciendo uso del CMAKE, el cual será tratado posteriormente. Está basado completamente en la librería Qt y tiene su código fuente bajo la licencia GNU GPL, de manera que los proyectos implementados en este entorno deberán ser desarrollados bajo esta licencia.

CMAKE es una herramienta multiplataforma que se utiliza para generar y/o automatizar código. Es decir, constituye un conjunto de herramientas diseñadas para construir, probar y empaquetar software independientemente de la plataforma en la que se desarrolle. Para lograr esto, genera ficheros makefiles nativos y espacios de trabajo que pueden usarse en el entorno de desarrollo deseado. Ello posibilita la generación de ficheros para varios sistemas operativos, lo que flexibiliza el mantenimiento y no se hace necesario poseer varios conjuntos de ficheros para cada plataforma.

1.5.2 Lenguajes de programación

Los lenguajes de programación constituyen instrumentos que posibilitan implementar programas y software. Ellos facilitan la labor de programación, pues disponen de formas adecuadas que permiten ser leídas y escritas por personas. Resultan independientes del prototipo de ordenador a utilizar.

PYTHON: es un lenguaje dinámico de programación de alto nivel cuya filosofía refiere una sintaxis muy limpia y que favorezca un código legible. Es multiparadigma lo que significa que permite a los desarrolladores el uso de varios estilos: programación orientada a objetos, programación imperativa y programación funcional; en vez de forzarlos a adoptar un estilo particular de programación. Permite escribir nuevos módulos fácilmente en C o C++ y puede incluirse en aplicaciones que necesitan una interfaz programable.

La implementación de Python está bajo una licencia de código abierto denominada Python Software Foundation License, que es compatible con la licencia pública general de GNU lo que atribuye a que sea de libre uso y distribución, incluso para el comercio. Puede ser utilizado en disímiles sistemas operativos tales como: Windows, Linux / Unix, OS / 2, Mac, entre otros.

El uso de dicho lenguaje será para la implementación de funcionalidades que resulten más fácil realizar en él que en C++ y principalmente para el desarrollo del core[6] de conexión con los plugins de integración al sistema a desarrollar.

C++: es un lenguaje de programación de propósito general diseñado con la intención de extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos llegando a ser un lenguaje híbrido. Es, por ende, un lenguaje de programación multiparadigma, poseedor de programación genérica, que se suma a los otros dos paradigmas ya existentes para ese entonces: programación estructurada y programación orientada a objetos.

Con el surgimiento de este potente lenguaje se agrega un concepto llamado sobrecarga de operadores lo que brinda la posibilidad de redefinir los operadores y poder crear nuevos tipos que se comporten como tipos fundamentales. C++ tiene características que simplifican la gestión de memoria, permite el acceso de bajo nivel a la memoria y a la vez contiene características de alto nivel.

Este lenguaje será principalmente usado en la implementación del negocio del sistema de información geográfica, por sus características.

1.5.3 Herramientas CASE

Una herramienta CASE es una tecnología para automatizar el desarrollo y mantenimiento del software, combinando herramientas de software y metodologías. Están destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de la misma en términos de tiempo y dinero. Constituyen un conjunto integrado que automatiza todas las partes del ciclo de vida de desarrollo de un software y por tanto ahorra trabajo. Son las aplicaciones en las que se apoyan los ingenieros y arquitectos de software para realizar las tareas de diseño de un proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, entre otras.

Rational Rose Enterprise: Herramienta muy potente que soporta el UML y facilita la generación de código para múltiples lenguajes de programación. Entre las principales características se encuentran:

a) Soporte para análisis de patrones ANSI C++, Rose J y Visual C++ basado en "Design Patterns: Elements of Reusable Object-Oriented Software"
b) Característica de control de componentes de modelo por separado, que permite una administración más granular y el uso de modelos.

[6] Núcleo o centro de un programa, del cual va a depender gran parte de él.

c) Soporte de ingeniería Forward y/o Reversa.

d) Generación de código ANSI C++, C++, Java y Visual Basic, con capacidad de sincronización modelo- código configurables

e) Capacidad de análisis de calidad de código

Otra de las ventajas de esta herramienta es el soporte de RUP para la ingeniería del sistema. Una de las principales desventajas es que no es multiplataforma, condición primordial para el modelado del sistema a implementar, por ello no es factible para la realización de los modelos del presente trabajo.

Visual Paradigm: Es una herramienta UML para el desarrollo de aplicaciones de software que soporta y ofrece un conjunto de herramientas útiles para los desarrolladores de un proyecto. Necesaria para la captura de requisitos, planificación de software, modelado de clases entre otras funcionalidades. Permite dibujar mayoritariamente todos los diagramas de clases además de admitir código inverso, generación de código desde diagramas y generación de la documentación.

Entre otras de sus principales características se encuentran:

a) Editor de Detalles de Casos de Uso - Entorno para la especificación de los detalles.

b) Soporte ORM - Generación de objetos Java desde la base de datos.

c) Generación de bases de datos. Transformación de diagramas de Entidad-Relación en tablas de base de datos.

d) Distribución automática de diagramas. Reorganización de las figuras y conectores de los diagramas UML.

e) Integración con Visio. Dibujo de diagramas UML con plantillas de MS Visio Editor de figuras.

Como principal característica posee que es multiplataforma, lo que permite y facilita el trabajo en varios escenarios; de ahí la selección de esta herramienta CASE para la representación de todos los modelos a realizar en la presente investigación.

1.5.4 Sistema Gestor de Base de Datos

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

“Un sistema gestor de base de datos se define como el conjunto de programas que administran y gestionan la información contenida en una base de datos.” **(ALVAREZ)**

Estos sistemas ayudan a definir los datos, mantener la integridad de estos dentro de la base de datos, controlar la seguridad y privacidad de los datos y manipularlos.

Los SGBD están compuestos por un gestor de base de datos, que comprende un conjunto de programas encargados de la privacidad, integridad y seguridad de los datos y la interacción con el sistema operativo. El mismo suministra una interfaz entre los datos, programas que los operan y los usuarios finales. El diccionario de datos es otros de los componentes de un SGBD y se puede describir como una base de datos (BD) donde se almacenan todas las propiedades de la BD, descripción de la estructura y las relaciones entre los datos. Otro de los componentes es el administrador de la base de datos, que se refiere a la persona responsable del control del SGBD. Y por último se encuentran los lenguajes, que son los encargados de definir y manipular la base de datos.

PostgreSQL 8.3

Para el almacenamiento de los datos persistentes del producto GeoQ, se selecciona como SGBD a PostgreSQL en su versión 8.3, el cual es catalogado el Sistema de Gestión de Bases de Datos de código abierto y gratuito más avanzado del mundo. Posee las características de los más potentes sistemas comerciales como Oracle o SQL Server.

“El Sistema Gestor de Bases de Datos Relacionales Orientadas a Objetos conocido como PostgreSQL está derivado del paquete Postgres escrito en Berkeley. Con cerca de una década de desarrollo tras el mismo, PostgreSQL es el gestor de bases de datos de código abierto más avanzado hoy día, ofreciendo control de concurrencia multi-versión, soportando casi toda la sintaxis SQL (incluyendo subconsultas, transacciones, y tipos y funciones definidas por el usuario), contando también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java, perl, tcl y python).” **(POSTGRESQL)**

Entre las características más notables que posee están:

1. Completo soporte para transacciones. Una transacción está formada por un conjunto de acciones de forma que se ejecutan todas o ninguna de ellas. Utilizando transacciones se asegura la consistencia de los datos.
2. Soporte completo ACID (Atomicity Consistency Isolation Durability): Es posible definir operaciones atómicas, es decir, formadas por comandos que se ejecutan todos o ninguno.
3. Consistencia, que garantiza que la base de datos nunca se queda en un estado intermedio de una transacción (con parte de los comandos ejecutados y parte no).

4. Aislamiento, que mantiene separadas las transacciones de usuarios distintos hasta que éstas han terminado.
5. Durabilidad, garantiza que el servidor de datos guarde las actualizaciones pendientes de forma tal que pueda recuperarse de una terminación inesperada.

La elección de este potente sistema gestor de base de datos se basa además en que es multiplataforma, de código abierto y posee una integridad referencial que permite un buen grado de seguridad en los datos, además de su extensión PostGIS que será analizada posteriormente. Este SGBD brinda grandes potencialidades para el desarrollo de software, pues aparte de ser libre presenta una gran comunidad en la universidad y en el país que aboga por la realización de software que use este gestor por las características antes expuestas.

Este SGBD posee además: **(LINPPC 2010)**

1. Probada fiabilidad e integridad transaccional por defecto (cumple con ACID).
2. Cuidadoso soporte para los estándares SQL.
3. Extensión de tipos y extensión de funciones.
4. No hay límite sobre el tamaño de la columna para soportar lo grandes objetos del SIG.
5. Facilidad de añadir funciones personalizadas.

PostGIS

PostGIS convierte al sistema de administración de bases de datos PostgreSQL en una base de datos espaciales mediante la adición de tres características fundamentales: tipos espacial, que se refiere a tipos de datos que abstraen y encapsulan las estructuras espaciales, tales como los límites y dimensiones; índices espaciales que son los métodos que permiten el acceso rápido y al azar a los subconjuntos de datos; y las funciones espaciales que son las encargadas del análisis de componentes geométricos, la determinación de las relaciones espaciales, y la manipulación de geometrías. **(LINPPC 2010)**

Debido a que está construido sobre PostgreSQL, PostGIS hereda automáticamente importantes características de este gestor así como los estándares abiertos para la aplicación.

PostGIS además de permitir realizar análisis geográfico a través de consultas espaciales, cumple con las siguientes funciones:

1) Conversión: Funciones que convierten entre geometrías y formatos geométricos externos.
2) Gestión: Funciones que administran la información acerca de las tablas espaciales.
3) Recuperación: Funciones de recuperar las propiedades y medidas de una geometría.

4) Comparación: Funciones que comparan dos geometrías respecto a su relación espacial.
5) Generación: Funciones que generan nuevas geometrías a partir de otras.

Todas estas características combinados en PostgreSQL proporcionan una vía de desarrollo viable en la construcción de GeoQ y brindan la posibilidad de agregar nuevos tipos espaciales.

1.6 Conclusiones

En este capítulo se han expuesto un conjunto de conceptos asociados al dominio de las áreas de Arquitectura de Software y Sistemas de Información Geográficos definiendo los principales estilos y patrones arquitectónicos, así como el lenguaje de descripción de la arquitectura. Se describen también algunas metodologías de desarrollo, lenguajes de programación, herramientas CASE y entornos de desarrollo integrados con el propósito de seleccionar el adecuado para el diseño del paquete arquitectónico a plantear. Por ende se concluye la primera de las tareas de la investigación y se obtuvieron las bases para el desarrollo del presente trabajo:

Los estilos y patrones arquitectónicos a usar son: Arquitectura Orientada a Objeto (AOO) y Arquitectura Basada en Componentes (ABC) en ambos casos. De igual forma el lenguaje de representación y modelación de la arquitectura seleccionado es UML y la herramienta CASE Visual Paradigm. Los lenguajes de programación son Phyton, seleccionado para la implementación a lo que integración de plugins se refiere y C++ para el desarrollo del SIG en su totalidad. Entre los entornos de desarrollos a usar están: QT-Creator, poseedor del CMAKE que permite crear aplicaciones multiplataforma y se propone también el entorno de desarrollo multiplataforma Eclipse el cual emplea módulos (plugins) para proporcionar funcionalidades para plataformas de escritorio. Se decide utilizar como gestor de base de datos a PostgreSQL en su versión 8.3 porque además de ser multiplataforma, con uso y distribución gratis, permite la creación de nuevos tipos de datos. Posee una integridad referencial que posibilita un buen grado de seguridad en los datos y una extensión PostGIS que admite el trabajo con objetos geográficos, los cuales junto a los alfanuméricos son los que utilizan los Sistemas de Información Geográfica.

CAPÍTULO 2: DESCRIPCIÓN DE LA BASE ARQUITECTÓNICA.

2.1 Introducción

En este capítulo, se hace una propuesta arquitectónica base que servirá como guía para el desarrollo del paquete de arquitectura de software. Se toman en cuenta las descripciones de las tecnologías y herramientas anteriormente expuestas. Se muestran los atributos de calidad y los requisitos que tienen impacto sobre la solución arquitectónica a plantear y se explica el flujo de procesos además de exponerse las vistas arquitectónicas (4+1 Vistas) que propone RUP representadas con UML.

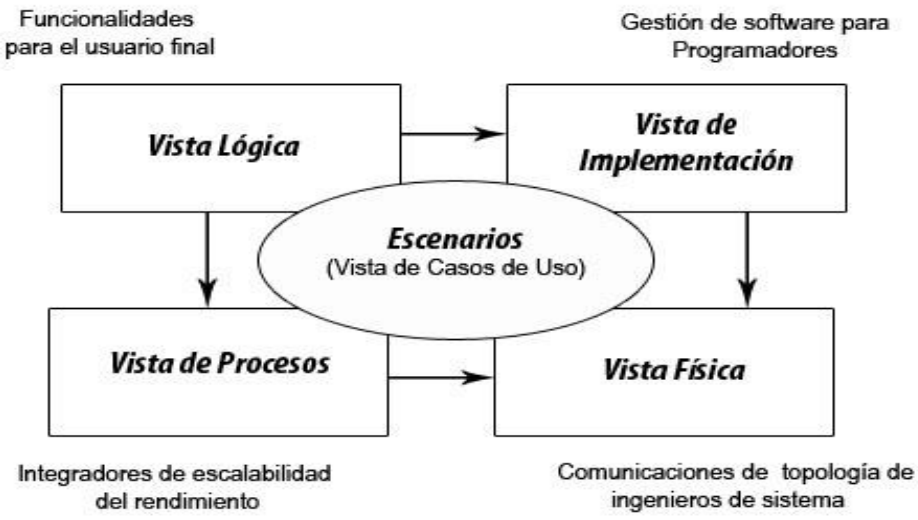

Ilustración 4. Diagrama 4+1 Vistas

2.2 Atributos de calidad y requisitos que tienen impacto sobre la arquitectura

La calidad puede definirse como la conformidad relativa con las especificaciones, el grado en que un producto cumple estas especificaciones o como comúnmente encontrar la satisfacción en un producto cumpliendo todas las expectativas que busca algún cliente.

Pressman la define como: “Concordancia del software producido con los requerimientos explícitamente establecidos, con los estándares de desarrollo prefijados y con los requerimientos implícitos no establecidos formalmente, que desea el usuario.”

La clasificación de calidad que el usuario percibe, es clasificada en los sistemas informáticos como “Atributos Externos de la Calidad”, es decir aquello que se percibe desde el exterior. La presente investigación utilizará los atributos de calidad del modelo ISO/EC 9126 adaptado para

arquitecturas de software para la evaluación del diseño arquitectónico debido a la correspondencia de dichos atributos con las necesidades del sistema. El modelo de calidad ISO 9126-1, identifica varias particularidades básicas de calidad que pueden estar presentes en cualquier producto de software. El estándar provee una descomposición de las características en subcaracterísticas.

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 restablecer 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 módulos 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 elementos reemplazables para cada componente.

Tabla 1. Atributos de calidad adaptados para arquitecturas de software

Se tienen entonces identificados los siguientes atributos que componen la calidad de GeoQ:

Funcionalidad: Se adecúan los diagramas de secuencia en cada iteración de refinamiento de los diagrama de clases y se propone realizar diagramas de secuencia independientes a los métodos

que están sobrecargados. Se asegura la conectividad con subsistemas y sistemas externos a la aplicación a través de la herramientas de integración SWIG Simplified Wrapper and Interface Generator. La seguridad informática, tema de gran importancia ya que es fundamental mantener la protección del sistema y evitar que sea vulnerable a robos de información o a accesos no autorizados, se garantiza haciendo uso del RBAC (Role Based Access Control) como una propuesta que reúne las principales bondades de los tipos de control de acceso existentes (Discretionary Access Control (DAC) y Mandatory Access Control (MAC)). Dicha seguridad se garantiza a través de un subsistema de gestión de la seguridad.

