Universidad de las Ciencias Informáticas

Título: Desarrollo de un componente visual para Qt utilizando el framework OpenCascade.

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autor: Miguel Arcia Salazar

Tutor: Ing. Rocny Morales Delgado

Ciudad de la Habana, junio de 2011.

Curso: 2010-2011

Agradecimientos:

En primer lugar a mis padres, por ser lo más grande que tengo en la vida.

A mis tías, a mis abuelos y a toda mi familia en general por su apoyo y preocupación durante mi carrera.

A mi novia por su cariño, su apoyo y por luchar conmigo todo este tiempo.

A mis suegros por acogerme durante este tiempo y convertirse en otra familia para mí.

A mis hermanos Yoandy, Rubén, Rayner, el kdt, Jorgito, Ivan, Marlu, Sure, Eduardo, por estar ahí para mí cuando los necesité.

A todos mis compañeros de grupo durante estos cinco años.

A mi tutor Rocny por guiarme.

A Eddy y los profes del proyecto por echarme una mano.

A mi tribunal de tesis por su asesoría.

A todos de corazón, gracias.

Declaración de autoría:

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

Para que así conste firmo la presente a los ___días del mes de _____ del año_____.

_____. _____.

Miguel Arcia Salazar. Ing. Rocny Morales Delgado.

Resumen

El presente trabajo define como problemática a resolver la visualización de objetos en tres dimensiones en Qt utilizando el framework OpenCascade, específicamente a través de un componente visual que se integre a Qt. Para tener un mejor dominio de la problemática planteada se realizó una investigación relacionada con materiales bibliográficos y aplicaciones informáticas vinculadas al objetivo propuesto, determinado a la herramienta QtOCC como referencia para la implementación en cuestión. Con el fin de perfeccionar la implementación del producto, se analizaron varias aplicaciones, lenguajes de programación y metodologías de desarrollo de software, seleccionando al lenguaje C++ y a la metodología XP como guiar de desarrollo. Como resultado se obtiene un componente integrado a la interfaz visual de Qt, sobre el cual se logra visualizar cualquier tipo de objeto tridimensional utilizando el framework OpenCascade. Además, se puede mencionar que el resultado de esta investigación puede constituir un punto de partida para la futura implementación de cualquier herramienta vinculada a la visualización tridimensional utilizando el framework OpenCascade.

Índice

Índice de Tablas

Índice de Figuras

Introducción.

Desde el comienzo de la historia de la humanidad, el hombre encontró vías para representar visualmente la realidad en que vivía. Luego de cientos de años de evolución, aparecieron las artes plásticas, el cine, la televisión; de modo que se fueron ampliando los medios para ofrecer cualquier tipo de información visual. Con el actual desarrollo de las Tecnologías de la Información y las Comunicaciones (TICs) y gracias al nacimiento de la Informática Gráfica (IG), se ha logrado visualizar o modelar sólidos en dos o tres dimensiones (2D ó 3D); dando origen a un nuevo paradigma de representación visual conocido como Diseño Asistido por Computadoras (CAD).

El inicio del modelado, diseño o visualización de sólidos a través de una computadora, comenzó en los Estados Unidos y se ha transformado en un importante recurso muy utilizado en varios sectores de la vida cotidiana tales como la minería, la salud, el audiovisual, entre otros. Este proceso de información visual, a través de imágenes dinámicas e interactivas, enriquece y facilita extraordinariamente la comprensión a la hora de representar un cuerpo en 3D. En general las aplicaciones más populares, se mueven alrededor de licencias del software privativo; por lo que esto se convierte en un inconveniente para cualquier institución que económicamente no se pueda permitir los gastos que impone el mercado. Sin embargo, el número de tecnologías destinadas a este sector se ha incrementado de igual manera, tanto en el software libre como en el privativo.

Hace algunos años, el desarrollo de este tipo de aplicaciones bajo la filosofía del software libre, presentaba muchas desventajas con respecto a las soluciones del software privativo; puesto que no existía una tecnología que pudiera asumir la confección de un producto de alta calidad. Afortunadamente OpenGL[1], VTK[2] y OpenCascade son una muestra de que ya se han implementado librerías y frameworks bajo licencias libres, capaces de sustentar productos tan profesionales como los privativos. Igualmente existen entornos de desarrollo con la capacidad de generar aplicaciones gráficas muy potentes, en ese caso se pueden mencionar a GTK[3] y al framework de desarrollo Qt.

Qt posee una potente estrategia para trabajar la visualización en 2D y 3D, ya que utiliza OpenGL como recurso interno de la herramienta, además de que su arquitectura admite la integración con componentes de visualización de otras librerías. En consecuencia, permite crear un desarrollo fusionado a otros frameworks o

[1] Biblioteca Gráfica para representar y modelar objetos en 3D.
[2] Framework de visualización y modelado 3D Visualization Toolkit (VTK).
[3] Biblioteca para crear interfaz gráfica de usuario.

bibliotecas de visualización, explotando al máximo los recursos que estas ofrecen. OpenCascade[4] por otra parte, resulta una solución muy poderosa en la visualización y modelado en 3D, pues cuenta con un arsenal de algoritmos que simplifican enormemente el trabajo en este sentido. Actualmente las aplicaciones desarrolladas en Qt[5] vinculadas con OpenCascade, son diseñadas con un fin en específico; por tanto, no constituyen un soporte genérico integrado que contribuya como punto de partida para la confección de herramientas de tipo CAD.

Una forma de ejemplificar lo descrito anteriormente, es mencionando la problemática existente en el proyecto GeolMin, perteneciente al centro de desarrollo Geysed en la Universidad de las Ciencias Informáticas. En dicho centro, se pretende implementar un módulo de visualización para un software minero, desarrollado a partir de los frameworks Qt y OpenCascade, pero a la hora de comenzar a implementar, no se cuenta con una herramienta que integre a ambas tecnologías. Por todo lo expuesto anteriormente se puede concluir que no existe una herramienta para visualizar objetos tridimensionales en Qt, utilizando el framework OpenCascade(Garrido 2009),(the_QtOCC_team 2007).

Luego de un análisis de la problemática planteada anteriormente surge el siguiente **problema a resolver:** ¿Cómo realizar la representación de objetos en tres dimensiones en el framework de desarrollo Qt utilizando el framework de visualización OpenCascade? Con el fin de dar una solución al problema propuesta, se define como **objeto de estudio:** El proceso de desarrollo de un componente para la visualización en tres dimensiones para Qt, delimitando como **campo de acción**: El desarrollo de un componente visual para Qt utilizando el framework OpenCascade. Del mismo modo se identifica como **objetivo general:** Desarrollar un componente visual para Qt que permita visualizar objetos en tres dimensiones utilizando el framework OpenCascade.

En correspondencia con el objetivo general definido se propone la siguiente **idea a defender:** Con el desarrollo de un componente visual para Qt, se logra obtener un soporte que permita representar objetos en 3D utilizando el framework OpenCascade.

Con el propósito de lograr un desarrollo exitoso con este trabajo, se elaboran las siguientes tareas descritas a continuación:

- Caracterizar el proceso de visualización en 3D.
- Caracterizar las tendencias y tecnologías actuales a desarrollar.
- Especificar los requisitos funcionales del software.

[4] Framework de visualización y modelado 3D.
[5] Framework para desarrollar interfaz gráfica de usuario.

- Entregar todos los artefactos que se generan del desarrollo ingenieril del componente.
- Implementar las funcionalidades definidas para el componente.
- Validar el componente propuesto.

Como posibles resultados a alcanzar con el desarrollo de este trabajo están:

- Un componente para Qt que permita visualizar de objetos en 3D, utilizando el framework OpenCascade.
- La documentación técnica asociada al desarrollo de la herramienta anterior.

Para llegar a comprender la situación existente, se utilizaron los siguientes métodos científicos de la investigación:

Métodos Teóricos:

- **Analítico-Sintético:** Se realiza un estudio detallado de toda la bibliografía conformada por documentos y sitios Web, lo que posibilitó obtener una síntesis detallada de la misma.
- **Análisis Histórico Lógico:** Permite ejecutar la búsqueda correspondiente en relación con la posible existencia de un componente visual para Qt utilizando el framework OpenCascade.

Estructura de la investigación:

A continuación se procede a describir los 4 capítulos por los que está compuesta la presente investigación:

Capítulo # 1: Se desarrolla la “Fundamentación Teórica” donde se explican los conceptos asociados al dominio del problema.

Capítulo # 2: Se analizan las “Tendencias y tecnologías actuales” que son necesarias para el correcto desarrollo de este trabajo.

Capítulo # 3: “Análisis y Diseño” Se muestra toda la descripción del negocio.

Capítulo # 4: “Implementación y Pruebas” aborda todo el proceso de construcción y pruebas de la solución propuesta.

Capítulo 1 "Fundamentación teórica"

1.1 Introducción

Para comprender la trascendencia y el alcance del presente trabajo, es necesario conocer el significado de los principales conceptos y terminologías asociadas al dominio del problema; además de comprender con claridad, cada uno de los aspectos que componen la fundamentación teórica. Para sostener lo mencionado anteriormente, en este capítulo se refleja el contexto que rodea al objeto de estudio a investigar, así como la problemática que da lugar al surgimiento de esta investigación. Del mismo modo se procede a realizar un análisis de la evolución y características de cada una de las soluciones existentes en la actualidad.

1.2 Conceptos asociados al dominio del problema.

1.2.1 Proceso de visualización en 3D.

"Un proceso de visualización 3D, es la unión de una serie de operaciones con el fin de realizar el despliegue de un objeto tridimensional en un dispositivo gráfico de salida. Por lo general este objeto es generado mediante una serie de datos procesados por un motor gráfico 3D."(Carmona 2008).

1.2.2 Transformaciones.

Las transformaciones más típicas sobre el modelo son las de traslación, escalado y rotación. Por lo general, el objeto es centrado en el punto de coordenadas (0,0,0) del sistema, para realizar su escalado y rotación. Si el centro del objeto es C (cx, cy, cz), los puntos del objeto son inicialmente trasladados mediante T (−cx, −cy, cz).
M = Rotaciones * S (sx, sy, sz) * T (−cx, −cy, −cz)
La matriz denominada Rotaciones contiene únicamente rotaciones y al igual que los escalamientos, se realizan generalmente con el objeto centrado en el origen(Carmona 2008).

Traslación: La traslación de un punto P(x, y) consiste en mover el punto según un desplazamiento T (dx, dy): dx respecto al eje x y dy respecto al eje y, obteniendo un nuevo punto P'(x', y'). Además la traslación admite incrementos y disminuciones en movimientos relacionados con el eje seleccionado.
Por ejemplo, si se tiene un cuadrado (en el plano), se desplazan todos sus puntos según dx y dy. No es necesario realizar la transformación sobre todos los puntos del

cuadrado basta con trasladar los vértices del mismo y luego redibujar las líneas (interpolación lineal)(Carmona 2008).

Escalado: Los objetos pueden ser escalados (estrechados o estirados) según un factor sx respecto al eje x y sy respecto al eje y, es decir que admite incrementos y disminuciones en el tamaño del objeto(Carmona 2008).

Rotación: Un punto puede ser rotado respecto al origen en un ángulo α, ya que permite giros a favor y en contra de las manecillas del reloj. Esta transformación se define matemáticamente como: $X' = x.\cos\alpha - y.\text{sen}\alpha$, $y' = x.\text{sen}\alpha + y.\cos\alpha$

1.2.3 Proyecciones.

En general, las proyecciones transforman puntos en un sistema de coordenadas de dimensión (n) a un sistema de coordenadas de dimensión inferior. La proyección de objetos tridimensionales serán definidos por la intersección de líneas rectas que van desde un centro de proyección u ojo, hasta cada punto del objeto con un plano llamado plano de proyección (π). Esta clase de proyecciones se denominan proyecciones geométricas planares, porque la proyección es hecha sobre un plano en vez de una superficie curva y por usar líneas rectas para la proyección (proyectores) en vez de líneas curvas(Carmona 2008).

Perspectiva: la distancia del centro de proyección al plano de proyección es finita.

Paralela: la distancia del centro de proyección al plano de proyección es infinita.

- Se especifica la dirección de vista.
- Todos los rayos de proyección son paralelos.

Proyección perspectiva.

En la proyección perspectiva se tiene un origen de proyección, el ojo. Por simplicidad, está ubicado en el origen del sistema de coordenadas y mirando en dirección (0, 0, −1). El plano de corte cercano π =NCP[6] es perpendicular a la dirección de visualización, por lo que su vector normal N es justamente (0, 0, −1). La distancia del plano al ojo se denomina como NCP, pues es el único parámetro que debe especificar el programador para π. Así, la ecuación del plano viene dada por $z = -NCP$.

Aunque la proyección perspectiva no es una transformación lineal en coordenadas cartesianas, en coordenadas homogéneas sí lo es, por lo que resulta más fácil

[6] Distancia del plano al ojo

combinar la proyección con las otras transformaciones del proceso de visualización(Carmona 2008).

Características:

- Simula el comportamiento de una cámara o el ojo humano.
- Se incrementa el realismo de la imagen, manifestándose la sensación de profundidad.
- El tamaño de un objeto inversamente proporcional a la distancia del objeto al plano de proyección.
- No resulta de utilidad en el reconocimiento de formas y medición de longitudes.
    - Las distancias son falsas.
    - Las longitudes no se mantienen.
    - Las líneas paralelas no se conservan.
- El eje z representa la dirección de vista.
- El eje y representa la vertical del observador.
- El eje x representa la horizontal del observador(Carracedo 2010).

Proyección paralela.

Constituye una buena aproximación cuando el observador se encuentra alejado del objeto. En este enfoque, los rayos de visualización son paralelos, dado un punto Pe (xe, ye, ze) del objeto en coordenadas de ojo, su imagen en el plano es:
$[x , y , z ,1]t = [x , y ,-ncp,1]t$(Carmona 2008).

Características:

- Se elimina la componente z.
- Se pierde información sobre la profundidad.
- Las líneas paralelas permanecen paralelas.
- Los ángulos únicamente se mantienen en las caras paralelas al plano de proyección.
- Se calcula fácilmente y de forma inmediata.
- Su uso resulta muy frecuente en software de modelado, en que se muestran tres vistas simultáneas del objeto(Carracedo 2010).

1.3 Objeto de Estudio.

1.3.1 Descripción General.

La visualización de objetos 3D es un recurso de la informática en constante desarrollo y con un gran potencial, demostrado por su utilidad al hombre. A medida que se desarrolla esta tecnología, aumentan las ramas de la industria y la variedad de artefactos que la utilizan con el fin de facilitar la vida del hombre moderno. A pesar de su importancia, es muy común no percatarse del alcance o la complejidad que puede asumir todo este intento de representación de la vida real; pues la misma es el resultado de una intenso trabajo y conocimiento acumulado, que se ha convertido en una poderosa herramienta, muy utilizada por el hombre. La repercusión de la visualización en 3D es enorme, su utilidad llega desde la arquitectura hasta la medicina, pasando por muchas esferas de la ciencia, pero esta representación de imágenes dinámicas, va mucho más allá de reflejado en un monitor.

1.3.2 Descripción actual del dominio del problema.

A la hora de confeccionar un software CAD, se hace necesario utilizar un arsenal tecnológico que, evidentemente, facilita y simplifica el trabajo. Es en ese momento que entra en juego lo que se conoce por framework y librerías, los cuales son un conjunto de estándares y algoritmos, ya implementados con anterioridad por otros desarrolladores. Cuando se implementa un producto de este tipo, es común que se reutilice de alguna manera la tecnología ya existente. Independientemente de la técnica utilizada para programar nuestro software, con frecuencia se vincula el código de la aplicación, con tecnología correspondiente a este trabajo, por medio de un componente o plugins[7]. Bajo esta filosofía se desarrollan la mayoría de estos softwares y aunque no es el único prototipo de trabajo, es una estrategia utilizada con frecuencia por los implementadores y que con seguridad perdurará por muchos años.

Evolución de los sistemas CAD.

El trabajo y desarrollo con un software de tipo CAD, es una de las habilidades más usadas en la informática gráfica, pues su adecuada manipulación resulta un recurso indispensable para cualquier institución vinculada a la visualización gráfica. Este desarrollo comienza en los años 50, cuando el ejército norteamericano crea los primeros trazadores gráficos (plotters) los cuales eran capaces de representar dibujos diseñados por un ordenador. Por esa misma fecha sale el primer software (SKETCHPAD), el cual era capaz de realizar los primeros dibujos utilizando pixeles, el

[7] Es una aplicación o complemento que le proporciona una función nueva y generalmente muy específica a nuestro software.

mismo fue creado en el Massachusetts Institute Of Technology (Instituto de tecnología de Massachusetts).