Confiabilidad: Se previenen los fallos, tanto de hardware mediante la utilización de componentes fiables, técnicas rigurosas de montaje de subsistemas y el aislamiento de hardware para protegerlo de interferencias o fallas esperadas; como de software realizando la especificación de requisitos, los métodos de diseño comprobados, utilizando lenguajes con abstracción de datos y modularidad, y utilización herramientas CASE adecuadas para gestionar los componentes. La separación de los servidores del cliente proporciona también un alto grado de confiabilidad puesto que las fallas en uno no repercuten directamente en el otro, además de la existencia de un servidor de respaldo. Se realizan pruebas que garantizan la veracidad y robustez del código. Con la correcta aplicación de este atributo se satisfacen principalmente tres necesidades: que el sistema cumpla con las especificaciones, es decir, que sea consistente con éstas; que tenga la capacidad de reaccionar bien ante situaciones excepcionales (tolerancia a fallas) y recuperarse en caso de fallas en el menor tiempo posible; y la capacidad de estar operativo el mayor tiempo posible. Todo proporciona al sistema una mayor confiabilidad.

Eficiencia: En este sentido a la aplicación le corresponde permitir que la información almacenada pueda ser consultada y actualizada permanentemente, sin que se afecte el tiempo de respuesta. El sistema debe poseer la capacidad de dar respuesta al acceso de cada usuario, con un tiempo de respuesta aceptable y uniforme, en la medida de las posibilidades tecnológicas que tengan el grupo de desarrollo. Aunque una práctica muy común en los desarrolladores es la optimización excesiva, lo importante es que el SIG sea implementado manteniendo un balance adecuado entre eficiencia y corrección. Obteniendo una buena eficiencia se logra que el software tenga la capacidad de hacer buen uso de los recursos que manipula, teniendo en cuenta principalmente los recursos de hardware, tiempo de procesador y ancho de banda.

Mantenibilidad: Es la capacidad del producto de software para ser modificado. Las modificaciones pueden incluir correcciones, mejoras o adaptaciones del software a los cambios del entorno y en los requisitos y especificaciones funcionales. Dado el uso de la arquitectura orientada a objeto la cual permite que los objetos estén débilmente acoplados, la modificación de ellos puede efectuarse sin comprometer a los otros, lo que, al igual que la arquitectura basada en

componentes como implementan interfaces bien definidas, permite el desarrollo en el sistema sin impactar otras partes del mismo.

Portabilidad: El sistema se implementará sobre tecnología libre y sobre un entorno de desarrollo que posibilite generar aplicaciones que puedan ser instaladas y ejecutadas en diferentes sistemas operativos. Se divide además los servidores proveedores de datos del sistema que los consume de la aplicación cliente, para facilitar la movilidad de este último y posibilitar un balance de carga adecuado para el Sistema, es decir que el SIG pueda ser instalado en cualquier ordenador personal. Como se define el uso de la arquitectura basada en componentes, se hace posible que la facilidad de instalación que permite reemplazar una nueva versión del sistema por la existente sin impacto en el mismo o en sus componentes. Estos elementos posibilitan la instalabilidad, remplazabilidad, adaptabilidad y coexistencia del sistema.

Además de estos atributos, la arquitectura de software responde a requisitos funcionales y no funcionales que debe cumplir el sistema, por lo que es necesario establecer cuáles son los principales requisitos que debe cumplir un Sistema de Información Geográfica.

Los requisitos no funcionales son propiedades o cualidades que se corresponden con las características del sistema a las que se aplican de manera general como un todo, más que a cualidades particulares del mismo. Son clasificados en dos grupos: Software y Hardware.

Requisitos de software:

1. Cliente
    a. Sistema de Información Geográfica GeoQ.
    b. Sistema Operativo Windows XP o GNU/Linux Ubuntu.
2. Servidor de datos
    a. Sistema Operativo Windows XP o GNU/Linux Ubuntu.
    b. Gestor de base de datos PostgreSQL v8.3 o superior con extensión PostGIS para la gestión de datos espaciales.

Requisitos de hardware

1. Cliente
    a. Al menos 512 MB de memoria RAM.
    b. Procesador 2.8 GHz o superior.
    c. Se requiere al menos 80 de disco duro.
    d. Procesador Pentium IV o superior.
    e. Tarjeta de red.

2. Servidores de datos
    a. Se requiere al menos 1GB de memoria RAM.
    b. Se requiere al menos 160GB de disco duro en cada servidor.
    c. Procesador 2.2 GHz o superior.
    d. Tecnología de respaldo de datos históricos.
    e. Procesador Pentium IV o superior.
    f. Tarjeta de red.

Los requisitos funcionales, los cuales son capacidades o condiciones que el sistema debe cumplir, dan la idea de cómo funciona la aplicación y por tanto el comportamiento de las clases lo que da la posibilidad de indicar cuáles son los posibles estilos a utilizar a la hora de la creación de un SIG. Los requisitos funcionales que son de gran interés para la arquitectura de software son:

1. Requisitos funcionales de las Capas.
    a. Añadir Capa Vectorial
    b. Añadir Capa Ráster
    c. Añadir Capa PostGIS
    d. Añadir Capa SpatiaLite
    e. Añadir Capa WMS
    f. Nueva Capa de archivo .shape

2. Requisitos funcionales de Navegación.
    a. Acercar/Alejar Zoom
    b. Zoom general
    c. Zoom a la selección
    d. Zoom a la capa
    e. Zoom siguiente/anterior

3. Requisitos funcionales de atributos.
    a. Regla
    b. Medir áreas
    c. Medir ángulo

4. Requisitos Funcionales de Digitalización/Edición (Avanzada).
    a. Gestionar objeto espacial
        - Agregar objeto espacial (Añadir punto, Añadir línea, Añadir polígono)
        - Modificar objeto espacial (Simplificar, Remodelar, Dividir, Combinar)
        - Cortar, Copiar y Pegar objeto espacial.

2.3 Integración de sistemas

Hoy en día la integración entre aplicaciones y procesos es una prioridad para las empresas. Los usuarios esperan el acceso rápido a todas las funciones del negocio que una empresa puede ofrecer, independientemente de las funcionalidades que puedan residir en este sistema, lo cual se puede lograr a través de la integración de negocios. Una solución de integración se logra mediante el uso de herramientas que proporcionen la vía para el transporte de los datos, la transformación de estos, y el enrutamiento de los mismos con el objetivo de facilitar la comunicación. Todo ello posibilita que, como en el diseño orientado a objeto, la solución para la integración permita la modificación y el trabajo con distintos objetos sin que uno afecte al otro. Surge entonces la arquitectura de soluciones de integración, la cual aún refiere una ardua tarea y no posee un diseño o categorización general sino que cada una de las que se diseñan están orientadas a la solución para la comunicación con los demás sistemas. Este proceso de integración se realiza con el fin de apoyar los procesos de negocios comunes y el intercambio de datos entre aplicaciones. Debe propiciar que sea eficiente, confiable y seguro entre los múltiples sistemas.

Existen distintos mecanismos de integración: Manual, Dentro de la aplicación y Capa de integración. El primero de ellos consiste en que el usuario sea el que decida el paso a seguir en cada acción, aunque es bastante sencillo de implementar es el menos eficiente. El segundo radica en implementar una solución de integración automática pero dentro de una de las aplicaciones que se han de integrar, aunque es una solución mejor que la anterior mantiene enlazado el proceso de integración al sistema. Y el último propone la realización de una nueva capa que gestione la integración de todas las aplicaciones, siendo esta la mejor propuesta a implementar por la independencia que tiene de los sistemas y los usuarios. De ellos el mecanismo “capa de integración” se puede dividir o clasificar en tres tipos **(DAVID TROWBRIDGE)**:

Portal de Integración: Conecta varios sistemas para proveer al usuario final de una única vista. Ello permite que ningún usuario tenga acceso a los sistemas independientes. En cambio el usuario recibe una vista comprendida por todos los sistemas en un formato visual consistente.

Entidad de Integración: Proporciona una representación lógica de las entidades de datos unificadas a través de múltiples almacenes de datos. Las aplicaciones pueden interactuar con esta representación como si se tratara de una fuente de datos única.

Proceso de Integración: Se centra en la organización de las interacciones entre múltiples sistemas. Los procesos automatizados de un negocio así como el procesamiento directo a menudo impulsan nuevas iniciativas de integración. Se recomienda modelar estos procesos fuera

de las aplicaciones para evitar el acoplamiento entre los sistemas entre sí y esto se puede lograr mediante la adición de una capa de proceso de integración que maneje la interacción entre las aplicaciones. Esta capa posibilitará el seguimiento del estado de cada instancia de los procesos del negocio y la presentación de informes centralizados.

Los tres tipos de integración de capa no son mutuamente excluyentes. Una solución podría utilizar Portal en algunas partes y Proceso para las demás. Las funciones de gestión de informes se pueden lograr mediante el uso de la Entidad de Agregación. Todos los métodos se pueden utilizar en paralelo.

Independientemente de la gran variedad de protocolos y soluciones informáticas existentes para la integración, tales como: Component Object Model (COM), Common Object Request Broker Architecture (CORBA), Protocol Buffers, Interactive Connectivity Establishment (ICE), Distributed Component Object Model (DCOM), Simple Object Access Protocol, Representational State Transfer (REST); se selecciona para las comunicaciones con otros sistemas la herramienta SWIG (Simplified Wrapper and Interface Generator), la cual simplifica la tarea de interconectar diferentes idiomas a C y C++. La misma ofrece una gran variedad de características de personalización que permiten cambiar una amplia gama de aspectos de los enlaces entre diferentes lenguajes, pues está diseñado bajo el concepto de que todas las aplicaciones son diferentes. **(SWIG 2011)**

Con esta potente herramienta se pueden conectar los programas escritos en C y C++ con los implementados en lenguajes de programación como Perl, Python, Ruby y Tcl. Por sus características puede ser usada para la construcción de potentes programas en C o C++, para la creación y depuración rápida de prototipos y para la integración de sistemas. Esta última la gestiona mediante la adopción de las declaraciones que se encuentran en la cabecera de los archivos de C / C++, utilizándolo para generar el código del wrapper[7] para la comunicación de los otros lenguajes con el código escrito en C++ o C de la solución existente. Todo ello posibilitará la integración con otros sistemas o negocios sin importar con que lenguaje de programación fue implementado. SWIG es comparado algunas veces con los compiladores de lenguaje de definición de interfaz (IDL), como los que se encuentran en sistemas tales como CORBA y COM. Aunque existen algunas similitudes, toda la cuestión de SWIG es hacer el compilador, por lo que no es necesario agregar una capa adicional de especificaciones IDL para la aplicación. En todo caso, es mucho más que una herramienta de desarrollo rápido de aplicaciones y de creación de prototipos. **(AUTORES)**

[7] Un componente de software que encapsula componentes del sistema (un procedimiento, un programa, un archivo, una API) con el fin de transformar su interrelación con su entorno. Traduce una interfaz para una clase en una interfaz compatible.

2.4 Vista de Casos de Uso

Dentro de las 4+1 vistas es precisamente la vista de casos de uso la base para las demás vistas. Está formada por los escenarios y casos de uso que son arquitectónicamente significativos los cuales no son más que las necesidades funcionales que cubre el sistema y que a través de ellos se podrá validar la arquitectura propuesta. Por ello la misma proporciona una base para planificar el contenido técnico de cada iteración.

Luego de realizado el diagrama de casos de uso del sistema, se seleccionan como casos de uso arquitectónicamente significativos:

1. Agregar capa
2. Conmutar edición
3. Gestionar objeto espacial
4. Realizar zoom
5. Realizar mediciones
6. Gestionar seguridad
7. Autenticar usuario
8. Administrar complementos
9. Integración

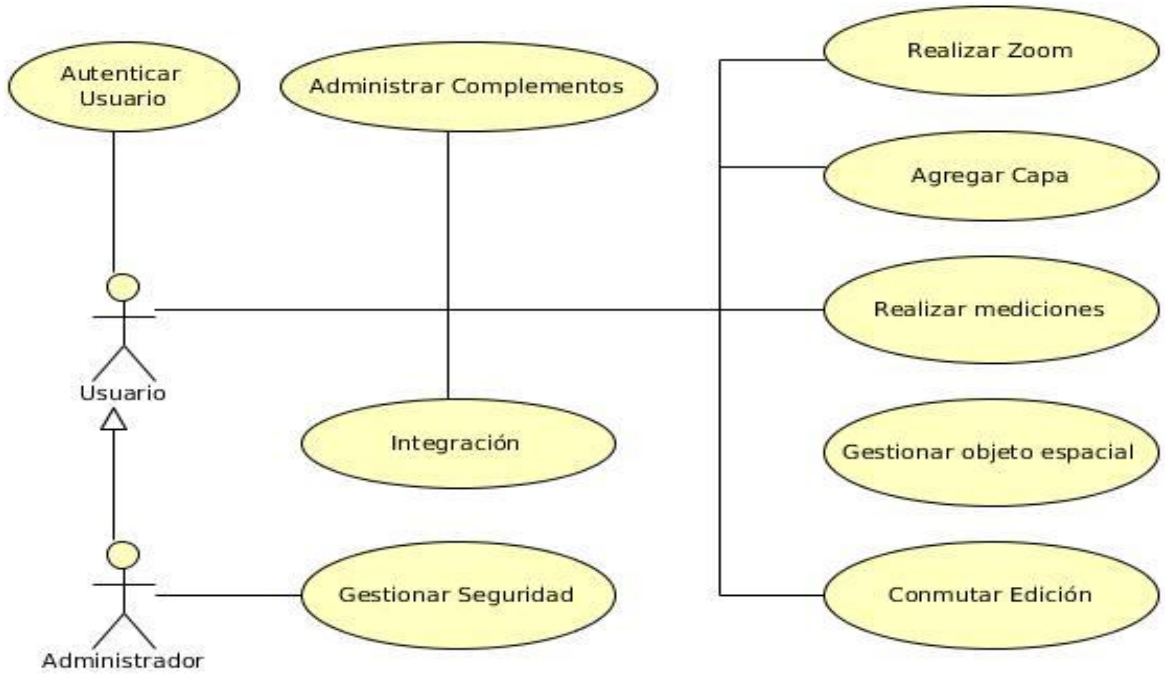

Ilustración 5. Diagrama de Casos de Uso arquitectónicamente significativos

Caso de Uso Agregar Capa: La precondición para la inicialización del caso de uso es que el usuario debe de estar previamente autenticado por el sistema de seguridad. El caso de uso posibilita añadir una capa al sistema según la opción seleccionada:

Añadir Capa WMS: El caso de uso permite cargar mapas de datos espaciales referidos de forma dinámica a partir de información geográfica haciendo uso del Web Map Service (WMS).

Añadir Capa Ráster: El caso de uso permite buscar y cargar capas ráster que pueden ser de varios tipos de extensiones GDA.

Añadir Capa Vectorial: El caso de uso admite buscar y cargar datos vectoriales en distintos formatos, incluyendo aquellos soportados por el proveedor de datos de la biblioteca OGR. El formato de archivo vectorial estándar usado en GeoQ es el archivo shape (.shp, .dbf y .shx) de ESRI.