De igual manera la expansión de la tecnología CAD en las industrias tuvo sus inicios en los años 60, dejando como consecuencia la invención del lápiz óptico a medida que se fue perfeccionando el hardware gráfico. A pesar de esto no se estableció totalmente hasta una década después, con la propagación de los computadores personales, los que cada vez eran más pequeños y fáciles de utilizar. En las décadas posteriores se continuarían desarrollando estos sistemas, aparecerían los modelos en dos y tres dimensiones (2D y 3D), además del diseño paramétrico y varacional respectivamente.

Los softwares de tipo CAD se generalizan en 4 clasificaciones, inicialmente existe una categoría que se encargan de la representación en 2D, la cual manipula esencialmente vistas, acotaciones y cortes. En una segunda categoría se pueden encontrar, los que permiten el trabajo con dos y tres dimensiones, estos además de realizar un trabajo en 2D, admiten un diseño de tres dimensiones aunque este último no es muy sofisticado. Como ejemplos que utilizan este patrón de trabajo se encuentran AutoCAD[8] y MicroStation[9]. Dentro de una tercera categoría se encuentran los encargados de manipular un modelado en 3D de gama media, caracterizándose por una interfaz amigable y fácil de utilizar, mientras que los de gama alta entrarían en una cuarta categoría, los cuales están diseñados para un trabajo más profesional que los de gama media(González 2004).

Frameworks y librerías de diseño gráfico.

El desarrollo de software para visualizar objetos en tres dimensiones se ha diversificado a una velocidad increíble, cada vez es más grande el número de aplicaciones y bibliotecas diseñadas con este objetivo. Hace unos años se podía decir que de cierta manera este punto del mercado, era totalmente dominado por determinados productos o sectores del software privativo, pero en la actualidad la realidad es muy distinta. En el mundo del software libre existen varias muestras con un potencial enorme, que en ocasiones pueden igualar o superar a sus similares del software privativo.

Librerías gráficas.

Las librerías gráficas son un conjunto de clases con miles de líneas de código, que permiten representar diversos tipos de imágenes en un ordenador, están constituidas por modelos matemáticos, patrones de textura, iluminación y colores. Generalmente se encargan de eliminar una dependencia con un tipo de hardware en específico, es

[8] Programa de tipo CAD en 2D y 3D
[9] Programa de tipo CAD

decir, que garantizan un funcionamiento en diversos tipos de tarjetas gráficas al mismo tiempo.

Clasificación de las librerías gráficas:

- **Librerías de rendering directo:**

Para el uso de estas librerías se necesita tener un amplio conocimiento de hardware de representación de 2D y 3D, por tanto constituye un proceso de desarrollo más largo. En este caso el implementador debe de tener conocimientos sobre la gestión de carga/descarga, el uso del buffer gráfico. Dentro de este grupo de librerías se encuentran: OpenGL, Direct3D(Borroso 2009).

- **Librerías de alto nivel o descriptivas:**

Estas librerías permiten un desarrollo más rápido que las anteriores, pues la implementación se simplifica debido a que la representación de las imágenes se sustenta mediante un árbol de escena. Además la librería de manera autónoma, gestiona la carga y descarga de memoria, además del uso del buffer gráfico, con el fin de ahorrar recursos del sistema y tiempo de procesado. Esta categoría se caracteriza además por presentar un conjunto de funciones y clases, generalmente escritas en C/C++ como paradigma de Programación Orientada a Objetos (POO). Además presenta capas de interfaces (bindings[10]), en caso de que se requiera utilizar otro lenguaje de programación. En este grupo se pueden encontrar a: VTK, OpenCascade, Matpack, OpenInventor (Borroso 2009).

Direct3D.

Esta librería se trata de una API para facilitar la implementación y procesado de productos con gráficos 3D, forma parte de las DirectX[11]. Además permite la manipulación y transformación de elementos como: líneas, polígonos, texturas. Directc3D pertenece a Microsoft y se considera además como una capa intermedia entre una aplicación y los drivers de tarjeta gráfica (Borroso 2009).

Open Graphics Library (OpenGL).

"OpenGL es una de las alternativas libres más usadas, esta es una biblioteca de primitivas, comandos o rutinas para situar objetos en el espacio" (Borroso 2009). Como casi todas las herramientas en el Diseño Asistido por Computadora, está orientada al desarrollo de aplicaciones interactivas, con el fin de diseñar interfaces

[10] Adaptación de una biblioteca para ser usada en otro lenguaje de programación distinto.
[11] Librería gráfica creada por Microsoft

gráficas para el manejo de 2D y 3D. A la hora de trabajar con esta librería, el implementador necesita conocer de antemano todo el proceso a desarrollar, antes de llevar a cabo el manejo y la renderización de la imagen tridimensional. Aunque no se le puede considerar una biblioteca de alto nivel, es innegable que ha prevalecido en esta industria (Borroso 2009).

VTK.

Otra de las herramientas muy utilizada es Visualization Toolkit (VTK), este framework simplifica la implementación, debido a los algoritmos que tiene implementado, a la forma de representación y a su modo de procesamiento de datos. Además tiene la ventaja de ser multiplataforma y la inmensa capacidad que posee de leer y exportar varios formatos. El uso de VTK abarca casi todos los sectores relacionados con la representación en tres dimensiones, utilizándose frecuentemente en la medicina y en la industria petrolera (Borroso 2009).

OpenCascade.

El framework OpenCascade es otra de las soluciones muy utilizadas en la visualización en 3D, además de ser otro producto multiplataforma desarrollado bajo la filosofía del software libre y que goza de mucha popularidad entre los programadores. El éxito de OpenCascade lo justifica las propias ventajas que posee la herramienta. Primeramente su paquete de clases brinda una serie de servicios tales como: tipos primitivos, cadenas y varios tipos de cantidades. Además ofrece una gestión automatizada de la memoria acumulada, manejo de excepciones, clases de manipulación de agregados de datos y herramientas matemáticas. Igualmente este framework suministra las estructuras de datos para representar modelos geométricos en dos y tres dimensiones, estas funcionalidades se encuentran localizadas en las clases: geometría 2D, geometría 3D, utilidades geométricas y topologías (opencascade.org 2007).

Qt & OpenCascade.

A la hora de analizar la evolución relacionada con la representación en tres dimensiones, es difícil dejar de notar que cada una de estos productos, por separado, forma parte de un motor gigantesco que no para de desarrollarse, trazándose metas más ambiciosas. Del mismo modo para este tipo de actividad el framework de desarrollo Qt resulta una herramienta noblemente robusta, abalada por su propia historia y por el inmenso número de excelentes proyectos desarrollados sobre ella (Garrido 2009).

La vinculación de alguna biblioteca o framework de desarrollo de productos CAD con Qt, es una ruta que ya tiene sus huellas, puesto que existen librerías que poseen un

componente visual que la vincula con el framework directamente. Como ejemplo se puede mencionar QVTKWidget, que es el componte correspondiente para Qt del framework VTK, el cual utiliza el Widget generado para crear su interfaz gráfica o panel sobre el cual visualizar los objetos en 3D. Esta estrategia de trabajo sostiene una ventaja fundamental, es la reutilización de los millones de líneas de códigos que contienen estas librerías, lo cual simplifica enormemente el tiempo de confección de cualquier producto (wtk.org 2010).

De igual modo, OpenCascade es una herramienta diseñada para el desarrollo rápido de aplicaciones, orientadas a la visualización y modelado en 3D asistido por ordenador, que de cierto modo ya tiene su historia. Esta solución tiene sus inicios en los años 90 y también sigue la filosofía del software libre, por lo que goza de la preferencia de muchos usuarios. La estructura de las clases de OpenCascade, están escritas en C++, además de ser una herramienta multiplataforma.

OpenCascade es muy utilizado por los desarrolladores del software libre, por lo que existen una infinidad de estrategias a la hora de manipular la herramienta, con el fin de vincularlo a otra herramienta de desarrollo. Se pueden encontrar softwares que utilizan OpenCascade y al mismo tiempo están utilizando otro framework de desarrollo como VTK, en este caso está el software Salomé[12]. Se hace necesario mencionar que generalmente, son desarrollados en frameworks o plataformas para generar interfaz, como son el caso de GTK o Qt (opencascade.org 2007).

1.3.3 Situación Problemática.

El constante perfeccionamiento de OpenCascade y Qt ha rendido sus frutos, pues con la unión de estas dos tecnologías se han podido implementar productos de una excelente calidad. Estos han sido creados con el fin de modelar objetos tridimensionales dirigidos hacia un objetivo en específico o hacia un sector particular de la industria. Por tanto ninguna cumple las características como componente visual que sirva de base para implementar una nueva aplicación. Por tanto el no contar con la existencia de una alternativa que vincule a Qt con OpenCascade, resulta un inconveniente a la hora de confeccionar un software con funcionalidades específicas.

Una manera de ilustrar la problemática planteada anteriormente, es reflejando lo que ocurre en el centro de producción GEYSED de la Universidad de las Ciencias Informáticas. El centro de producción tiene incluido dentro de sus líneas de trabajo, la producción de software de minería a partir de los frameworks Qt y OpenCascade. En el momento de comenzar a implementar un módulo para la representación en tres dimensiones de objetos geológicos, utilizando las tecnologías mencionadas

[12] Aplicación de tipo CAD

anteriormente, no se cuenta con un componente para Qt que pueda asumir la implementación del módulo.

1.4 Análisis de las soluciones existentes.

La producción de software de tipo CAD a nivel mundial está muy diversificada. En el mundo constantemente aparecen soluciones muy novedosas que superan o aportan elementos a la producción mundial de este de tipo de software. Independientemente de las tecnologías utilizadas, existen software especializados en la minería y al mismo tiempo se pueden encontrar soluciones orientadas a la arquitectura, la mecánica o incluso diseñados para la medicina o el audiovisual.

A la hora de realizar un análisis de la producción de softwares de tipo CAD hasta nuestros días hay que mencionar la controversia existente entre el software privativo y el software libre. El software privativo al ser desarrollados por empresas muy bien respaldadas económicamente, siempre se ha caracterizado por ser una solución muy potente o profesional, pero al mismo tiempo traen como principal inconveniente el problema de las licencias y la dificultad económica que esto representa. Por otro lado, existen otras alternativas que demuestran de que se pueden realizar proyectos muy profesionales bajo la filosofía del software libre.

1.4.1 FreeCad.

Es un modelador de 3D, desarrollado a partir de OpenCascade, Python y Qt, generalmente está dirigido hacia la ingeniería mecánica, pero también es utilizado en otras ramas como es la arquitectura. Esta aplicación cuenta con puntos de contactos con herramientas como: Catia, Solid Works y Solid Edge, por lo que se puede clasificar en: CAD/CAM[13] y CAE[14]. Entre de las principales ventajas de del software se encuentran:

- Es una aplicación desarrollada bajo la filosofía del software libre.
- Cuenta con una arquitectura modular, esto se traduce en que se le pueden adicionar otros módulos adicionales sin modificar su sistema central.
- Cuenta con una serie de módulos que abarcan la mayoría del proceso de modelado en 3D.

[13] Fabricación asistida por computadora, también conocida por las siglas en inglés CAM (Computer Aided Manufacturing).
[14] Ingeniería asistida por computadora Computer Aided Engineering

Los principales módulos de este producto son:

Meshes (Mayas): Este módulo es el encargado de importar, reparar y convertir mayas diseñadas en otras aplicaciones para poder manipularlas en la aplicación. Permite trabajar con una gran variedad de formatos como son: *.stl, *.ast, *.obj, *.nas, *.iv y *.bms., además permite exportar a: *.stl, *.ast, *.obj, *.nas, *.brl, *.wrl, *.bms y *.py. Además en este módulo se puede trabajar con herramientas de pruebas y reparación de mayas.

2D Drafting (Dibujo 2D): En esta parte de la aplicación es donde se manejan las superficies planas, en esta sección se puede crear cualquier tipo rectángulo, circunferencia o cualquier otra superficie en 2D y aplicarlo sobre cualquier plano en tres dimensiones. También en esta parte se permite introducir a los modelos de textos, o realizar operaciones de modificación además de exportar a los siguientes formatos: (*.dxf), Open Cad Format (*.oca, *.gcad) y SVG (*.svg).

CAD: En este módulo, es donde se realizan cualquier tipo de funcionalidad relacionada con el modelado en tres dimensiones. Permite el diseño de formas paramétricas (cilindro, esfera, cubos) y de componentes topológicos (vértices, aristas, planos) además de realizar las operaciones booleanas correspondientes (interpolación, intercepción) permite exportar a los siguientes formatos: (*.stp,*.step, *.igs, *.iges, *.brp).

Raytracing: El principal objetivo de este Módulo es la exportación de la geometría de FreeCad, además de la renderización para crear imágenes de alta calidad.

Drawing (Dibujo). En este módulo es donde se permite exportar cada una de las proyecciones de vistas de la geometría en 3D para la generación de una plantilla svg (Aik-Siong Koh 2007).

1.4.2 Salomé.

Se trata de una plataforma genérica, desarrollada sobre Qt y la fusión de los frameworks VTK y OpenCascade. El objetivo fundamental de esta aplicación es el pre y pos procesado de simulaciones numéricas, está desarrollado sobre la licencia pública general para bibliotecas de GNU (LGPL), bajo la filosofía del código abierto. Entre las principales ventajas del producto se encuentran:

- Está desarrollada bajo filosofía del software libre, utilizando la licencia (LGPL), lo que permite tener acceso total al código.

- Está diseñada en una estructura modular, permitiéndole al implementador agregarle otros módulos posteriormente sin modificar el sistema, lo cual hace al producto muy flexible.

Como desventaja de este software se puede mencionar:

- No es multiplataforma, por lo que limita al usuario a un solo sistema operativo, aunque las características de su licencia permita hacer modificaciones al código fuente, con lo cual se podría resolver este problema (Ferreiro and Rodríguez 2009).

1.4.3 QtOpenCascade ó QtOCC.

Está desarrollada sobre Qt y OpenCascade que tuvo sus inicios en 1999. En su interior abarca algunas de las funcionalidades relacionadas con la visualización y modelado de imágenes en 3D. Su principal objetivo es la visualización de sólidos utilizando las bibliotecas de OpenCascade a través de una interfaz generada en Qt.

Como ventajas del software se pueden mencionar:

- Es multiplataforma pues la aplicación puede correr sobre: Linux, Solaris y Windows.
- Está desarrollada bajo la filosofía del software libre, lo que garantiza una libre distribución y acceso al código.

Como desventajas se pueden mencionar:

- No es un software sofisticado, por lo que sus funcionalidades son básicas, además no cuenta con una barra de herramientas, en su lugar sólo cuenta con 4 menús sobre los que se pueden realizar operaciones como: zoom, rotaciones, entre otras.
- Este producto básicamente es sólo un visor básico para BREP[15], STEP[16] e IGES[17], por lo que resulta un problema a la hora de tratar con otros formatos, además de que no permite la creación ni edición de diferentes tipos de geometrías (the_QtOCC_team 2007).

[15] Formato de datos 3D en ingles (Boundary Representation).
[16] Formato de datos 3D en ingles (*Standar for the Exchange of Product model data*).
[17] Formato de datos 3d en ingles(Initial Graphics Exchange Specification).

QtOCC es lo que más cercano a un componente de OpenCascade para Qt, ya que solamente genera una interfaz modesta con funcionalidades similares a las propuestas por los compontes de visualización para Qt existentes hasta el momento. A pesar de las desventajas notables que presenta esta aplicación, resulta una buena referencia para el desarrollo de esta investigación, pues explota de una manera exitosa la integración de OpenCascade con Qt. Igualmente QtOCC genera una interfaz muy básica a partir de la manipulación de los Widget de Qt, lo que representa otro punto en común con componentes como QVTKWindget[18] y por tanto constituye otra estrategia de trabajo para el desarrollo de esta investigación.

1.5 Conclusiones Parciales.

Con la realización de este capítulo se puede concluir que al mencionar cada uno de los elementos y conceptos asociados al dominio del problema, se contribuye a un mayor entendimiento de la problemática en cuestión. De igual forma el presentar los elementos que conforman nuestro objeto de estudio permite profundizar en la complejidad que contiene todo el campo de acción a estudiar. Luego de analizar cada una de las soluciones existentes hasta el momento relacionadas de alguna manera con este trabajo, se determina que la herramienta QtOCC es la más cercana al objetivo que persigue esta investigación debido a sus características de visor 3D.

[18] Componente visual del framework VTK para Qt.

Capítulo 2: Tendencias y Tecnologías Actuales.