Añadir Capa PostGIS: El caso de uso permite agregar capas de tipo de datos postgis las cuales se almacenan en una base de datos PostgreSQL. Esta funcionalidad permite realizar una conexión entre la base datos del usuario con GeoQ cuyas ventajas son indexamiento espacial, filtrado y las capacidades de consulta que provee.

Añadir Capa SpatiaLite: El caso de uso permite abrir y cargar bases de datos de tipo SQLite (.sqlite), las cuales son bases de datos geográficos con información espacial adoptando lineamientos del OGC.

Nueva Capa: El caso de uso permite crear capas shape y spatialite, ya sea de tipo puntual, lineal o poligonal o de forma múltiple según el tipo de capa a crear.

Caso de Uso Conmutar Edición: Todas las sesiones de edición comienzan al seleccionar la acción “Conmutar edición”. Una vez que la capa está en modo de edición, los marcadores aparecerán en los vértices y estarán disponibles botones adicionales en la barra de herramientas. Para guardar los cambios recientes se debe desactivar la acción y ello debe permitir además confirmar que los conjuntos de datos pueden aceptar todos los cambios.

Caso de Uso Gestionar objeto espacial: La precondición para la inicialización del caso de uso es que antes se debe haber accionado “Conmutar edición”. El caso de uso consiste en la selección de una de las funciones de Digitalización/Edición (Avanzada), las cuales permiten añadir objetos espaciales (punto, línea, polígono), modificarlos (Simplificar, Remodelar, Dividir, Combinar) ó Cortar, Copiar y Pegar estos objetos espaciales.

Caso de Uso Realizar Zoom: La precondición para la inicialización del caso de uso es que el usuario debe de estar previamente autenticado por el sistema de seguridad. El caso de uso consiste en la selección de una de las acciones de Realizar Zoom y según la acción seleccionada es que procede a realizar su función de navegación, las cuales pueden ser de acercamiento, centrar el mapa entre otras según la acción que necesite realizar el usuario. Debe tener cargada y seleccionada una capa. Los zoom a realizar son:

Acercar Zoom

Alejar Zoom

Zoom General

Zoom a la Selección

Zoom Anterior

Zoom Siguiente

Zoom a la carpeta

Caso de Uso Realizar mediciones: Posibilita la medición de obras proyectadas dentro de un único sistema de coordenadas. Sus funcionalidades son: medir distancias entre puntos reales determinados de acuerdo a un elipsoide definido, “Medir área” en grado cuadrado o “Medir ángulo” en grados.

Caso de Uso Administrar Complementos: GeoQ será diseñado con una arquitectura de complementos lo que permite que se añadan nuevas funciones. De ahí que muchas de estas funciones estén en realidad implementadas como complementos, divididos en dos tipos: integrados o aportados por usuarios. El caso de uso permite seleccionar los complementos a usar en la aplicación.

Caso de Uso Gestionar Seguridad: La precondición para la inicialización del caso de uso es que el usuario debe de estar previamente autenticado por el sistema de seguridad. El caso de uso permite gestionar los datos de un usuario o un individuo que desee acceder a la aplicación. Permitirá al administrador de seguridad crear, modificar o eliminar los usuario, sus roles y niveles de acceso al sistema; y en el mismo se gestionará toda la seguridad para la autorización de los usuarios que deseen interactuar con la plataforma.

Caso de Uso Autenticar Usuario: El caso de uso consiste en permitir a un usuario autenticarse para acceder a la aplicación. Para ello debe introducir nombre de usuario y contraseña y el sistema validará si el mismo tiene acceso a la plataforma o no. Este caso de uso constituirá la primera interacción de todo usuario con el sistema a implementar.

Caso de Uso Integración: El caso de uso permite la integración entre el sistema de información geográfica y otra aplicación. Ello se realizará mediante la herramienta de integración SWIG la cual permite la comunicación entre varios lenguajes y C y C++ lo que posibilita el entendimiento de sistemas informáticos externos con el SIG GeoQ.

2.5 Vista Lógica

Esta vista soporta el análisis y la especificación de los requisitos funcionales que son los que el sistema suministra en términos de prestaciones a los usuarios finales. Este sistema se modela con un conjunto de abstracciones clave tomadas del dominio del problema representadas a través de objetos o clases. Todo ello se traduce en que la vista lógica es la responsable de describir las clases más importantes que formarán parte del ciclo de desarrollo además de los paquetes del sistema y las relaciones existentes entre ellos. La notación a usar es UML y dentro de esta, diagramas de clases y paquetes. El estilo es el Orientado a Objetos.

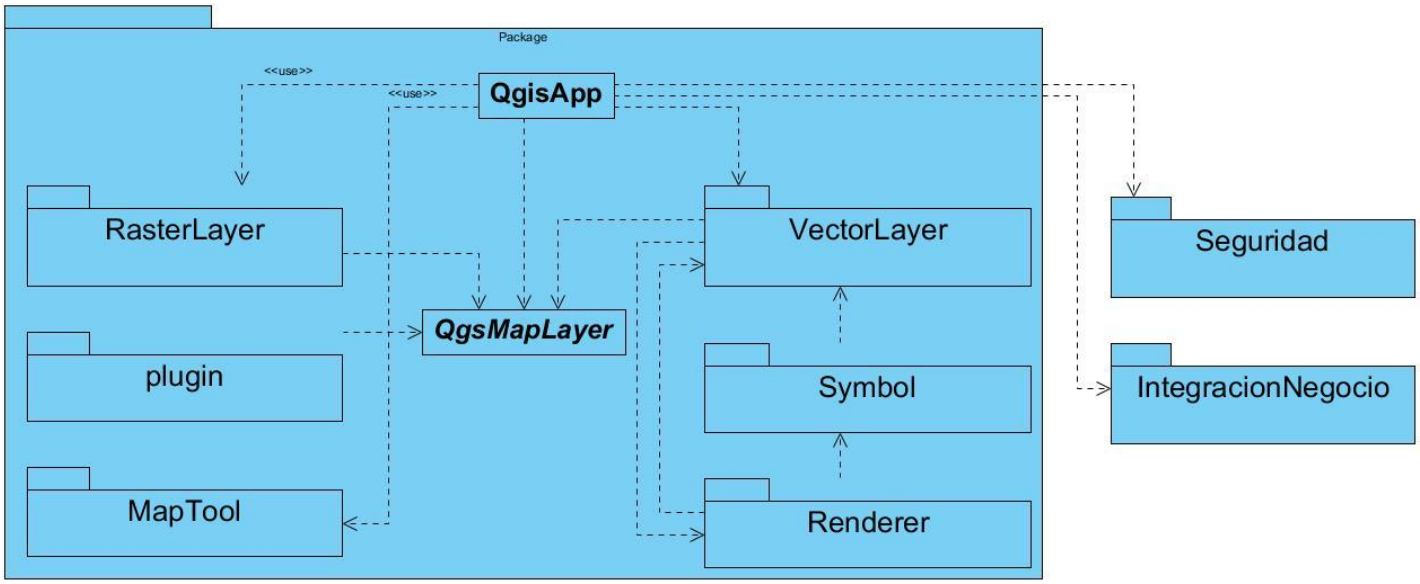

Ilustración 6. Vista lógica general del sistema de información geográfica GeoQ

El diagrama anterior será descrito por separado con el objetivo de facilitar la comprensión de cada uno de los paquetes y subsistemas representados.

GeoQ

Comprende el sistema de información geográfica en su concepción inicial, solamente para la gestión de mapas. Está compuesto por los siguientes paquetes:

Integración de Plugins

Conserva las clases que posibilitan la integración de plugins al sistema para ampliar sus funcionalidades.

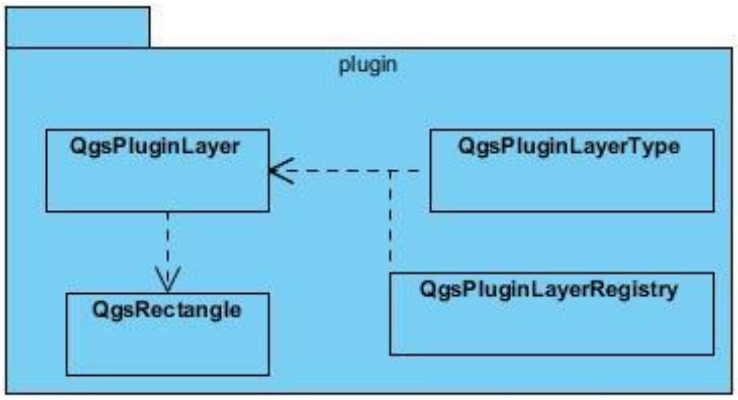

Ilustración 7. Vista lógica del paquete plugin.

Symbol

Tiene las clases contenedoras de la información para realizar el render a los símbolos y las de representar la leyenda del mapa. Contiene además las que encapsulan las configuraciones y clasificaciones del dibujo del mapa.

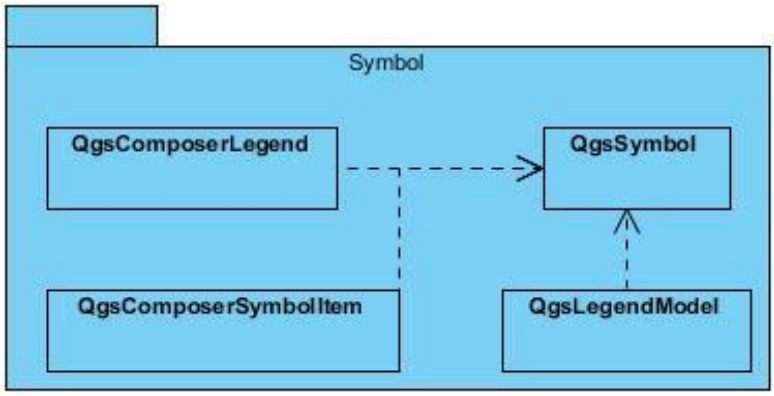

Ilustración 8. Vista lógica del paquete Symbol.

VectorLayer

Posee las clases que manejan los datos de las capas vectoriales así como los del vector, de los diagramas y las etiquetas. Estas clases implementan las formas de mostrar capas vectoriales. Su clase principal QgsVectorLayer se puede utilizar con cualquier almacén de datos para los que un plugin esté disponible.

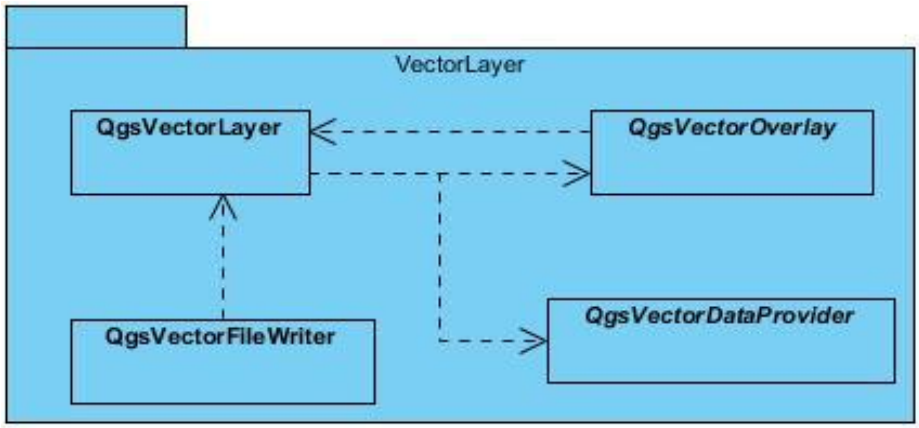

Ilustración 9. Vista lógica del paquete VectorLayer.

Renderer

Aguarda las clases necesarias para el render de una capa vectorial y dibujarla en el mapa, así como las que poseen la información sobre la operación de un render y las que permiten comunicarse con los plugins del render. Su clase principal posee toda la información necesaria para elaborar el contenido de una capa vectorial a una vista del mapa. Cada una de las propiedades de dibujo de la capa vectorial es pasada por el render.

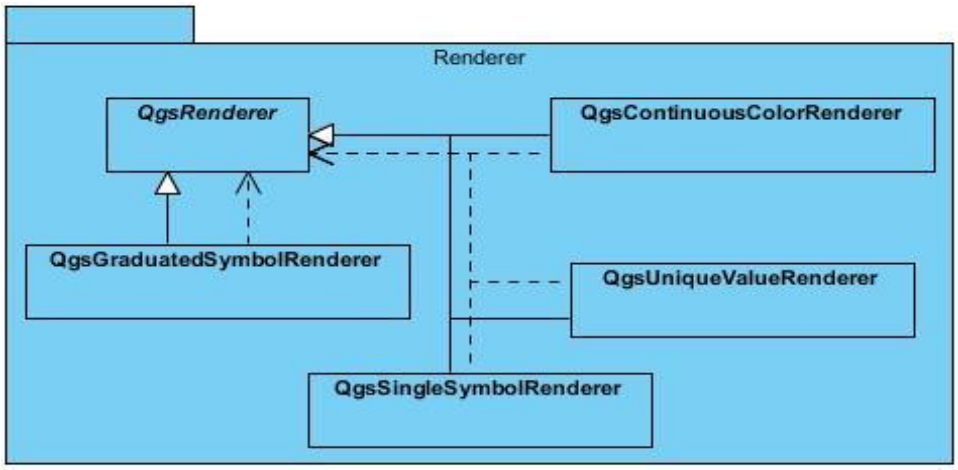

Ilustración 10. Vista lógica del paquete Renderer.

RasterLayer

Asume las clases que se encargan de las capas ráster, las cuales proporcionan a GeoQ la capacidad de representar conjuntos de datos ráster en el mapcanvas y contener su información. La clase qgsrasterlayer, su clase principal, hace uso de GDAL y apoya por lo tanto cualquier formato GDAL compatible. El constructor intenta deducir qué tipo de archivo se abre, no en términos del formato de archivo sino más bien en términos del tipo de imagen.

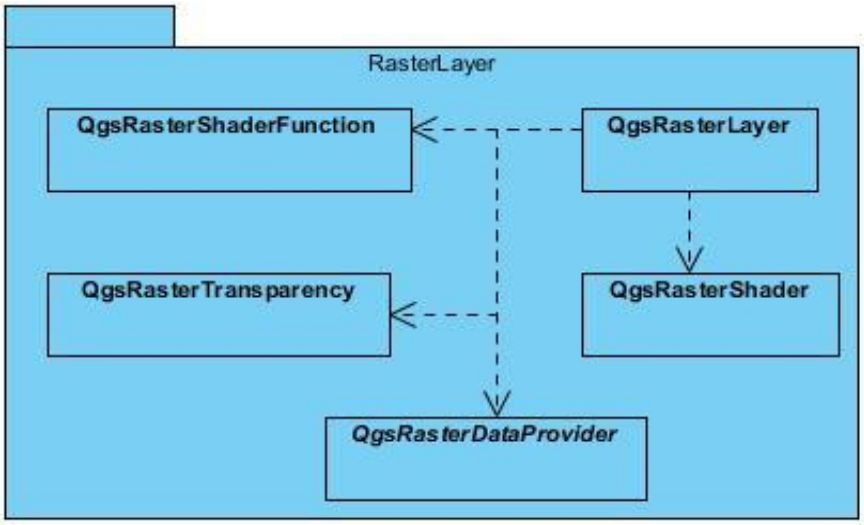

Ilustración 11. Vista lógica del paquete RasterLayer.

MapTool

Contiene las clases de las herramientas interactivas de usuarios para la manipulación del mapcanvas.