2.1 Introducción.

El desarrollo alcanzado por las TICs, trae consigo un continuo perfeccionamiento de las herramientas informáticas, por tanto, al implementar cualquier tipo de software, se hace necesario realizar un estudio detallado en relación a cualquier recurso a utilizar. En el presente capítulo se describen las tecnologías y herramientas empleadas en la presente investigación, realizando un análisis de las principales características, ventajas y desventajas de cada una de ellas, con el fin de realizar una mejor selección que pueda sostener el desarrollo de la solución propuesta.

2.2 Desarrollo de aplicaciones informáticas Libres y Privativas.

La revolución informática iniciada hace medio siglo, está directamente relacionada con el desarrollo actual y futuro de la humanidad; el desarrollo de aplicaciones informáticas tiene como principal objetivo, proporcionar una herramienta al usuario para realizar uno o varios tipos de trabajo. Gracias a su enorme utilidad, puede llegar desde el proceso de automatización de alguna complicada tarea, hasta constituir alguna herramienta de oficina como son los procesadores de texto y hojas de cálculo. El principal dilema en relación a este tema, está relacionado con las filosofías de software libre y propietario, pues de ahí se derivan varios aspectos como: las patentes, la distribución, el costo entre otras, todas de interés tanto para las empresas, como para cualquier tipo de usuario.

La filosofía del software propietario o privativo se basa en la el desarrollo de cualquier tipo de programa informático, en el que los usuarios tienen limitadas posibilidades de usarlo, distribuirlo o modificarlo, además de que el código fuente no está disponible. En esta filosofía una persona física o jurídica posee los derechos de autor de dicha aplicación, negando u otorgando los derechos de uso de la misma con cualquier propósito, de modo que aún cuando se haga público el código fuente, no se puede modificar o distribuir este conocimiento.

La manipulación de cualquier tipo de tecnología privativa, tiene innegables ventajas, ya que por lo general estos productos son desarrollados bajo un estricto control de calidad y una inmensa difusión en el mercado. Por otra parte se puede mencionar que estas son desarrolladas por un personal muy calificado y bajo una profunda investigación en relación al uso que le da el usuario, con la finalidad de hacerlo un producto más competente. Esta táctica tiene también su lado oscuro o grandes desventajas, pues por ejemplo, es ilegal realizar cualquier tipo de distribución, copia o modificación del código de la aplicación sin los permisos del propietario. La principal

causa de esto es que el principal objetivo de estas empresas es esclavizar al consumidor a sus productos. De modo que no se pueden realizar ningún tipo de mejora o aporte al producto y es aquí donde el software libre comienza a ganar terreno (Culebro Ju´arez, G´omez Herrera et al. 2006).

¿Por qué utilizar realizar una aplicación bajo software libre?

El software libre presenta varios inconvenientes en relación con las tecnologías privativas, puesto que: la garantía, la interfaz gráfica, la lentitud de aprendizaje, además de que para su configuración se necesita tener una mínima noción de programación, son aspectos a considerar por el usuario estándar. En esta filosofía se destaca la luz por encima de las manchas, ya que son sus rasgos positivos los que han logrado que cada vez gane más seguidores. En primer lugar se puede mencionar su bajo costo y libre uso, a diferencia de su contraparte, pues en este tipo de tecnología se puede disponer del código fuente de la aplicación y por tanto hacer uso pleno de la misma.

Como un rasgo notable son los bajos requisitos de hardware, además de la innovación tecnológica pues en el software libre se comparte la información en forma de comunidad, por tanto, constantemente se realizan mejoras que están al alcance de todos. Otra importante particularidad es la personalización, pues gracias a que se cuenta con el código fuente, el usuario puede realizar su propia versión del mismo.

Luego del anterior análisis, se puede llegar a la conclusión, que el desarrollo de una aplicación bajo tecnología y filosofía libre, es el camino a seguir para la futura implementación del componente propuesto, pues se estiman necesarios cada uno de los beneficios que brinda esta filosofía. Por otra parte, se considera que toda la documentación e investigación, se nutrirán con la información disponible en relación con el tema que ocupa el presente trabajo.

2.3 OpenCascade como framework de visualización en 3D.

OpenCascade es framework diseñado para el modelado en 3D asistido por ordenador, fue diseñando originalmente a finales de los años 90 por MatraDataVision. Incluye componentes para la superficie 3D, modelado de sólidos, visualización, intercambio de datos y el desarrollo rápido de aplicaciones. Esta aplicación está escrita en C++ y se caracteriza por ser multiplataforma ya que puede correr sobre, Windows, Linux y Solaris, además de que se puede clasificar como una aplicación de tipo: CAD, CAM[19], CAE[20]. Actualmente se pueden encontrar toda la documentación referida a este software en: ***www.opencascade.org***, mientras que la aplicación se encuentra

[19] Fabricación asistido por computadora
[20] Ingeniería asistida por computadora *Computer Aided Engineering*

disponible en el repositorio BlogDrake[21], donde se encuentra todo el paquete como una potente aplicación CAD.

"WOK (Taller de Organización Kit) es el ambiente de desarrollo de OpenCascade, que permite a un gran número de desarrolladores trabajar en una variedad de productos en forma simultánea. En él los desarrolladores pueden producir varias versiones de los productos de hardware y plataformas de software diferentes, incluidas las versiones correspondientes a las necesidades específicas de comercialización. Al mismo tiempo permite la reutilización de componentes de software con el máximo de efecto posible, en otras palabras el diseño está orientado a facilitar el desarrollo a escala industrial" (opencascade.org 2007).

Resumen de topología:

La topología de OpenCascade está diseñada con referencia a la norma ISO-PASO 10303-42.

La estructura es un gráfico orientado a un solo sentido, donde la clases padres se refieren a sus hijos y no hay referencias anteriores. La estructura se implementa como clases escritas en C + +.

Ventajas de OpenCascade:

Acceso al código fuente:

El acceso a la calidad del código ofrece mayor estabilidad y robustez para OpenCascade soluciones basadas en tecnología y aplicaciones. El código puede ser libremente adaptado, modificado y enriquecido por la funcionalidad necesaria de acuerdo a las necesidades particulares.

No hay derechos de licencias:

No hay limitación en cuanto al número de copias instaladas, lo que significa reducción de costes en gran medida. La ausencia de derechos de licencia reduce los costos del período de evaluación, creación de prototipos y proyectos piloto.

La continuidad del producto:

Producto de la evolución permanente y continuidad están garantizados por OpenCascade SA, los clientes industriales de apoyo a su propia OpenCascade con soluciones basadas en la tecnología y la comunidad en todo el mundo.

[21] Blog para la distribución de Linux Mandrake

Fiabilidad:

Una gran comunidad de usuarios y los clientes están constantemente probando el software. El aumento de la calidad y robustez del software está garantizado por el desarrollador inicial (OpenCascade SA).

La apertura y la interoperabilidad:

OpenCascade ofrece amplias posibilidades de intercambio de datos, a través de formatos neutros y directamente con los sistemas CAD. Esto permite la viabilidad de las soluciones abiertas.

Soporte comercial completo:

Para los editores de software comercial en busca de una completa plataforma de desarrollo y de expertos de apoyo, la tecnología OpenCascade es una combinación ideal pues paga por el soporte y no para los derechos de licencia. Los servicios son muy flexibles e incluyen numerosas opciones.

2.4 Lenguajes de programación.

El surgimiento y desarrollo de los lenguajes de programación y las computadoras están estrechamente relacionados, de modo que luego de 50 años de desarrollo informático, existen más de 2500 lenguajes documentados (Restrepo 2010).

Características de los lenguajes de programación:

- La programación se define como una actividad general del hombre, que significa la acción de extender o cambiar la funcionalidad de un sistema.
- Programar es decirle a un computador (o a alguna máquina) como realizar su trabajo.
- La programación es una actividad de amplio espectro realizada tanto por no especialistas como por especialistas.
- La programación (de sistemas de software) consta de dos partes esenciales: la ciencia y la tecnología.

Clasificación de los lenguajes de programación:

- Lenguaje de máquinas: Son aquellos que están escritos en lenguaje inteligibles por las computadoras, pues sus instrucciones están dadas en números binarios.

- Lenguaje de bajo nivel: Estos son más fáciles de entender que los lenguajes de máquina, pero al igual que ellos de alguna manera dependen de la computadora, el lenguaje de bajo nivel por excelencia es el ensamblador.
- Lenguaje de alto nivel: Están diseñados para que los programadores los escriban y entiendan con más facilidad que los lenguajes de máquina y ensambladores y presentan como principal característica que no dependen de ninguna arquitectura de hardware en específico.
- Lenguajes visuales: Surgen con la llegada de la informática gráfica, descienden directamente de los lenguajes tradicionales como Pascal, Basic y C, pero tienen como principal característica que utilizan una interfaz más amigable gracias a los componentes visuales (Restrepo 2010).

Los principales paradigmas de programación son:

- Declarativos (Funcional, Lógico, Por Restricciones).
- Imperativo.
- Orientado a Objetos.
- Concurrente.
- Orientado a aspectos.
- Orientado a agentes.

2.4.1 Java.

Java es un lenguaje de programación orientado a objetos creado a principios de los años 90 por la empresa Sun Microsystems, que presenta como principales rasgos ser un lenguaje robusto, por incluir soporte para el trabajo en la red, además de permitir el trabajo en múltiples sistemas operativos. Contiene en su sintaxis algunos elementos heredados de C++, pero al mismo tiempo fue desarrollado con el objetivo de eliminar algunas de las complejidades del mismo, pues contiene un recolector de basura que optimiza en gran medida la gestión de memoria, a diferencia de C++. Es un potente lenguaje que se destaca por no permitir la generación de código maligno, del mismo modo que presentar una curva de aprendizaje muy rápida, pues cualquier usuario familiarizado con lenguajes como C++, lo entendería con facilidad. La tendencia hacia la cual se proyecta Java en un futuro, está dirigida hacia el desarrollo de ambientes reducidos y vasados en red, particularidad muy empleada en dispositivos electrónicos como: tarjetas inteligentes y celulares (Galiano 2007).

Ventajas de Java:

- Fácil de aprender.
- Orientado a objetos.
- Independiente de plataforma.
- Es un lenguaje seguro.
- Es un lenguaje robusto.
- Es portable.

Desventajas de Java:

- Como su máquina virtual es un intérprete, presenta algunos problemas en relación al rendimiento y la velocidad.

2.4.2 Python.

Python es un lenguaje de programación interpretado y orientado a objetos, creado a finales de los años 80 por Gido Van Rossum y que surge como sucesor del lenguaje ABC. Python presenta como principal objetivo la facilidad tanto en lectura como en diseño, de igual forma es un lenguaje dinámico, de libre distribución que se caracteriza además, por ser multiplataforma y multiparadigma, pues permite además de la programación orientada a objetos, la programación estructural y funcional. Además presenta gran soporte e integración con otros lenguajes y la integración de varias bibliotecas estándar. Igualmente se caracteriza por estar fuertemente tipado[22] y por considerarse como un lenguaje de alto nivel, gracias a que presenta un tipado dinámico[23]. Debido a estas ventajas, el número de adictos al Python es mayor, además de que plataformas como YuoTube y Google, son una prueba incuestionable de su fortaleza (Downey, Elkner et al. 2002), (Jones 2008).

Ventajas que ofrece Python:

- Rápido de desarrollar.
- Sencillez y velocidad.
- Sus bibliotecas hacen gran parte del trabajo.
- Soporta varias bases de datos.

Desventajas de Python:

- Los programas interpretados son más lentos que los compilados.

[22] Cuando se define un tipo de dato en la definición de una variable, pues el chequeo de tipado se realiza en tiempo de compilación.
[23] Cuando una variable puede tomar distintos tipos de datos, pues el chequeo de tipado se realiza en tiempo de ejecución.

2.4.3 C++.

C++ es un lenguaje de programación orientado a objetos que surge en los años 80 por Bjarne Stroustrup, surge como una ampliación del lenguaje C, con el objetivo de poder manipular objetos. Una vez añadida la programación genérica incorpora otros paradigmas de programación (programación estructurada y orientada a objetos); por lo que se puede considerar como un lenguaje multiparadigma. El lenguaje es calificado como uno de los más potentes, ya que permite trabajar la programación a alto y bajo nivel. En C++, el manejo de la sobrecarga de operadores es uno de sus rasgos más atractivos pero de igual modo, el trabajo con punteros puede resultar una fortaleza o una fuente de errores, por lo que otros lenguajes descendientes del C++, como C# y Java han eliminado esta característica (Prata 2002), (Schildt 1995).

Ventajas de C++:

- Es un lenguaje muy versátil.
- Es muy potente a la hora de realizar sistemas complejos.
- Se puede compilar código de C.
- Es un lenguaje muy empleado y con mucha documentación.
- Existen muchos algoritmos y librerías implementadas en C++, por lo que se pude adaptar fácilmente.

Desventajas de C++:

- El trabajo con punteros y el manejo de memoria pueden convertirse en una potencial fuente de errores.
- No es recomendable para el desarrollo de páginas web.
- El trabajo con librerías dinámicas puede convertirse en problema a la hora de manejar la memoria.

2.4.4 ¿Por qué utilizar C++?

Luego de valorar algunos de los lenguajes de programación vinculados total o parcialmente con los objetivos trazados por la presente investigación y con el fin de lograr una eficiente implementación del componente en cuestión, se decide seleccionar a C++ como la mejor opción existente. Con el propósito de argumentar la anterior selección, se puede mencionar que: en primer lugar C++ es un lenguaje utilizado por varias herramientas libres de desarrollo, sobre las cuales se han implementado disímiles herramientas, relacionadas con el resultado a esperar, lo que evidencia la fortaleza y versatilidad del mismo. Otro aspecto a importante a mencionar es que cada una de las clases que componen las bibliotecas del framework de

visualización OpenCascade, están escritas en C++. Del mismo modo el framework de desarrollo Qt, utiliza a C++ como lenguaje de desarrollo, por tanto constituye el denominador común entre ambas tecnologías y por ende, aparece como la mejor estrategia para lograr una fusión entre ellas. Independientemente de los inconvenientes que puede aportar la utilización de C++, se impone como una opción versátil, potente para desarrollar cualquier tipo de sistema y con una enorme documentación en la red, lo cual es otro aspecto que lo reafirma como el indicado para utilizar en esta investigación.

2.5 Qt como framework para desarrollar interfaz de usuario.

Qt es un framework multiplataforma para desarrollar interfaces gráficas de usuario, desarrollado por la empresa Trolltech en 1992, pero no fue hasta 1995 que se puso a disposición pública. En la actualidad junto a las bibliotecas GTK y mxWidgets, está incluida entre las herramientas libres más potentes para el desarrollo de casi cualquier tipo de software. Como uno de los rasgos principales que presenta de esta herramienta, es que utiliza como lenguaje nativo a C++, lo cual permite generar aplicaciones sustentadas en la fortaleza de este lenguaje. Del mismo modo Qt contiene una serie de librerías y clases con una gran herencia de programación orientada a objetos, que permite enriquecer el desarrollo, ya que simplifican algunas de las deficiencias del lenguaje C++. Otra característica notable es la excelente documentación disponible, lo cual es muy agradecido para cualquier usuario que comience a trabajar con Qt. Cuenta además con una arquitectura lista para plugins, lo que permite el desarrollo mediante la integración con otro tipo de productos, como son los frameworks de visualización en 3D.

Como un inconveniente que puede mencionar, es el tema de las licencias, puesto que Qt posee dos versiones (Qt comercial edition, Qt Open Source edition), pero de igual manera siempre habrá una versión con el código fuente disponible, todo depende del tipo de aplicación a desarrollar. C++ es el lenguaje de programación nativo de esta herramienta, pero también posee la posibilidad del desarrollo bajo otros lenguajes, esto es posible mediante los bindings, entre las que se encuentran:(**PythonQt** bindings para Python, **Qyoto** Bindings para C# u otros lenguajes .NET, **QtRuby** Bindings para Ruby, **QtJambi** Bindings para Java, **PHP-Qt** Bindings para PHP). Actualmente, algunos de los productos desarrollados en Qt son: Adobe Photoshop Album (aplicación para organizar imágenes), Google Earth (simulador de mapas en 3D), Psi (cliente de mensajería instantánea para XMPP), VCL Media Player (reproductor multimedia de código abierto), KDE (popular entorno de escritorio para

sistemas operativos tipo-Unix), los cuales constituyen una muestra de la versatilidad y fortaleza del framework (Meyer 2007), (Summerfield 2008).