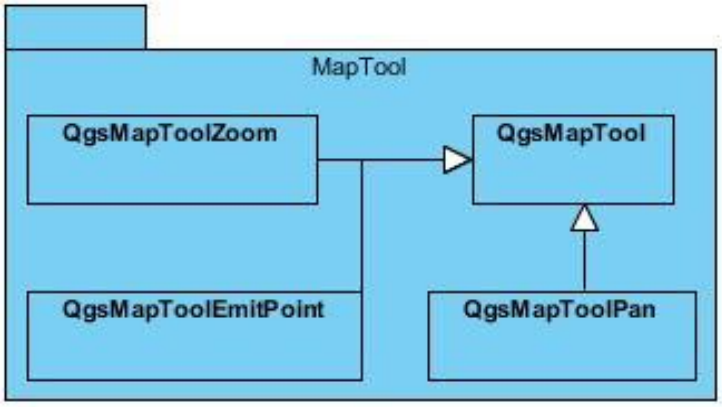

Ilustración 12. Vista lógica del paquete MapTool.

Seguridad

Subsistema que gestiona la seguridad de GeoQ. Es independiente completamente del negocio que maneja el SIG. Tiene las clases que gestionan la identificación, autenticación y autorización de un usuario al sistema.

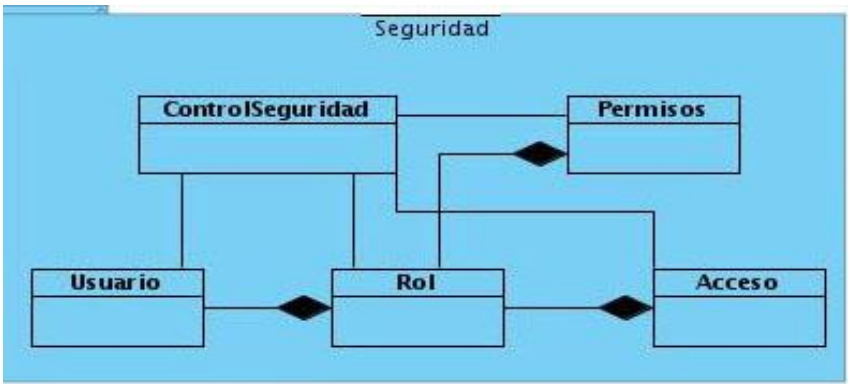

Ilustración 13. Vista lógica del subsistema de Seguridad.

Integración

Subsistema que contiene las clases que dan cumplimiento al patrón de integración para la comunicación con otros negocios a través de la herramienta SWIG. Es independiente completamente del negocio que maneja el SIG.

Las dos clases que se muestran en la vista lógica general (QgisApp y qgsMapLayer) son las principales clases con las que se gestiona el sistema de información geográfica GeoQ. La primera se encarga de construir la interfaz del sistema con cada una de las funcionalidades que el mismo posea y la segunda constituye la clase base para los tipos de capas de mapas (Vectorial y Ráster).

2.6 Vista de Implementación

Esta vista es la que describe cómo serán implementados cada uno de los componentes los cuales se agrupan en subsistemas. Detalla el mapeo de los paquetes y clases de la vista lógica a subsistemas y componentes físicos.

“La vista de implementación muestra el empaquetado físico de las partes reutilizables del sistema en unidades sustituibles, llamadas componentes. Una vista de implementación muestra los elementos físicos del sistema mediante componentes, así como sus interfaces y dependencias entre componentes. Los componentes son piezas reutilizables de alto nivel a partir de las cuales se pueden construir los sistemas.” **(DUEÑAS 2009)**

En esta vista se ilustran: datos, archivos, ejecutables, código fuente, directorios entre otros; los cuales son representados a través de diagramas de componentes. Muestra la organización y las dependencias entre un conjunto de mecanismos, lo cual no necesariamente deben de ser todos los del sistema sino que se pueden realizar por partes.

"El diagrama de componentes describe la descomposición física del sistema de software en componentes, a efectos de construcción y funcionamiento. La descomposición del diagrama de componentes se realiza en términos de componentes y de relaciones entre los mismos. Los componentes identifican objetos físicos que hay en tiempos de ejecución, de compilación o de desarrollo y tienen identidad propia con una interfaz bien definida. Los componentes incluyen código en cualquiera de sus formatos, DLL, Active X, bases de datos, etc. Cada componente incorpora la implementación de ciertas clases del diseño del sistema." **(AUTORES)**

Cada uno de los paquetes que se representan en el siguiente diagrama encapsulan varios componentes que se interrelacionan para concebir el correcto funcionamiento del sistema GeoQ, estableciendo además las dependencias representadas. Cada paquete será explicado por separado con el objetivo de obtener una mejor comprensión del mismo.

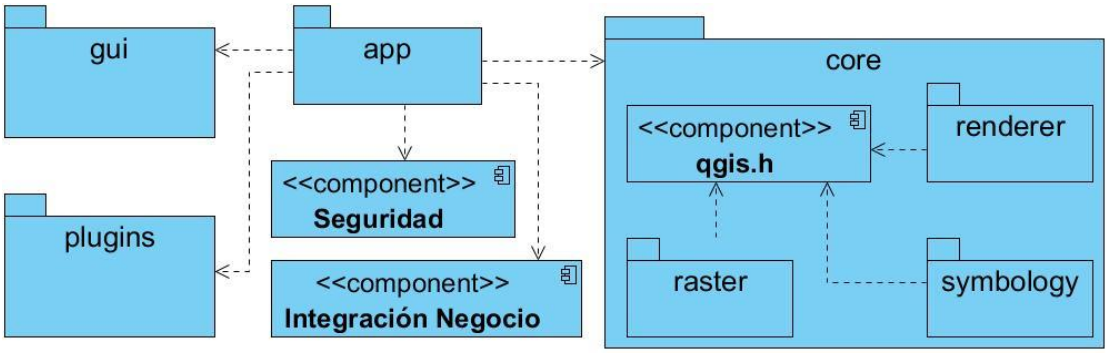

Ilustración 14. Diagrama de componentes de GeoQ

Paquete App

En App se encontrarán las clases que le dan estructura a GeoQ, de ahí que se encuentre en el mismo la clase qgisapp.cpp la cual tiene la responsabilidad de construir la ventana principal del sistema tomando las funcionalidades de los demás componentes de implementación.

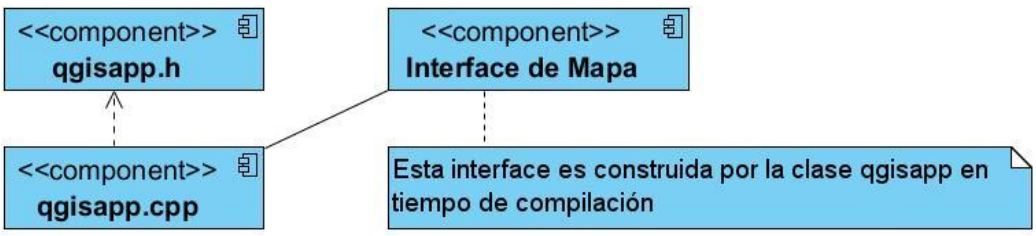

Ilustración 15. Paquete App del diagrama de componentes de GeoQ.

Paquete GUI

El paquete gui contará con las clases y componentes que den apoyo a la implementación del SIG GeoQ, entre ellas la herramientas de mapas, o como se le llama al componente, maptool. Las herramientas de mapas o maptools no son más que un conjunto de herramientas interactivas de usuarios para manipular el mapcanvas[8]. El componente maptool será la clase base para todas las herramientas de mapa a implementar.

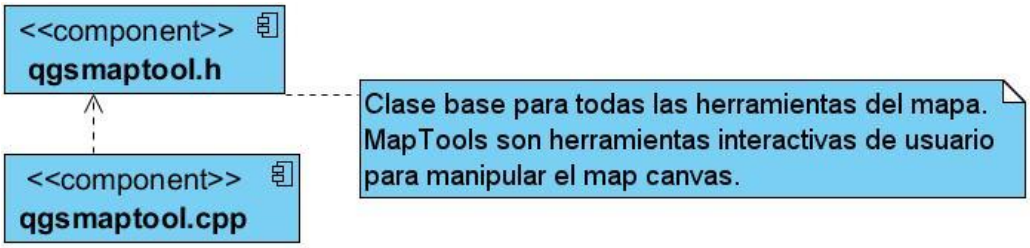

Ilustración 16. Paquete GUI del diagrama de componentes de GeoQ.

Paquete plugin

Este paquete contiene la implementación de cada uno de los plugin que han de incrementar las funcionalidades del SIG, y para ello se establece una clase base qgisplugin.h de la que heredan los plugins a desarrollar para GeoQ. Se deben implementar además otras clases que posibiliten la comunicación entre estos plugins y la aplicación pues estos serán implementados principalmente en phyton y el SIG en C++, pero dichas clases se implementan en otro paquete, phyton.

Componente Integración Negocio

Representa el conjunto de clases en base a las funcionalidades que se deben implementar para lograr que el SIG GeoQ pueda reconocer la existencia de otros negocios o sistemas informáticos y comunicarse con ellos mediante protocolos o soluciones de integración.

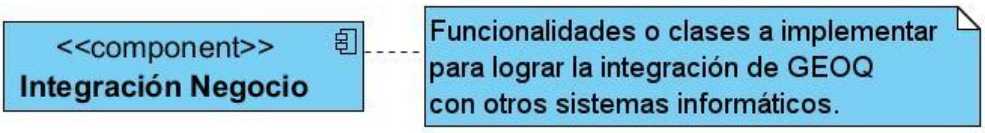

Ilustración 17. Componente Integración Negocio del diagrama de componentes de GeoQ.

[8] Dibujo que se puede realizar sobre el mapa. Corresponde a una capa del mapa sobre la cual pueden trazarse polígonos.

Componente Seguridad

Constituye la implementación de un subsistema de seguridad que pueda comunicarse con GeoQ para impedir el acceso no autorizado a la aplicación.

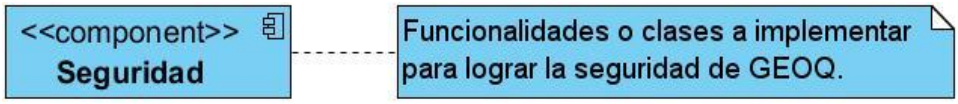

Ilustración 18. Componente Seguridad del diagrama de componentes de GeoQ.

Paquete core

En este directorio se encontrarán los principales componentes de implementación de GeoQ y las clases del núcleo de este SIG.

qgis: clase que proporciona constantes globales para su uso en toda la aplicación.

qgsmaplayer: Esta es la clase base para todos los tipos de capa de mapa (vectorial, ráster).

qgsvectorlayer: Capas vectoriales respaldado por un proveedor de datos.

Este paquete cuenta además con otros paquetes para su organización:

ráster: paquete donde se encapsulan las clases correspondientes al manejo de las capas ráster. El mismo posee la clase qgsrasterlayer la cual proporciona a QGIS la capacidad de representar conjuntos de datos ráster en el mapcanvas. Esta clase hace uso de GDAL para la entrada y salida de los datos, y soporta por ende cualquier formato compatible con GDAL.

renderer: paquete donde se encontrarán las clases que procesan toda la información necesaria de una capa vectorial para elaborar el contenido de una vista del mapa (pintar en el mapcanvas). En el mismo estará ubicada, entre otras, la clase qgsrenderer que servirá como clase base para el render[9].

symbology: paquete que poseerá las clases que encapsulan las configuraciones para el dibujo y las clasificaciones. Contiene como clase principal a qgssymbol.

[9] Término usado para referirse al proceso de generar una imagen a partir de un modelo.

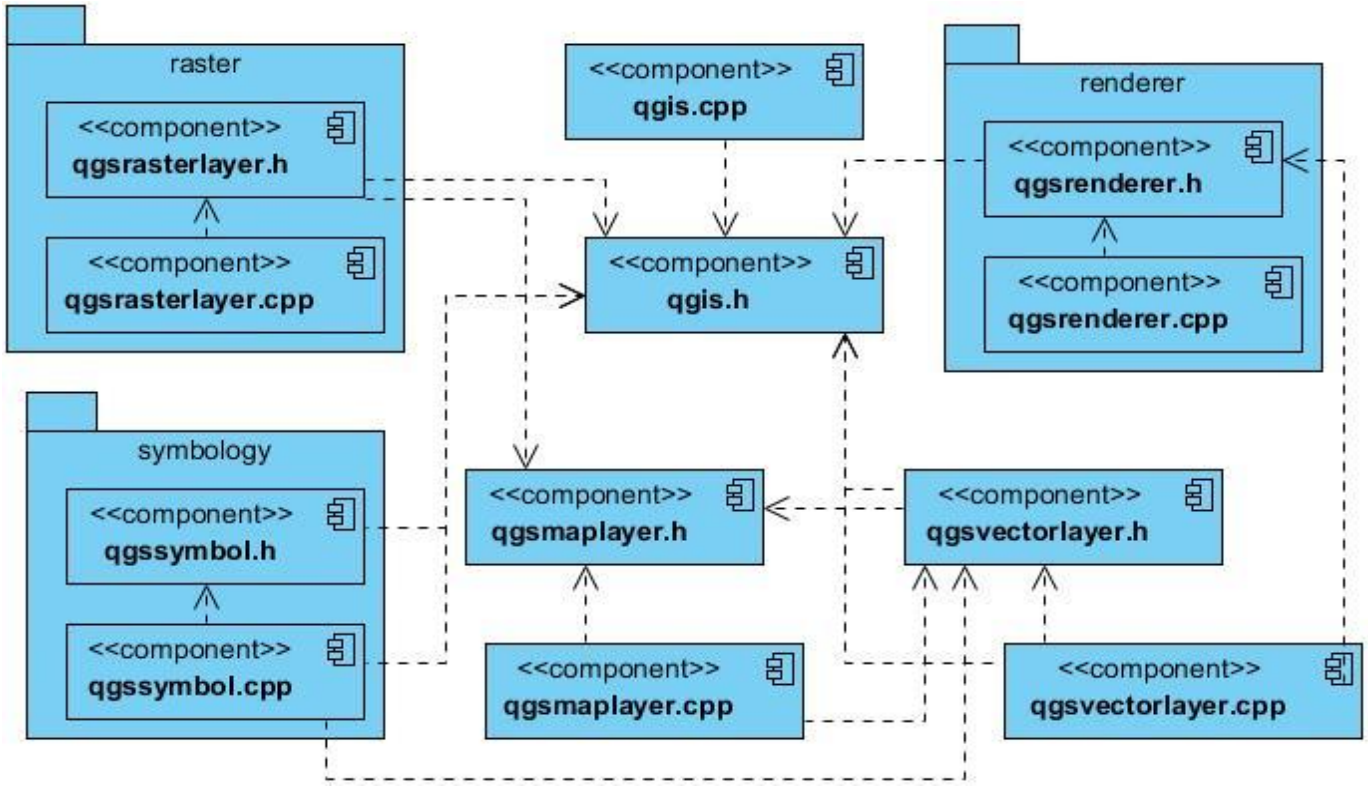

Ilustración 19. Paquete core del diagrama de componentes de GeoQ.

2.7 Vista de Procesos

En esta vista se realiza la descripción de las tareas en función de procesos e hilos, interacciones y configuraciones. Se representa mediante uno o varios diagramas de clases.

Solamente se realiza si la aplicación debe soportar un acceso concurrente, por lo que para el caso de GeoQ no es necesaria, puesto que el SIG es un sistema de escritorio el cual estará en ejecución en cada una de las máquinas que ha de utilizarse y con un solo usuario autenticado y haciendo uso de el.

2.8 Vista de Despliegue

La vista de despliegue suministra una base para la comprensión de la distribución física del sistema y cada uno de sus módulos a través de nodos y la asignación de esos componentes ejecutables a estos nodos. Se satisfacen además parte de los requisitos no funcionales de hardware y software y se especifican los protocolos de comunicación entre estos.

“Un diagrama de despliegue es un grafo de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes de software, objetos, procesos. En general un nodo será una unidad de computación de algún tipo, desde un sensor a un mainframe. Las

instancias de componentes software pueden estar unidas por relaciones de dependencia, posiblemente a interfaces (ya que un componente puede tener más de una interfaz).” **(MARCA HUALLPARA HUGO MICHAEL)**

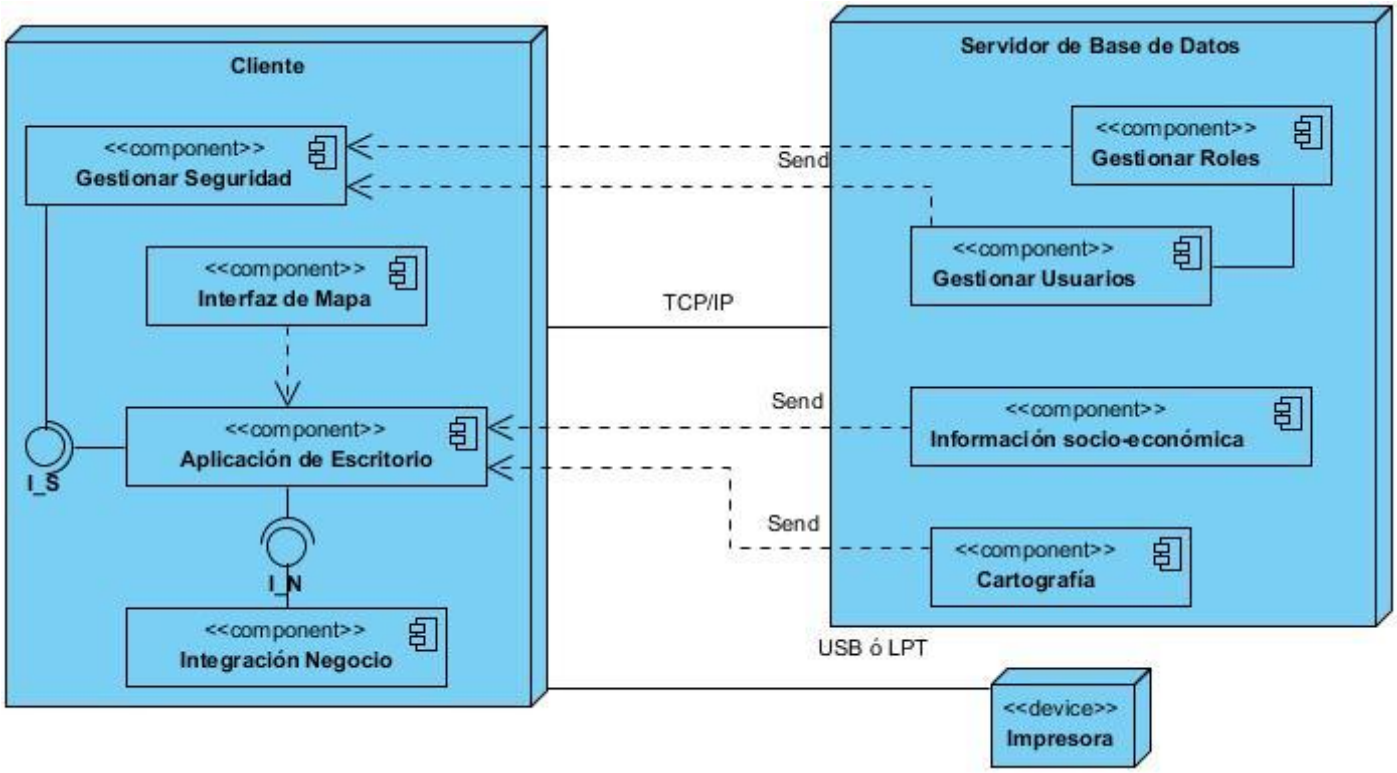

Ilustración 20. Diagrama de despliegue de GeoQ

Nodo Servidor de Base de Datos: Servidor donde estarán alojadas cada una de las bases de datos necesarias para la correcta ejecución y funcionamiento del SIG GeoQ.

Base de Datos de Seguridad: Donde van a estar centralizados los datos de los usuarios, así como sus roles y permisos que le corresponden según el rol que posea. Estos datos serán gestionados por el personal autorizado al acceso a ellos (Administrador o usuarios con privilegios).

Base de Datos Socioeconómico: Está concentrada la información socioeconómica o no espacial del sistema. Dicha información mostrará al usuario datos de una determinada región del mapa.

Base de Datos Espaciales: Se almacena toda la información espacial con la que trabajará la aplicación; tanto los datos ráster como vectoriales.

Nodo Cliente: Computadora que posee la aplicación cliente (GeoQ) sobre la cual se representará todo el trabajo con los mapas así como la información que estos puedan brindar.

Dispositivo Impresora: Este dispositivo estará a disposición de la estación de trabajo con la aplicación cliente para satisfacer los requisitos de los clientes de impresión de mapas con todas las configuraciones previamente realizadas.

Protocolos

La comunicación entre los nodos se define a través del protocolo TCP/IP, el cual no es un único protocolo sino un conjunto de protocolos que cubren los distintos niveles del modelo OSI. Los dos protocolos más importantes son el Protocolo de Control de Transmisión (TCP) y el Protocolo de Internet (IP), que son los que dan nombre al conjunto. El mismo es compatible con cualquier sistema operativo y cualquier tipo de hardware. Una de las funciones y ventajas principales del TCP/IP es proporcionar una abstracción del medio, de forma que sea posible el intercambio de información entre medios diferentes y tecnologías que inicialmente son incompatibles.

Para conectar el dispositivo impresora con el nodo cliente se define USB o TSP, según el tipo de PC-Cliente e Impresora que posea el usuario que use el sistema.

2.9 Conclusiones

Luego de la búsqueda y análisis de la información y las tecnologías a utilizar para el producto GeoQ; de definido los principales conceptos asociados al dominio de las áreas de Arquitectura de Software y Sistemas de Información Geográficos, en este capítulo se ha abordado la descripción de la base arquitectónica para el desarrollo del paquete de arquitectura de software del SIG con el apoyo de la descripción de los artefactos de la AS según RUP.

Se presentaron un conjunto de elementos que ayudaron a definir la funcionalidad del sistema, como es la descripción atributos de calidad y los requisitos que tienen impacto sobre la arquitectura así como la herramienta de integración a utilizar para la comunicación con otros sistemas. Se muestran los casos de uso arquitectónicamente significativos, los cuales serán la base para el desarrollo de las demás vistas. En la vista lógica se realizó una estructura de paquetes, teniendo en cuenta el estilo arquitectónico y patrones utilizados. Las vistas: implementación y despliegue muestran la infraestructura de tecnologías que se necesita para el correcto funcionamiento del SIG, haciendo énfasis en sus principales componentes. Ambas vistas muestran la distribución física del sistema, en la primera desde el punto de vista de la implementación y la segunda según los nodos en los que se despliega.

En el capítulo no solo se deja plasmado la descripción arquitectónica de GeoQ sino, con apoyo del anterior, todo lo necesario para la implementación de un SIG omitiendo solo la descripción del paquete en su totalidad para dar cumplimiento al problema planteado.

CAPÍTULO 3: DESCRIPCIÓN DEL PAQUETE DE ARQUITECTURA.

3.1 Introducción

En el presente se hace una propuesta de solución al problema planteado en el diseño teórico de la investigación, tomando como base las descripciones abordadas en el capítulo anterior. Se muestran las representaciones de cada una de las arquitecturas de software según los enfoques arquitectónicos definidos. Se figura cada vista arquitectónicas (4+1 Vistas) que propone RUP, tal como las categorizaciones de la importancia de los SIG y su comunicación o no con otro negocio. La descripción de las representaciones en UML expresadas en el presente capítulo, están referenciadas en el capítulo anterior, en el epígrafe que corresponde a la vista que esté revisando el lector en la investigación.

En cada caso se dejan solamente los paquetes necesarios para la implementación del SIG en el nivel correspondiente y en algunos de estos paquetes las clases que den cumplimiento a los requisitos planteados según el enfoque que este posea. Por ello sin duda alguna se irán reduciendo las representaciones de los artefactos arquitectónicos tal y como los requieran los casos de uso. Estos últimos también pueden sufrir variaciones pues los que involucran a otros casos de uso (casos de uso base) pudieran ser podados de algunos de sus otros casos de uso, tal como zoom y agregar capa.

3.2 Arquitectura de software para nivel de importancia alto

Denotado en los enfoques de los SIG como 100% SIG sin negocio, se refiere a un Sistema de Información Geográfica que no poseerá comunicación con otro sistema informático y estará orientado solamente al trabajo con los mapas y la edición de la cartografía. Ello supone la descripción del capítulo anterior, omitiendo el paquete de integración a través de la herramienta SWIG. Este, en fin, es la solución para el desarrollo de un SIG dedicado únicamente al trabajo con los datos cartográficos sin necesidad de integración con otro negocio. Por ello las descripciones del capítulo 2 figuran la arquitectura de software para un nivel de importancia alto en la propuesta del paquete a realizar.

3.3 Arquitectura de software para nivel de importancia medio

Este nivel se expresa como un SIG que al menos requiere el 50% de sus funcionalidades (las principales), es decir que no es necesario las posea todas, pero sí las más importantes necesarias para la solución con la que se comunica. El SIG se fusiona con otro sistema informático como parte importante donde va a existir una amplia comunicación entre ambos negocios.

3.3.1 Vista de Casos de Uso

Para un nivel de importancia medio se requieren requisitos que cumplan con las funcionalidades principales de los SIG y que sirvan de gran apoyo en la toma de decisiones al sistema informático con el que se integre. Para ello se seleccionan:

1. Agregar capa
2. Conmutar edición
3. Gestionar objeto espacial
4. Realizar Zoom
5. Gestionar Seguridad
6. Autenticar Usuario
7. Administrar Complementos
8. Integración

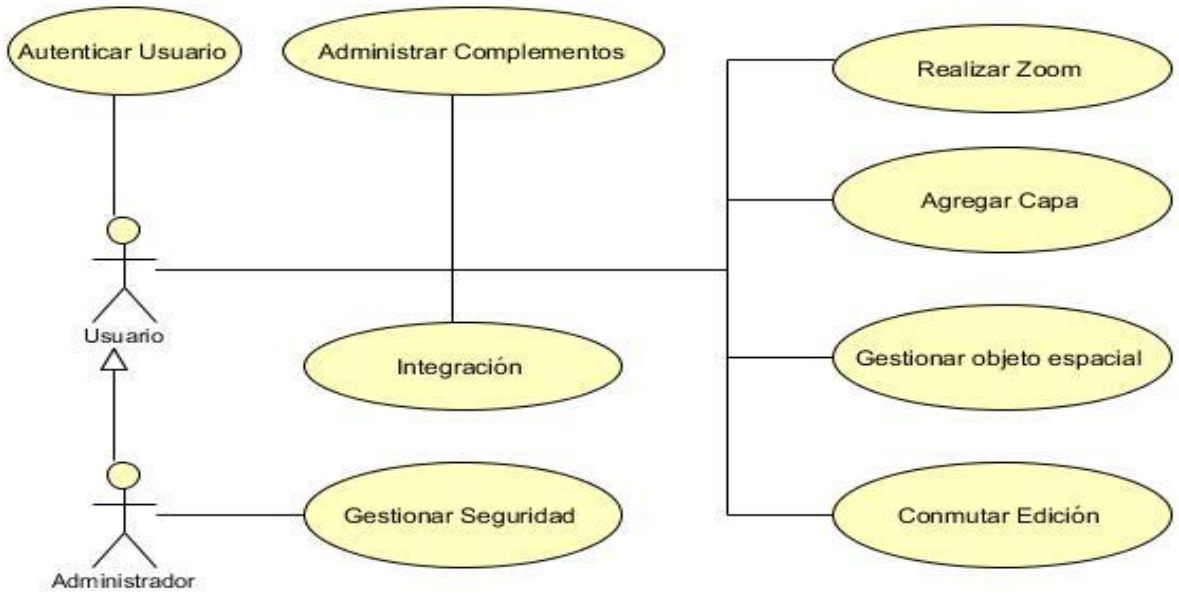

Ilustración 21. Vista de Casos de Uso para un nivel de importancia medio.

3.3.2 Vista Lógica

Responde a los requisitos identificados para satisfacer el diseño de la arquitectura de software para un SIG de nivel medio. El análisis y la especificación de los requisitos funcionales modelados a través de clases son encapsulados en paquetes para su mejor comprensión y organización.

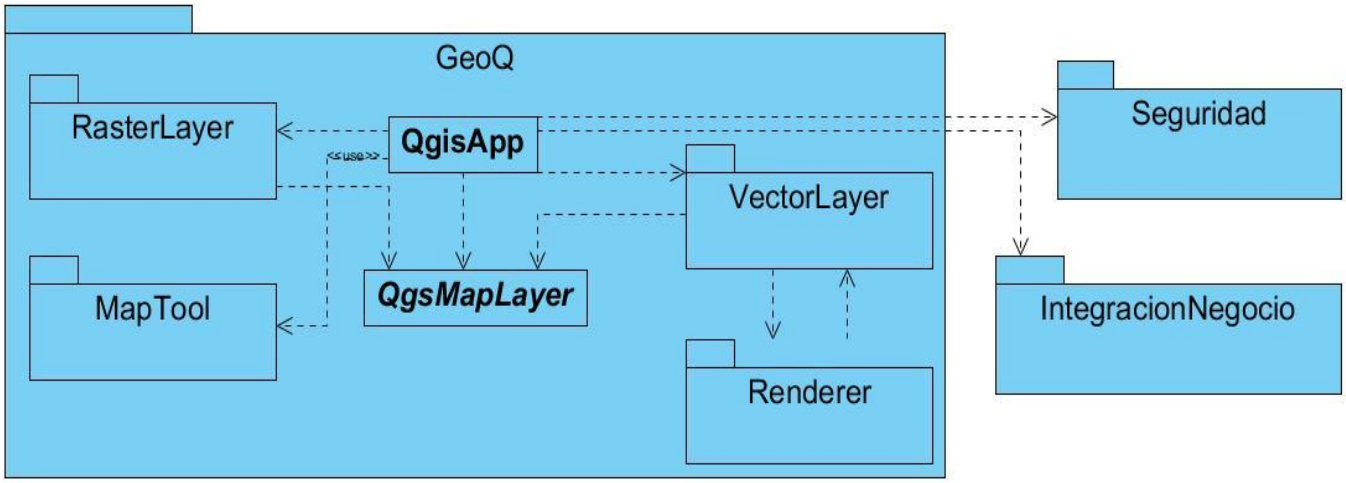

Ilustración 22. Vista Lógica para un nivel de importancia medio.

3.3.3 Vista de Implementación

Da cumplimiento a las características que exige el sistema a este nivel. Cada uno de los paquetes que se representan en el siguiente diagrama encapsulan varios componentes que se interrelacionan para concebir el correcto funcionamiento del sistema GeoQ, para un nivel de importancia medio.