Componentes que integran el framework Qt:

- **Las librerías Qt :**(clases en C++).
- **QtDesigner:** Para crear formularios visualmente utilizando QtAssistant que permite un acceso rápido a la documentación.
- **QtLinguist:** Traducción rápida de programas.
- **Qmake:** Simplifica el proceso de construcción de proyectos en las diferentes plataformas soportadas.
- **QtCreator:** Es un Entorno Integrado de Desarrollo o IDE (esto es, editor + compilador + depurador) bastante completo, moderno, potente, fácil de manejar, eficiente, abierto y gratuito, que permite el desarrollo rápido de aplicaciones en MS Windows, Mac OS y Linux (Summerfield 2008).

2.6 Lenguaje Unificado de Modelado.

El Lenguaje Unificado de Modelado (UML) nace con el objetivo de lograr una representación de un proyecto informático, logrando establecer un estándar para describir un plano del sistema. Además incluye aspectos conceptuales tales como, procesos del negocio y funciones del sistema, así como aspectos concretos como expresiones de lenguajes de programación, esquemas de base de datos, y componentes de software reutilizables. Por tanto “UML es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar un sistema de software”. Define una notación que” permite modelar sistemas de información y su objetivo es lograr modelos que, además de describir con cierto grado de formalismo tales sistemas, puedan ser entendidos por los clientes o usuarios de aquello que se modela” (Booch, Rumbaugh et al. 2000).

Funciones de UML:

- Visualizar: Refleja gráficamente un sistema de modo que pueda ser entendido por otro sistema.
- Especificar: Definir las características de un sistema antes de su construcción.
- Construir: A partir de modelos construye los sistemas diseñados.
- Documentar: Los elementos gráficos constituyen la documentación del sistema, con el fin de que puedan ser revisados posteriormente (Booch, Rumbaugh et al. 2000).

Diagramas de UML

Para poder representar correctamente un sistema, UML ofrece una amplia variedad de diagramas para visualizar el sistema desde varias perspectivas, los cuales se detallan a continuación:

Los Diagramas de Estructura enfatizan en los elementos que deben existir en el sistema modelado.

- Diagrama de Clases.
- Diagrama de Objetos.
- Diagrama de Componentes.
- Diagrama de Despliegue.

Los Diagramas de Comportamiento enfatizan en lo que debe suceder en el sistema modelado:

- Diagrama de Estados.
- Diagrama de Actividades.
- Diagrama de Casos de Uso.

Los Diagramas de Interacción son un subtipo de diagramas de comportamiento, que enfatizan, estas, sobre el flujo de control y de datos entre los elementos del sistema modelado:

- Diagrama de Secuencia.
- Diagrama de Colaboración.

Los diagramas más interesantes (y los más usados) son los de casos de uso, clases y Secuencia (Orallo 2006).

Aunque la metodología XP no plantea la utilización de diagramas como un elemento necesario para la documentación de un sistema, permite la utilización de algunos diagramas como mecanismo para lograr un mayor entendimiento del negocio. En relación con este trabajo se utilizará a UML para modelar la relación entre cada unas de las clases que componen la aplicación, así como para representar la composición y funcionamiento de la arquitectura seleccionada.

2.7 Metodologías de desarrollo de software.

Dentro del proceso de desarrollo de software, la metodología de desarrollo de software se define como “un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software”(Méndez

2010). Por lo que la selección de una adecuada metodología, se puede considerar un paso trascendental para el desarrollo del proyecto. La utilización de una metodología que no se ajuste a las características del producto a desarrollar, puede traer como consecuencia, la posterior inconformidad del cliente o la entrega del producto fuera del cronograma, por tanto una incorrecta selección, puede incidir directamente en el fracaso del proyecto en general (Roberth G. Figueroa and Cabrera 2010). Cada metodología está compuesta por los siguientes elementos:

- ¿Cómo dividir un proyecto en etapas?
- ¿Qué tareas se llevan a cabo en cada etapa?
- ¿Qué restricciones deben aplicarse?
- ¿Qué técnicas y herramientas se emplean?
- ¿Cómo se controla y gestiona un proyecto?

La diversidad existente en la producción de software, provoca un crecimiento en la variedad de propuestas metodológicas relacionadas al mismo. Dentro de esta variedad se pueden mencionar dos clasificaciones. En primer lugar están las metodologías estructuradas, las cuales agrupan cualquier desarrollo orientado a procesos, orientado a datos o mixtos; y en segundo lugar están las no estructuradas, las que incluyen los productos orientados a objetos y sistemas en tiempo real. Existen además lo que se conoce como metodologías tradicionales, las cuales se aplican eficientemente a proyectos de gran envergadura y las metodologías ágiles, aplicadas generalmente a proyectos pequeños, de poco personal y tiempo de desarrollo (Sanchez 2004).

2.7.1 Metodologías Tradicionales.

Estas metodologías se dirigen esencialmente al control de proceso, ya que establecen rigurosamente las actividades, artefactos, herramientas y notaciones que se usarán. Estas propuestas han demostrado ser muy efectivas y necesarias, pero también presentan una serie de errores en cuanto a la aproximación al factor humano o al producto software. Las metodologías tradicionales generalmente son utilizadas para proyectos de gran envergadura, por lo que enfatizan en una exhaustiva documentación de todo el proyecto guiadas por un plan o cronograma. Todo esto es definido en la fase inicial del proyecto, por lo que no es muy conveniente emplearlas, si se considera introducir cambios inesperados en el plan.

2.7.2 Framework de Soluciones de Windows (MSF).

MSF más que una metodología de desarrollo de software es un compendio de las mejores prácticas en cuanto a administración de proyectos, asimismo presenta como

principal característica la posibilidad de adaptarse a cualquier proyecto de tecnología de la información. MSF se inclina hacia los modelos de procesos y de equipos por encima de las elecciones tecnológicas, además se identifica por ser muy adaptable, ya que su uso no es específico a un estándar determinado, además de ser escalable, pues puede estar constituido por un equipo de 3 o 4 personas y de permitir el trabajo en cualquier ambiente de desarrollo por lo que se denomina de gran flexibilidad (Sanchez 2004), (Roberth G. Figueroa and Cabrera 2010).

Faces de MSF:

- Visión y Alcances.
- Planificación.
- Desarrollo.
- Estabilización.
- Implantación.

Los equipos de trabajo utilizados por MSF, tienen como meta principal, compensar las desventajas jerárquicas que caracterizan los proyectos tradicionales. Estos resultan ser pequeños y se distinguen por balancear las destrezas de sus integrantes, además de compartir responsabilidades con el fin de mantenerse enfocados en los objetivos planteados. El modelo está compuesto por seis roles, donde cada rol puede ser desempeñado por más de una persona, pues no es un modelo jerárquico, por tanto cada rol tiene igual importancia en su aporte al proyecto.

2.7.3 Proceso Racional Unificado (RUP).

Esta se puede considerar como la metodología tradicional más popular, tiene como principal objetivo asegurar la producción de software de alta calidad, respetando un cronograma y un presupuesto, aunque está pensado además para adaptarse a cualquier tipo de proyecto. RUP se caracteriza por ser iterativo-incremental y se basa en casos de uso para la descripción de lo que se espera obtener del sistema, está orientado a la arquitectura del sistema y utiliza a Lenguaje Unificado de Modelado (UML, por sus siglas en ingles) como herramienta principal de modelado. Por lo general, el proceso de desarrollo es muy grande por lo que en muchas ocasiones hay que adaptarlo a las necesidades de la empresa, tratando de obtener versiones reducidas del mismo (Roberth G. Figueroa and Cabrera 2010), (Sanchez 2004), (Molpeceres 2002).

El proceso de desarrollo según RUP se divide en cuatro fases:

- Concepción (inicio del proyecto).
- Elaboración (definición, análisis, diseño).
- Construcción (implementación).
- Transición (fin del proyecto y puesta en producción).

Cada una de estas fases, son desarrolladas mediante un ciclo de iteraciones, en las cuales se reproduce el ciclo de vida en cascada, hay que mencionar además que el objetivo de cada una de estas iteraciones, depende directamente de las evaluaciones de las iteraciones precedentes. Un rasgo fundamental de esta metodología es que en cada ciclo se hace exigente el uso de artefactos y roles, por lo que se considera una de las más importantes, para obtener un grado de certificación en el desarrollo de software.

Ventajas de RUP.

- Evaluación en cada fase que permite cambios de objetivos.
- Funciona bien en proyectos de innovación.
- Es sencillo, ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el software.
- Seguimiento detallado en cada una de las fases.