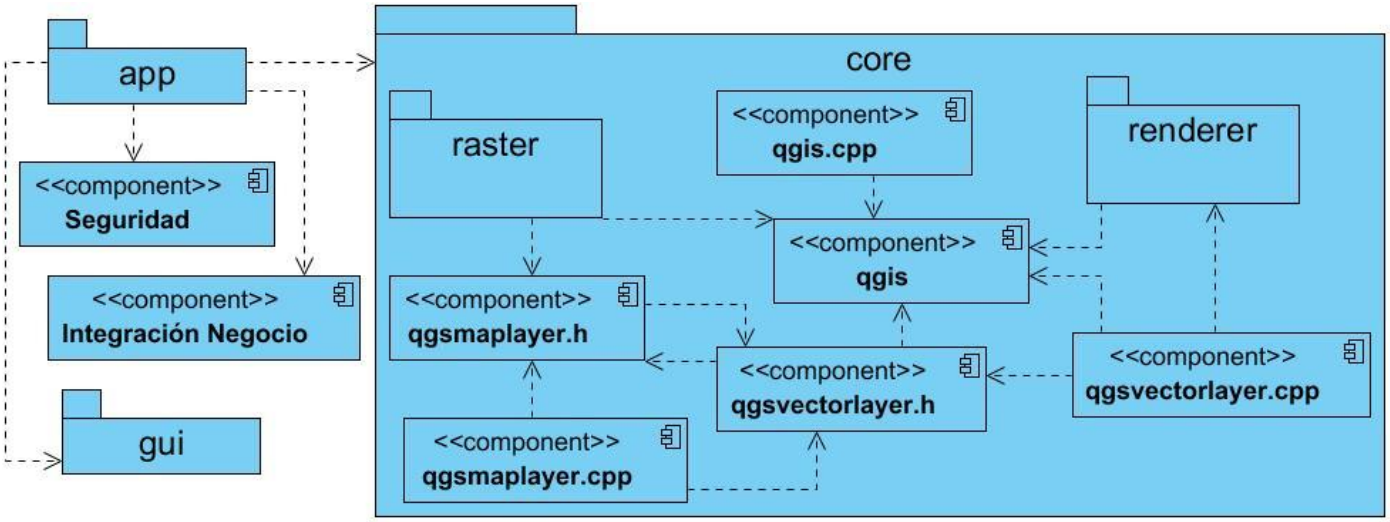

Ilustración 23. Vista de Implementación para un nivel de importancia medio.

3.3.4 Vista de Despliegue

Es la distribución física del sistema con cada uno de sus componentes representados a través de nodos. Supone los mismos componentes y nodos que la descripción general, despliegue que se propone actualmente para GeoQ.

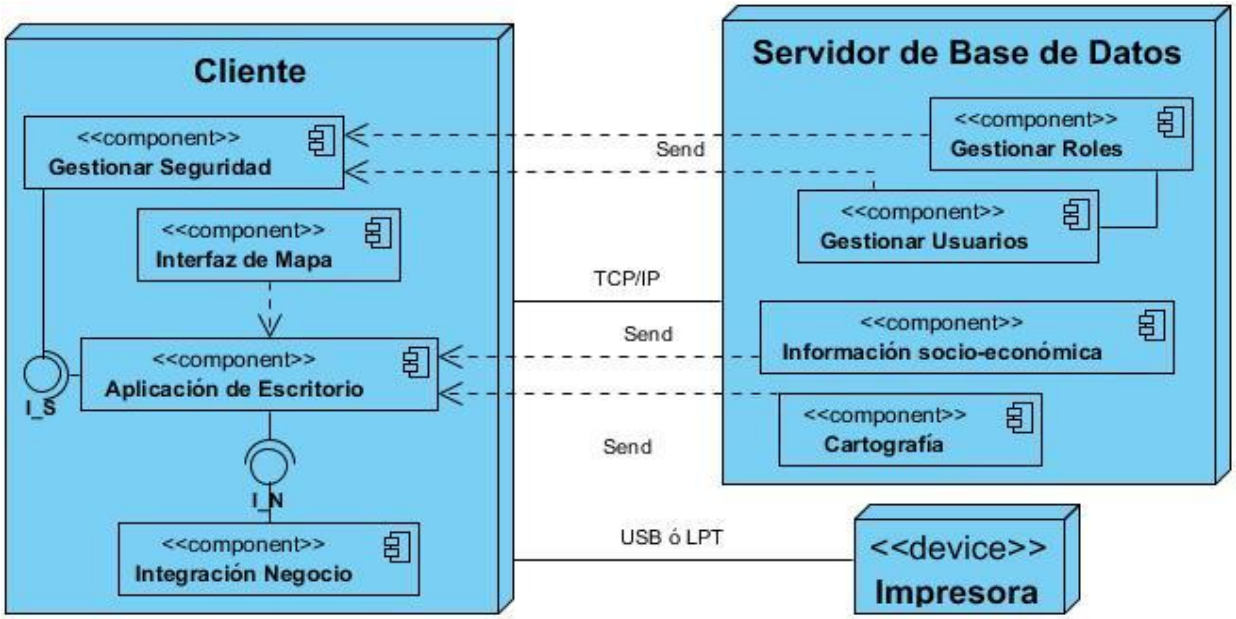

Ilustración 24. Vista de Despliegue para un nivel de importancia medio.

3.4 Arquitectura de software para nivel de importancia bajo

Refiere a un SIG que se comunica con un sistema informático, donde el primero no es de mucha importancia o relevancia en el negocio. En esta clasificación el sistema de información geográfica no posee más funcionalidades que las necesarias para el sistema con el que establece la comunicación, generalmente las funciones que sirvan para la toma de decisiones; en este caso con mucho menos características que la del nivel medio.

3.4.1 Vista de Casos de Uso

Para el diseño de la arquitectura de software de un SIG un nivel de importancia bajo en la integración con otros negocios, se seleccionan como casos de uso arquitectónicamente significativos:

1. Agregar capa
2. Realizar Zoom
3. Integración

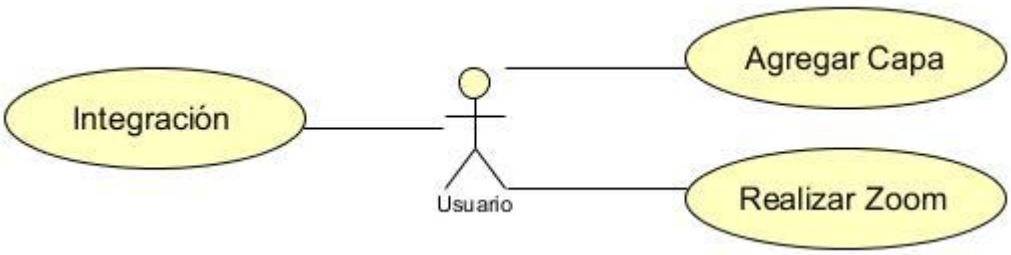

Ilustración 25. Vista de Casos de Uso para un nivel de importancia bajo.

3.4.2 Vista Lógica

El análisis y la especificación de los requisitos funcionales que el sistema suministra en términos de prestaciones a los usuarios finales, a través de un conjunto de abstracciones representadas mediante clases, serán descritos como sigue.

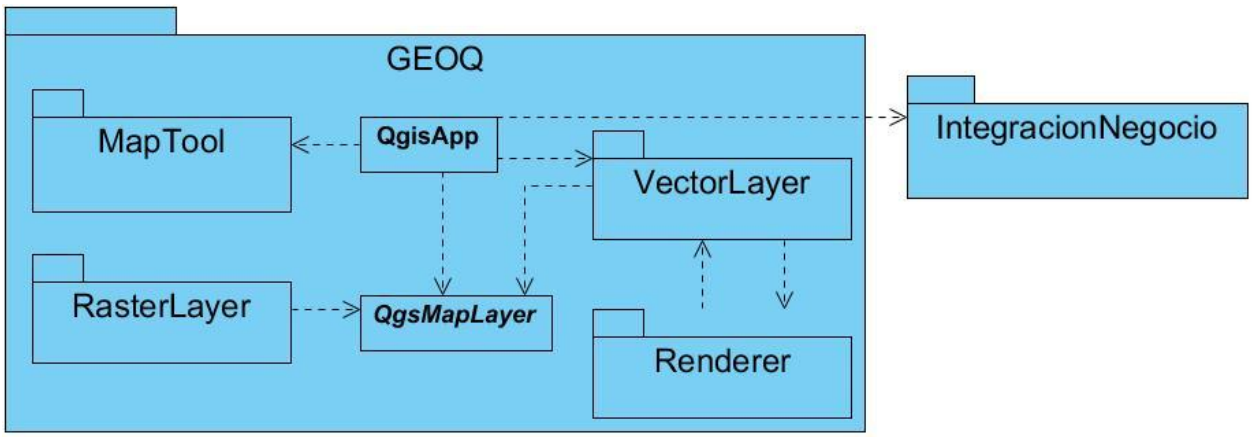

Ilustración 26. Vista Lógica para un nivel de importancia bajo.

3.4.3 Vista de Implementación

En esta vista se realizará la descripción de cómo serán implementados cada uno de los componentes agrupados en subsistemas. Se detalla el mapeo de los paquetes y clases de la vista lógica a subsistemas y componentes físicos.

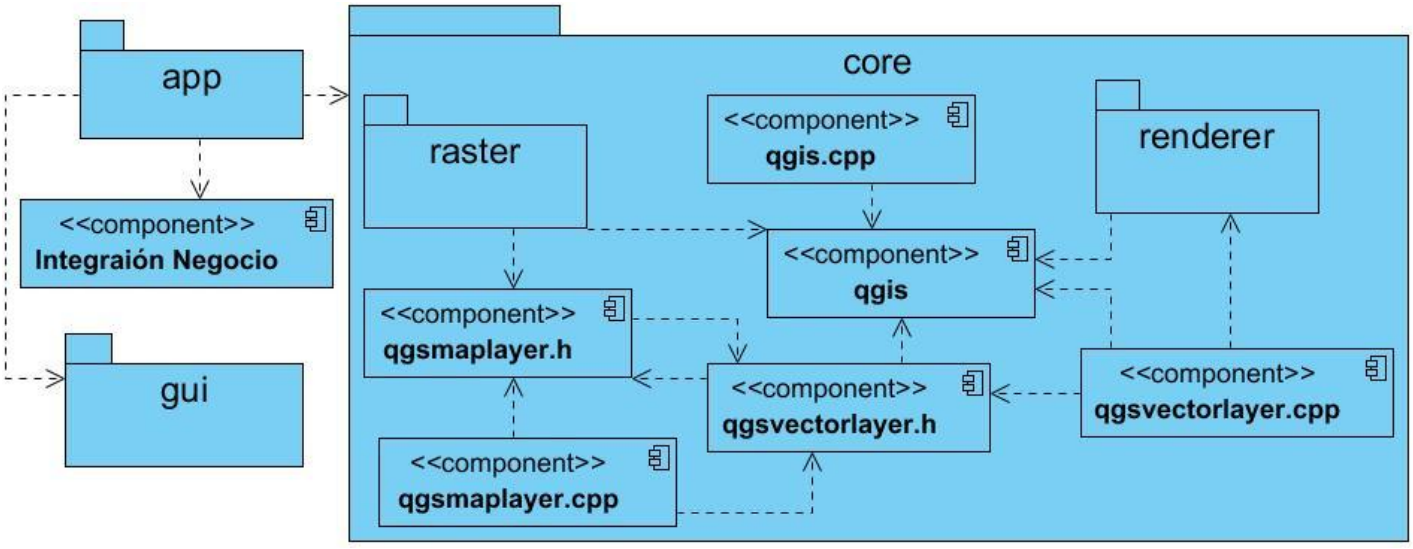

Ilustración 27. Vista de Implementación para un nivel de importancia bajo.

3.4.4 Vista de Despliegue

Para la comprensión de la distribución física del sistema se representa la vista física con cada uno de sus componentes a través de nodos.

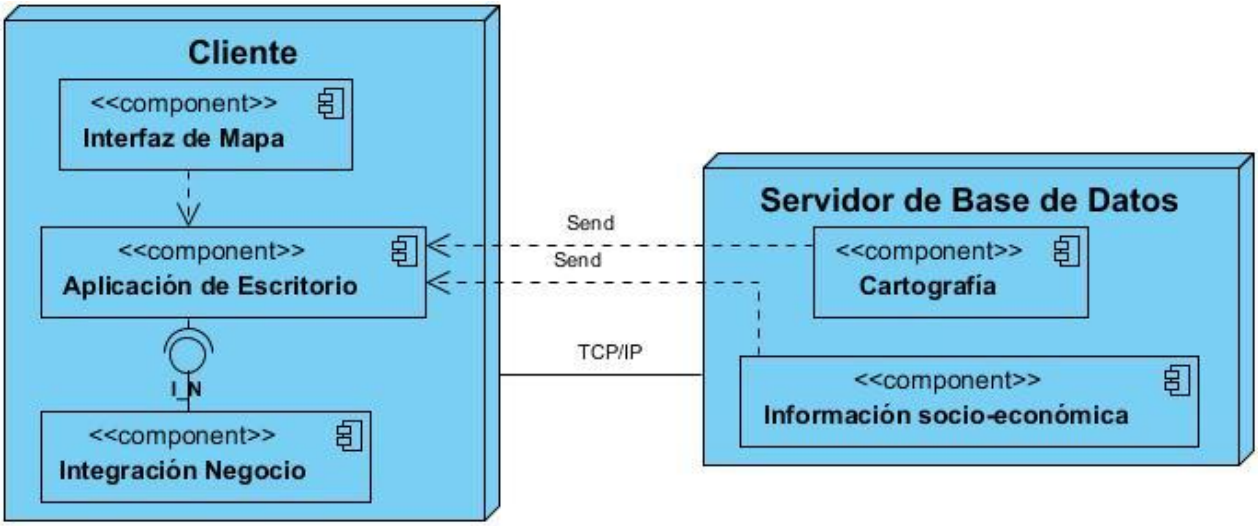

Ilustración 28. Vista de Despliegue para un nivel de importancia bajo.

3.5 Conclusiones

Realizada la descripción de la base arquitectónica para el desarrollo del paquete de arquitectura de software del sistema de información geográfica GoeQ, en este capítulo se procede precisamente a realizar el diseño y descripción del mismo. Como guía para su desarrollo se tienen las representaciones en el capítulo anterior y los enfoques arquitectónicos enumerados previamente. Cada uno de estos enfoques figura la descripción arquitectónica del paquete, definiendo las vistas que propone RUP para el desarrollo de arquitecturas de software.

Para el primer nivel (100% SIG) se define que la representación arquitectónica del paquete coincide con la descrita en el capítulo 2. El segundo de los niveles (50% SIG, 50% Negocio) propone una arquitectura de software de un SIG capaz de integrarse a otro sistema informático y que posee sus principales funcionalidades, que han de servir en gran medida al software con el que se comunique. En el tercer nivel (20% SIG, 80% Negocio) se define la descripción arquitectónica correspondiente a un SIG con el mínimo de sus funciones que pueda integrarse a otro negocio y le sirva de apoyo a este último, de forma primordial en la toma de decisiones.

Este capítulo figura la solución al problema planteado en el diseño teórico de la investigación y deja plasmado los pasos a seguir para la construcción de un sistema de información geográfica de escritorio para distintas funciones según el enfoque en el que se encuentre identificado.

CAPÍTULO 4: EVALUACIÓN DE LA ARQUITECTURA.

4.1 Introducción

La AS representa una gran importancia en el desarrollo de todo software, pues la misma incluye muchas decisiones en pos de lograr no solo la culminación exitosa, sino su fácil evolución futura. Para alcanzar tal objetivo es necesario plantear una rigurosa evaluación del diseño arquitectónico con el fin de determinar a tiempo las debilidades y tomar medidas respecto a ellas.

En el presente capítulo se pretende evaluar la arquitectura del software analizando los resultados obtenidos de acuerdo con las técnicas y métodos de evaluación. Ello se realiza para que la propuesta arquitectónica sea funcional y cumpla con los atributos de calidad definidos en los diferentes modelos de evaluación; por lo que se somete a prueba en busca de que el sistema que la soporte sea robusto, seguro y de gran rendimiento y usabilidad.

4.2 Evaluando la Arquitectura de Software

A la AS se le realizan evaluaciones arquitectónicas con el objetivo de conocer si se pueden habilitar los requisitos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los usuarios finales. Con el propósito de analizar e identificar los principales riesgos en sus propiedades y estructura, los cuales pueden afectar el resultado del software que la soporte; verificar la existencia de los requisitos no funcionales e identificar en qué grado se satisfacen los atributos de calidad. Dichas evaluaciones podrían entonces determinar todos los posibles desastres de un diseño arquitectónico y el nivel de aceptación de la AS diseñada para el sistema. Aunque no definen una calificación sí perciben los riesgos, es decir las fortalezas y debilidades identificadas. Un buen momento para desarrollarla es cuando se comiencen a tomar decisiones que dependan de la línea base, principalmente en los casos en que sea más costoso deshacer una decisión que evaluar la arquitectura. A partir de su realización podrían surgir nuevas interrogantes:

1. ¿Se puede seguir el proyecto con las áreas de debilidad encontradas en la evaluación?
2. ¿Habrá que reforzar la A.S.?
3. ¿Se tendrá que comenzar de nuevo toda la A.S.?

De ahí que las ventajas de su aplicación residen en que cuanto más temprano se encuentre un problema en un proyecto de software, mejor será para su posterior desarrollo y que refiere ser la forma más económica de evitar desastres.

“La evaluación de una arquitectura de software es una tarea no trivial, puesto que se pretende medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo los diseños arquitectónicos. Por ello, la intención es más bien la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos. Las mediciones que se realizan sobre una arquitectura de software pueden tener distintos objetivos, dependiendo de la situación en la que se encuentre el arquitecto y la aplicabilidad de las técnicas que emplea. Algunos de estos objetivos son: cualitativos, cuantitativos y máximos y mínimos teóricos”. **(YOAN ARLET CARRASCOSO PUEBLA 2008)**

1. La medición cualitativa se aplica para la comparación entre arquitecturas candidatas y tiene relación con la intención de saber la opción que se adapta mejor a cierto atributo de calidad. Este tipo de medición brinda respuestas afirmativas o negativas, sin mayor nivel de detalle.
2. La medición cuantitativa busca la obtención de valores que permitan tomar decisiones en cuanto a los atributos de calidad de una arquitectura de software. El esquema general es la comparación con márgenes establecidos, como lo es el caso de los requisitos de desempeño, para establecer el grado de cumplimiento de una arquitectura candidata, o tomar decisiones sobre ella. Este enfoque permite establecer comparaciones, pero se ve limitado en tanto no se conozcan los valores teóricos máximos y mínimos de las mediciones con las que se realiza la comparación.
3. La medición de máximo y mínimo teórico contempla los valores teóricos para efectos de la comparación de la medición con los atributos de calidad especificados. El conocimiento de los valores máximos o mínimos permite el establecimiento claro del grado de cumplimiento de los atributos de calidad.

La arquitectura de software puede ser evaluada en cualquier momento de su diseño o implementación mas los mejores resultados se obtendrían cuando esta se encuentra expuesta totalmente y no se ha iniciado su ejecución. Existen dos variantes útiles para realizar una evaluación: la temprana y la tardía.

La evaluación temprana: Para realizar esta evaluación no es necesario que la arquitectura se encuentre completamente especificada. Ella permite efectuar decisiones sobre la arquitectura en cualquier nivel, puesto que se pueden imponer cambios arquitectónicos como producto de una evaluación, en función de los atributos de calidad esperados.

La evaluación tardía: Se realiza cuando la arquitectura del sistema se encuentra establecida y se ha terminado su implementación; es decir, en el momento de adquisición de un sistema ya terminado. Consideran los autores que la evaluación en este punto es muy importante y útil, puesto que puede observarse el cumplimiento de los atributos de calidad asociados al sistema y cómo será su comportamiento general. La evaluación de la arquitectura de *software* debe

realizarse cuando esta contiene suficientes elementos como para justificarla. Un buen momento para determinar cuándo realizar la evaluación es cuando el equipo de desarrollo comienza a tomar decisiones que dependen de la arquitectura, y que de no tenerlas en cuenta, aumentaría el costo de realizar una evaluación.

4.3 Método de evaluación ATAM

Su estrategia general consiste en elaborar y filtrar escenarios de calidad y relacionarlos con aspectos específicos de la arquitectura que ayudan a cumplirlos. Indica cuán bien una arquitectura de software satisface las metas de calidad y provee ideas de cómo esas metas interactúan y realizan concesiones mutuas (tradeoff) entre ellas. Por ello la selección de este método para la evaluación. Consta de nueve pasos, divididos en cuatro grupos:

Presentación

1. Presentación del ATAM: A todos los participantes en la evaluación se les describen los pasos del ATAM en resumen, así como las técnicas que serán utilizadas para la obtención y el análisis. Se trata de establecer las expectativas y se responden las preguntas propuestas. Se detallan las salidas de la evaluación.
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 presentan las funciones más importantes del sistema y las restricciones técnicas. Así como la mayoría de los stakeholders y las guías de la arquitectura de software.

Las principales funciones del sistema son las descritas en el Capítulo 2 de la investigación

Restricciones del sistema:

a) Se debe recopilar la información cartográfica con los que trabaja el SIG GeoQ referente a la geografía a tratar.
b) Los datos son de vital importancia y de carácter estratégico en la toma de decisiones de muchos negocios por lo que es de vital importancia garantizar su seguridad.
c) Debe existir un único repositorio central para almacenar todos los datos recopilados.
d) Debe existir Bases de Datos de respaldo que resguarden la información sensible que se manipule en el sistema.
e) Generalmente en relación con el negocio en el que se despliegue dicho sistema, la cantidad de información que se maneja es muy grande.

Metas de los atributos de calidad que dan forma a la arquitectura:

a) Lograr que el sistema realice el trabajo para el cual fue concebido.
b) Lograr que el sistema se mantenga operativo por largo tiempo.
c) Lograr que el grado con el que el sistema cumpla sus funciones designadas, dentro de ciertas restricciones como velocidad, exactitud o uso de memoria, sea satisfactorio.
d) Lograr la capacidad de al someter el sistema a reparaciones y evoluciones, de ser necesario, estas se desarrollen de manera rápida y a bajo costo.
e) Lograr la habilidad de realizar cambios futuros al sistema sin provocar graves afectaciones a lo ya desarrollado.
f) Lograr que el sistema pueda ser ejecutado en diferentes ambientes computacionales.

3. El arquitecto describe las propuestas arquitectónicas, enfocándose en cómo esta cumple con los objetivos del negocio. Esta descripción no se presenta en esta sección pues la arquitectura que está siendo evaluada se mostró completamente en los capítulos anteriores.

Investigación y Análisis

4. Identificación de los enfoques arquitectónicos: El ATAM centraliza el análisis de la arquitectura en el entendimiento de sus propuestas arquitectónicas, en este paso son capturadas por el equipo de evaluación pero no son analizadas. Quedan identificadas todas las propuestas arquitectónicas que serán analizadas. En esta sección se especifica la tecnología a utilizar, lo estilos y patrones arquitectónicos y los de diseño, mas estos elementos no se muestran puesto que se encuentran detallados en los capítulos anteriores.

5. Generación del Árbol de Utilidad: Se priorizan los atributos de calidad del sistema especificados en forma de escenarios. Se anotan los estímulos y respuestas y se establece la prioridad entre ellos. Este paso es crucial, pues guía el resto del análisis. Para ello se define la prioridad del escenario como un par (X, Y) en el cual X define el esfuerzo de satisfacer el escenario e Y indica los riesgos que se corren al excluirlos del árbol. Los valores pueden ser A (Alto), M (Medio) y B (Bajo):

Funcionalidad	Seguridad	(A,A) El acceso a cualquier manipulación del sistema, tanto entrada como análisis de datos debe estar sometido a un proceso de autentificación del usuario donde será especificado rol, usuario y contraseña.
		(A,M) Las funcionalidades del sistema se muestran de acuerdo al rol y permisos que posea el usuario autenticado.
		(A,A) Activar protección contra ataques que puedan afectar la integridad de los datos almacenados.
		(A,A) Paralelo a la base de datos primaria se debe mantener una base de datos de respaldo

		para evitar la pérdida de los datos ante algún desastre natural.
Confiabilidad	Recuperabilidad	(M,A) En caso de fallo restablecer el nivel de desempeño y recuperar el sistema y los datos entrados sin enviar, mediante mecanismos o dispositivos de software.
Eficiencia	Utilización de recursos	(A,M) Revisar y mejorar la relación de los componentes en términos de espacio y tiempo con el objetivo de que los tiempos de respuesta a las peticiones realizadas por los usuarios sean mínimos.
Portabilidad	Adaptabilidad	(A,B) Debe existir la presencia de mecanismos de adaptación, que faciliten el uso de la aplicación en los sistemas operativos Windows y GNU/Linux.
		(A,B) Acceder a las Bases de Datos del sistema desde cualquier parte del país donde se encuentre instalada la aplicación y con los permisos correspondientes.

Tabla 2. Árbol de Utilidad

6. Análisis de los enfoques arquitectónicos: Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. En este paso se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance:

Escenario # 1	Escenario: El acceso a cualquier manipulación del sistema, tanto de entrada como de análisis de datos debe estar sometido a un proceso de autentificación del usuario donde será especificado rol, usuario y contraseña.			
Atributo	Funcionalidad-Seguridad			
Entorno	Integración GeoQ-Seguridad			
Estímulo	Necesidad de proteger al sistema de usuarios que no deban acceder al mismo.			
Respuesta	Mostrar ventana de autenticación para que se introduzcan las credenciales.			
Decisión arquitectónica		Sensitivity points	Tradeoff points	Riesgo	No Riesgo
Subsistema de Seguridad		-	-	R1	NR1
Razonamiento	R1: Como GeoQ es software libre, no se puede asegurar el acceso a las funcionalidades del sistema que sean programadas en el propio SIG, sino solamente las que necesiten de datos almacenados.			

	NR1: Desarrollado como un subsistema independiente, que gestiona principalmente la seguridad en las bases de datos, posibilita integrarse a cualquier sistema informático y gestionar su seguridad.

Tabla 3. Análisis de los enfoques arquitectónicos. Escenario 1.

Escenario # 2	Escenario: Las funcionalidades del sistema se muestran de acuerdo al rol y permisos que posea el usuario autenticado.			
Atributo	Funcionalidad-Seguridad			
Entorno	Integración GeoQ-Seguridad			
Estímulo	Necesidad de evitar el acceso a funcionalidades del sistema por usuarios que no posean los permisos correspondientes.			
Respuesta	Los usuarios solo podrán interactuar con las funcionalidades a las que tienen acceso de acuerdo al rol que posean.			
Decisión arquitectónica	Sensitivity points	Tradeoff points	Riesgo	No Riesgo
Subsistema de Seguridad	-	-	R2	NR2
Razonamiento	R2: Como GeoQ es software libre, no se puede asegurar el acceso a las funcionalidades del sistema que sean programadas en el propio SIG, sino solamente las que necesiten de datos almacenados. NR2: Desarrollado como un subsistema independiente, que gestiona principalmente la seguridad en las bases de datos, posibilita integrarse a cualquier sistema informático y gestionar su seguridad.			

Tabla 4. Análisis de los enfoques arquitectónicos. Escenario 2.

Escenario # 3	Escenario: Activar protección contra ataques que puedan afectar la integridad de los datos almacenados.			
Atributo	Funcionalidad-Seguridad			
Entorno	Seguridad del sistema			
Estímulo	Necesidad de evitar el ataque directo o a través de la aplicación a las BD por agentes externos.			
Respuesta	No tienen efecto los intentos de ataques.			
Decisión arquitectónica	Sensitivity points	Tradeoff points	Riesgo	No Riesgo

Uso del SGBD PostgreSQL	S1	-	R3	NR3
Durante el despliegue del sistema se han de tener en cuenta la separación del nodo cliente del nodo servidor de bases de datos.	S2	-	R4	NR4
Razonamiento	S1: Aumenta la eficiencia S2: Aumenta la confiabilidad R3: Bajo una condición de stress, como la conexión concurrente de múltiples usuarios a la BD haciendo diferentes o las mismas peticiones, puede afectar la eficiencia del sistema. R4: Fallo o lentitud en la conexión del nodo cliente al nodo servidor de base de datos. NR3: Maneja las conexiones concurrentes de muchos usuarios de forma eficiente. NR4: La distribución física del sistema en distintos nodos contribuye a la protección de los datos y hace más difícil que el atacante conozca la ubicación real del servidor de BD.			

Tabla 5. Análisis de los enfoques arquitectónicos. Escenario 3.

Escenario # 4	Escenario: Paralelo al servidor de base de datos primario se debe mantener un servidor de base de datos de respaldo para evitar la pérdida de los datos ante algún desastre natural.			
Atributo	Funcionalidad-Seguridad			
Entorno	Base de Datos			
Estímulo	Necesidad de salvar los datos en caso de algún fallo en la base de datos primaria.			
Respuesta	En caso de ocurrir algún fallo o incidencia en el servidor de base de datos se pueden recuperar los datos almacenados en el servidor de respaldo.			
Decisión arquitectónica		Sensitivity points	Tradeoff points	Riesgo	No Riesgo
Durante el despliegue del sistema se han de tener en cuenta añadir		S3	-	-	NR5

un servidor de respaldo, así como su conectividad con el resto de los nodos.				
Razonamiento	S3: Aumenta la confiabilidad NR5: Mediante la decisión arquitectónica de un sistema de respaldo de bases de datos los datos estarán garantizados en otro servidor de bases de datos.			

Tabla 6. Análisis de los enfoques arquitectónicos. Escenario 4.

Escenario # 5	Escenario: En caso de fallo restablecer el nivel de desempeño y recuperar el sistema y los datos entrados sin enviar, mediante mecanismos o dispositivos de software.			
Atributo	Confiabilidad-Recuperabilidad			
Entorno	SIG GeoQ			
Estímulo	Error del SO o SIG que reinicia la PC cliente.			
Respuesta	Se recuperan solamente los datos enviados al servidor de base de datos.			
Decisión arquitectónica		Sensitivity points	Tradeoff points	Riesgo	No Riesgo
		S4	T1	R5	-
Razonamiento	S4: Se afecta la confiabilidad del sistema. T1: La afectación de la confiabilidad disminuye la usabilidad. R5: Aumenta el riesgo de que los datos queden inconsistentes ante una eventual falla por falta de fluido eléctrico, error del sistema operativo u otro.			

Tabla 7. Análisis de los enfoques arquitectónicos. Escenario 5.

Escenario # 6	Escenario: Revisar y mejorar la relación de los componentes en términos de espacio y tiempo con el objetivo de que los tiempos de respuesta a las peticiones realizadas por los usuarios sean mínimos.
Atributo	Eficiencia-Utilización de recursos
Entorno	Múltiples usuarios haciendo peticiones a las bases de datos.
Estímulo	Necesidad de realizar peticiones continuas que necesiten de datos almacenadas en el servidor de base de datos.

Respuesta	Dar respuesta a las peticiones en el menor tiempo posible.			
Decisión arquitectónica	Sensitivity points	Tradeoff points	Riesgo	No Riesgo
Uso del SGBD PostgreSQL	S5	-	R6	NR6
Implementar procedimientos almacenados	S5	-	-	NR7
Razonamiento	S5: Aumenta la eficiencia R6: Bajo una condición de stress, como la conexión concurrente de múltiples usuarios a la BD haciendo diferentes o las mismas peticiones, puede afectar la eficiencia del sistema. NR6: Maneja las conexiones concurrentes de muchos usuarios de forma eficiente. NR7: Permite que las respuestas de las consultas a la BD sean más rápidas.			

Tabla 8. Análisis de los enfoques arquitectónicos. Escenario 6.

Escenario # 7	Escenario: Debe existir la presencia de mecanismos de adaptación que faciliten el uso de la aplicación en los sistemas operativos Windows y GNU/Linux.			
Atributo	Portabilidad-Adaptabilidad			
Entorno	Desarrollo del SIG GeoQ			
Estímulo	Necesidad de instalar y ejecutar el SIG en diferentes sistemas operativos (SO).			
Respuesta	Asegurar la presencia de mecanismos de adaptación para varios SO.			
Decisión arquitectónica	Sensitivity points	Tradeoff points	Riesgo	No Riesgo
Al desarrollarse el SIG GeoQ se debe tener en cuenta que el mismo pueda ser ejecutado en distintos SO.	S6	T2	-	-
Razonamiento	S6: Aumenta la usabilidad. T2: Aumenta la usabilidad pero disminuye el rendimiento.			

Tabla 9. Análisis de los enfoques arquitectónicos. Escenario 7.

Escenario # 8	Escenario: Acceder a las Bases de Datos del sistema desde cualquier parte del país donde se encuentre instalada la aplicación y con los permisos correspondientes.			
Atributo	Portabilidad-Adaptabilidad			
Entorno	Bases de Datos del sistema			
Estímulo	Necesidad de establecer la conexión del sistema con las bases de datos ubicada fuera de la instalación donde está desplegado el SIG.			
Respuesta	Asegurar la conectividad entre el sistema y el servidor de base de datos.			
Decisión arquitectónica		Sensitivity points	Tradeoff points	Riesgo	No Riesgo
Apreciar la ubicación y la conectividad de los servidores con el sistema en el despliegue.		S7	-	-	NR8
Razonamiento	S7: Aumenta la funcionalidad NR8: Independiza al sistema de la protección y almacenamiento de los datos.				

Tabla 10. Análisis de los enfoques arquitectónicos. Escenario 8.

Pruebas

7. Lluvia de ideas y establecimiento de prioridad de escenarios: Este paso consiste en la generación de nuevos escenarios para representar los intereses de los stakeholders que no hayan sido comprendidos, con la colaboración de todos los involucrados. Los escenarios resultantes son comparados con los del árbol de utilidad y se adicionan al obtenido durante el proceso de evaluación.

Escenario #9	Atributo de Calidad	Importancia	Fac. alcance
Comunicación del SIG con sistemas externos.	Funcionalidad-Interoperabilidad	A	A

Tabla 11. Nuevo escenario. Escenario 9.

Escenario #10	Atributo de Calidad	Importancia	Fac. alcance

Componentes con las funciones responsables de los cálculos o funcionalidades.	Funcionalidad- Exactitud	A	B

Tabla 12. Nuevo escenario. Escenario 10.

8. Análisis de los enfoques arquitectónicos: Este paso repite las actividades del paso 6, mapeando los escenarios recientemente generados con ranking más alto en los artefactos arquitectónicos. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento:

Escenario # 9	Escenario: Comunicación del SIG con sistemas externos.			
Atributo	Funcionalidad- Interoperabilidad			
Entorno	Integración entre sistemas			
Estímulo	Necesidad de integrar GeoQ con sistema informático			
Respuesta	Se logra la integración entre ambos sistemas según el enfoque del SIG.			
Decisión arquitectónica	Sensitivity points	Tradeoff points	Riesgo	No Riesgo
Realización de un paquete de Integración.	-	-	-	NR9
Uso de la herramienta SWIG	S8	-	-	-
Razonamiento	S8: Aumenta la usabilidad NR9: La creación de un paquete de integración para GeoQ permite la comunicación entre el SIG y de diferentes negocios sin importar cuales sean.			

Tabla 13. Análisis de los enfoques arquitectónicos. Escenario 9.

Escenario # 10	Escenario: Componentes con las funciones responsables de los cálculos o funcionalidades.			
Atributo	Funcionalidad- Exactitud			
Entorno	SIG GeoQ			
Estímulo	Necesidad de la existencia de componentes que realicen las funciones de funcionalidades específicas.			
Respuesta	Claridad en la implementación del sistema y reutilización de las funcionalidades.			
Decisión arquitectónica	Sensitivity points	Tradeoff points	Riesgo	No Riesgo

Realización de paquetes de funcionalidades con las clases que las manejan.	-	-	-	NR10
Razonamiento	NR10: Se pueden reutilizar dichos paquetes en la implementación de otro SIG.			

Tabla 14. Análisis de los enfoques arquitectónicos. Escenario 10.

9. Se resume y presentan las salidas más importantes de la información recolectada a lo largo del proceso de evaluación del ATAM validando el paquete arquitectónico propuesto:

- ❖ Conjunto de escenarios priorizados: Anexo 5.
- ❖ Árbol de utilidad: Anexo 6.
- ❖ Riesgos descubiertos: Anexo 7.
- ❖ Los no riesgos documentados: Anexo 8.
- ❖ Los sensitivity points y tradeoff points encontrados: Anexo 9.

4.4 Conclusiones

Se evaluó la arquitectura del software tomando como premisa los resultados obtenidos de acuerdo con las técnicas y métodos de evaluación. Para obtener los posibles errores de un diseño arquitectónico y el nivel de aceptación de la AS diseñada para el sistema, se evalúa con el método ATAM el cual en sus resultados arroja:

- Conjunto de escenarios priorizados
- Árbol de utilidad
- Riesgos descubiertos
- Los no riesgos documentados
- Los puntos de sensibilidad y equilibrio encontrados

La elaboración y filtrado de los escenarios de calidad, los puntos de sensibilidad e intercambio obtenidos así como los riesgos y no riesgos derivados de la evaluación arquitectónica, demuestran que la propuesta arquitectónica es funcional y cumple con los atributos de calidad definidos y por ende el sistema que la soporte poseerá como características: robusto, seguro y de gran rendimiento y usabilidad. Ello expone que se pueden habilitar los requisitos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los usuarios finales.

CONCLUSIONES

Todos y cada uno de los sistemas informáticos deben de estar respaldados por una arquitectura sólida que facilite su entendimiento, que organice el desarrollo del software y asegure que el mismo sea reutilizable; todo ello es de vital importancia. De ahí que en la desarrollo de la presente investigación se haya hecho un estudio de los principales aspectos arquitectónicos. Así como la descripción del paquete arquitectónico propuesto, con cada una de las arquitecturas basadas en los estilos seleccionados, sus componentes, configuraciones y restricciones.

Se fundamentaron los patrones los cuales proporcionaron al sistema calidad, claridad y sencillez en la estructura y diseño de la solución. Se hizo uso de varias vistas arquitectónicas, dictaminadas por la metodología seleccionada RUP, entre las que figuran la de los Casos de Uso, Lógica, Implementación y Despliegue. Se definieron las herramientas y tecnologías necesarias para el desarrollo y finalmente se evaluó dicha arquitectura. Lo que demostró el cumplimiento de los objetivos de la investigación, a través de la alta correspondencia entre los requerimientos, atributos de calidad y diseño arquitectónico; posibilitando la construcción de una aplicación robusta, flexible y reusable.

Con la descripción de todo ello se concluye que:

- El estudio de otros Sistemas de Información Geográfica a nivel mundial mostró que no existe un sistema que integre un amplio conjunto de funcionalidades de alto procesamiento y que además posean técnicas de integración y seguridad de sistemas, construido sobre entornos de escritorio.
- Los diseños propuestos para GeoQ proporcionan una estructura altamente escalable, de manera tal que se le pueden seguir incorporando funcionalidades, en pos de realizar personalizaciones para un negocio específico que lo requiera.
- Las herramientas, metodología y lenguaje de modelado identificados a partir de la investigación de las tendencias y tecnologías actuales para el desarrollo de sistemas de información geográficas, garantizó una modelación estable y organizada de los procesos que formarán parte de GeoQ.
- La modelación del sistema propuesto así como el diseño correspondiente al mismo contribuirá a una correcta implementación del SIG.

RECOMENDACIONES

Realizado el análisis de una significativa cifra de elementos que favorecieron la elaboración de la presente investigación, es necesario que se enumeren algunas recomendaciones para la mejora continua de la propuesta en cuestión:

1. Analizar y aplicar constantemente el refinamiento de la arquitectura propuesta durante el ciclo de desarrollo y en vistas de versiones superiores del producto para lograr la correcta implementación del sistema.
2. Según los objetivos que se persigan con la arquitectura de software se propone validarla con otro método para identificar las principales debilidades en el diseño arquitectónico.
3. Se recomienda desarrollar GeoQ a través de esta propuesta y valorar la posibilidad de hacerla extensiva a otras áreas temáticas relacionadas con los SIG.
4. Investigar y añadir funcionalidades al sistema en función de lo nuevos requerimientos que puedan ser identificados.
5. Implementar los Casos de Uso arquitectónicamente significativos en una primera iteración y posteriormente los secundarios.
6. Realizar los diagramas de interacción correspondientes al diseño para ilustrar mejor el comportamiento de las clases definidas así como sus relaciones.
7. Realizar el estudio y análisis de ArcGis el cual constituye una serie integrada de software de Sistemas de Información Geográfica que trabaja como un motor compilador de información geográfica alfanumérica. Accesible desde clientes desktop, navegadores web y terminales móviles. Sus versiones para plataformas de escritorio: ArcGIS Desktop.

REFERENCIAS BIBLIOGRÁFICAS

ALVAREZ, S. *Sistemas gestores de bases de datos*.

ALVARO DE J. CARMONA, J. J. M. R. *Sistemas de información geográfica*. Disponible en: http://www.monografias.com/trabajos/gis/gis.shtml?monosearch

SIVARAMAKRISHNAN SOMASEGAR, S. G., DAVID HILL. *Application Architecture Guide. patterns & practices (2nd edition), 2009.*

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

GEOENSEÑANZA. *Los Sistemas de Información Geográfica, 2006.*

SWIG. *Página oficial de SWIG, 2011. [Disponible en:* ***http://www.swig.org/index.php****]*

IVAR JACOBSON, G. B., JAMES RUMBAUGH. *Rational Unified Process. Best Practices for Software Development Teams, 1999.*

LINPPC. *Sistema Estatal de Información Geográfica implementada a través de software libre/abierto y estándares abiertos, 2010*.

SWIG. *SWIG Documentation, 2011.*

DUEÑAS, C. P. *UML. El modelo dinámico y de implementación, 2009.*

CARLOS REYNOSO, N. K. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft*, 2004.

DAVID GARLAN, M. S. *An Introduction to Software Architecture*, 1994.

DAVID TROWBRIDGE, U. R., GREGOR HOHPE, DRAGOS MANOLESCU. *Integration Patterns. Patterns & Pract ices*.

DEWAYNE E. PERRY, A. L. W. *Foundations for the Study of Software Architecture*, 1992. 17.

DIJKSTRA, E. W. *The Structure of the "THE"-Multiprogramming System*.

ING. YAMILA VIGIL REGALADO, I. E. F. C. *La arquitectura de software como disciplina científica*, 2008.

MARCA HUALLPARA HUGO MICHAEL, Q. L. N. S. *Análisis y diseño de sistemas II. Diagrama de Despliegue*.

OSVALDO DÍAZ VERDECIA, V. D. Q. C. *Una guía práctica de Arquitectura de Software desde la experiencia del proyecto ERP-Cuba.*, Universidad de las Ciencias Infomáticas, 2009. p.

POSTGRESQL, E. D. D. D. *Manual del usuario de PostgreSQL*.

PRESSMAN, R. S. *Ingeniería de software. Un enfoque práctico (6ta edición)*, 2005.

SENA, G. D. A. *Arquitectura de Software*.

SOMMERVILLE, L. *Ingeniería del Software (7ma Edición)*, 2005

YOAN ARLET CARRASCOSO PUEBLA, E. C. G., ANISLEYDI CÉSPEDES VEGA. *Procedimiento para la Evaluación de Arquitecturas de Software basadas en Componentes*, 2008.

BIBLIOGRAFÍA

LARMAN, C. Appling UML and Pattern. An introduction to Object-Oriented Analysis and Design and the Unified Process.

GARY E. SHERMAN, T. S., RADIM BLAZEK, STEPHAN HOLL, OTTO DASSAU, TYLER MITCHELL, BRENDAN MORELY, LARS LUTHMAN, GODOFREDO CONTRERAS, MAGNUS HOMANN, MARTIN DOBIAS. *Quantum GIS. Guía de Usuario e Instalación, 2007.*

LINPPC. *Sistema Estatal de Información Geográfica implementada a través de software libre/abierto y estándares abiertos, 2010*.

BASTARRICA, M. C. *Atributos de Calidad y Arquitectura del Software*. Disponible en:

CASANOVAS, J. *Usabilidad y arquitectura del software*, 2004. [Disponible en: http://www.desarrolloweb.com/articulos/1622.php

ERICH GAMMA, R. H., RALPH JOHNSON, JOHN VLISSIDES. *Design Patterns. Elements of Reusable Object-Oriented Software*.

FRANK BUSCHMANN, R. M., HANS ROHNERT, PETER SOMMERLAD, MICHAEL STAL. *Pattern-Oriented Software Architecture. A System of patterns*.

JAMES RUMBAUGH, I. J., GRADY BOOCH. *El Lenguaje Unificado de Modelado. Manual de referencias.*

KRUCHTEN, P. *Architectural Blueprints—The "4+1" View Model of Software Architecture*, 1995. [Disponible en:

REYNOSO, C. B. *Introducción a la Arquitectura de Software*, 2004. [Disponible en:

RICK KAZMAN, P. K., ROBERT L. NORD, JAMES E. TOMAYKO. *Integrating Software-Architecture-Centric Methods into the Rational Unified Process*, 2004.

GLOSARIO DE TÉRMINOS

En el glosario de términos y siglas se incluye una explicación de los términos y siglas utilizados en el texto para facilitar su comprensión. El Glosario contiene los términos que tienen menor difusión en el campo profesional de los Sistemas de Información Geográficos, los de otro campo profesional tal como la Arquitectura de Software, o aquellos términos conocidos, pero que se usan con un significado diferente en el texto. Son incluidas las siglas utilizadas en el documento para facilitar su lectura. La estructura y organización del glosario es similar a la de un diccionario.

ABC: Arquitectura basada en componentes

AOO: Arquitectura Orientada a Objeto

AS: Arquitectura de Software

Core: Núcleo o centro de un programa, del cual va a depender gran parte de él.

CASE: Ingeniería de Software Asistida por Computación. Es la aplicación de métodos y técnicas a través de las cuales se hacen útiles a las personas comprender las capacidades de las computadoras, por medio de programas, de procedimientos y su respectiva documentación. Representan una forma que permite Modelar los Procesos de Negocios y desarrollar los Sistemas de Información.

GDAL: Es una biblioteca de software para la lectura y escritura de formatos de datos geoespaciales, publicada bajo la licencia X/MIT style Open Source por la fundación geoespacial de código abierto (Open Source Geospatial Foundation).

GPL: Licencia Pública General de GNU/Linux está principalmente orientada a proteger la libre distribución, modificación y uso del software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

OGC: Consorcio Geoespacial Abierto. Su fin es la definición de estándares abiertos e interoperables dentro de los Sistemas de Información Geográfica y persigue acuerdos que posibiliten la interoperación de sus sistemas de geoprocesamiento y faciliten el intercambio de la información geográfica en beneficio de los usuarios.

SIG: Sistema de Información Geográfica.

SIG-DESKTOP: Sistema de Información Geográfica de escritorio.