Desventajas de RUP.

- La evaluación de riesgos es compleja.
- Excesiva flexibilidad para algunos proyectos.
- Coloca al cliente en una situación que puede ser muy incómoda para él.
- El cliente deberá ser capaz de describir y entender a un gran nivel de detalle, para poder acordar un alcance del proyecto con él.

2.7.4 Metodologías ágiles.

Con el paso de los años y auge de todo tipo de opiniones en relación con las metodologías tradicionales, surge un nuevo enfoque como respuesta a los errores que presentan, de modo que es así como surgen las metodologías ágiles. Esta nueva teoría se basa principalmente en la planificación adaptativa, por lo que ponen en práctica que la capacidad de dar respuesta a un cambio es más importante que el seguimiento estricto de un plan, esto es un principio fundamental ya que para muchos clientes el estar preparados para el cambio, se traduce en reducir el costo del proyecto.

El auge de las metodologías ágiles, ha revolucionado a la producción de software a nivel mundial, el tema ha generado una polémica entre los defensores de ambas metodologías, pero hay un aspecto que no se puede omitir y es que han de mostrado su efectividad para proyectos acelerados y de poco presupuesto. Con esta planificación a corto plazo se tiene una parte del software disponible para el cliente, independientemente de cualquier inconveniente que atrase o acelere el producto. Entre los principales métodos ágiles se tiene el XP (Extreme Programming), Scrum, Iconix, Cristal Methods, AUP entre otras.

Los principales principios de este tipo de metodologías son:

- Los individuos y las relaciones entre ellos, superan en importancia a las herramientas y procesos empleados.
- El crear un software que funcione es más importante que tener una documentación exhaustiva.
- La colaboración e intercambio con el cliente debe prevalecer por encima del contrato.
- La capacidad de dar respuesta a un cambio es más importante que el seguimiento de un plan.

El retrasar las decisiones cuanto sea posible, es un aspecto positivo tanto para el cliente como para el equipo, pues permite mantener la satisfacción por parte del cliente el mayor tiempo posible. Como primer elemento favorable de esta táctica, está el reducir el número de decisiones de alta inversión así como el número de cambios necesarios para el proyecto. De igual manera se pueden nombrar como otra ventaja y consecuencia de este aspecto el decremento del costo de cambios destinado para la confección del producto.

2.7.5 Scrum.

Esta es una metodología ágil destinada para controlar y administrar el proceso de desarrollo de software, se desarrolla de forma iterativo-incremental donde cada ciclo termina con una porción del producto que incorpora una nueva funcionalidad. Scrum se ha utilizado durante los últimos 10 años con gran éxito, ya que su principal objetivo está dirigido hacia proyectos con un rápido cambio de requisitos. Además se destaca por contener una serie de iteraciones denominadas sprints, con una duración de 30 días, donde cada iteración es un incremento del producto para mostrar al cliente. Igualmente se puede mencionar que entre sus principales rasgos están las reuniones

diarias de 15 minutos con del equipo de trabajo, con el fin de lograr una mayor integración y coordinación.

Scrum está especialmente diseña diseñada para admitir cualquier tipo de cambios en los requerimientos, especialmente en un mercado de alta competitividad, aparte de priorizar el trabajo en dependencia del valor que tenga para el negocio. Los requerimientos y las prioridades se revisas y ajustan en intervalos de tiempos muy cortos y regulares, por lo permite adaptar el software a las necesidades del cliente en tiempo real. Otra característica fundamental es que el equipo de desarrollo, se dirige hacia un único objetivo: construir un producto de calidad, por lo que la gestión de proyecto es el encargado de definir las características del producto, así como de eliminar cualquier obstáculo que impida el cumplimiento del objetivo final del equipo.

Esta metodología contiene un conjunto de reglas, basadas en principios de inspección continua, adaptación, auto-gestión e innovación, donde el cliente se compromete y entusiasma con el crecimiento del producto. Por otra parte los desarrolladores cuentan con las condiciones ideales para su superación profesional, lo que resulta un incentivo para el entusiasmo del equipo (Roberth G. Figueroa and Cabrera 2010).

2.7.6 Programación Extrema (XP).

Esta es una de las metodologías más exitosas dentro de los procesos ágiles, para proyectos a corto plazo y corto equipo. La programación extrema se diferencia de las metodologías tradicionales en que pone más énfasis en la adaptabilidad que en la previsibilidad. En XP, cualquier modificación en los requisitos sobre la marcha es un aspecto normal, pues tiene la capacidad de adaptarse a los cambios desde cualquier punto de vida del proyecto. La misma consiste en una programación rápida, donde el usuario final forma parte del equipo de desarrollo, lo cual es esencial para el éxito del proyecto. Los aspectos fundamentales de XP, esa la comunicación entre usuarios y desarrolladores, logrando una retroalimentación concreta, entre el equipo de desarrollo, clientes y los usuarios finales(Roberth G. Figueroa and Cabrera 2010), (Molpeceres 2002).

Las características fundamentales de XP son:

- **Desarrollo iterativo e incremental:** Pequeñas mejoras, unas tras otras.
- **Pruebas unitarias continuas:** Frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- **Programación por parejas:** Se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone que la mayor calidad del código escrito de esta manera el código es revisado y discutido

mientras se escribe- es más importante que la posible pérdida de productividad inmediata.

- **Frecuente interacción:** Del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- **Corrección:** De todos los errores antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
- **Refactorización:** Del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- **Propiedad del código compartida:** En vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.
- **Simplicidad en el código:** Es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que en más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo (Escribano 2002).

Ventajas de XP:

- Apropiado para entornos volátiles.
- Estar preparados para el cambio, significa reducir su coste.
- Planificación más transparente para nuestros clientes, conocen las fechas de entrega de funcionalidades ya que son vitales para su negocio.
- Permitirá definir en cada iteración cuales son los objetivos de la siguiente.
- Permite tener realimentación de los usuarios muy útil.
- La presión esta a lo largo de todo el proyecto y no en una entrega final.

Desventajas de XP:

- Delimitar el alcance del proyecto con nuestro cliente.

2.7.7 ¿Por qué utilizar XP?

Luego de un análisis de cada una de las metodologías mencionadas con anterioridad, se selecciona a la metodología XP como la más adecuada para encaminar el desarrollo de esta investigación. XP como toda metodología ágil, se ajusta a un proyecto con un pequeño equipo de trabajo, así como corto período de desarrollo, por lo que esta característica es un punto de contacto con el desarrollo de esta investigación.

Como otro principio para seleccionar esta metodología es que la misma asimila cambios en cual fase de vida del proyecto, por tanto esto resulta muy útil pues el desarrollo de esta investigación está centrado en la confección del software por encima de la documentación. También se puede mencionar que en esta investigación se asume como cliente al jefe del proyecto minería de la UCI, por lo que se cuenta permanentemente con el mismo para definir el alcance y complejidad de cada uno de los requisitos del sistema en cada iteración, lo cual es otra característica importante de XP.

2.8 Conclusiones Parciales.

En este capítulo se analizaron diversas tecnologías, con el objetivo de obtener la mejor selección que permita sostener el desarrollo de la herramienta propuesta. En primer lugar se determina utilizar como lenguaje de programación a C++, utilizando los frameworks Qt y OpenCascade para la confección de la interfaz de usuario. De igual se establece utilizar la metodología de desarrollo XP, por ser la que más se ajusta a las características propias de la investigación, además planificar un desarrollo regido por la filosofía del software libre.

Capítulo 3 "Análisis y Diseño"

3.1 Introducción.

En el presente capítulo se presentan las 2 primeras fases de la metodología XP (exploración, planificación) en las que se verá reflejado todo el análisis y diseño del proyecto. Mediante las historias de usuarios definidas por el cliente, se describen cada uno de los requisitos funcionales así como su prioridad respecto a la implementación. Luego de estimar los esfuerzos de cada funcionalidad, en la planificación de iteraciones se procede a seleccionar que funcionalidad se implementa en cada iteración, hasta que en el plan de entrega se estima el tiempo que va a tomar el desarrollo de las mismas. Finalmente se completa todo el diseño correspondiente a la propuesta en cuestión, detallando todos los aspectos relacionados con la arquitectura del sistema y el estándar de código definido.

3.2 Flujo Actual del Proceso.

Fases de XP.

El proceso de desarrollo en XP generalmente cuenta con 4 fases y tiene un éxito total cuando el cliente selecciona correctamente el valor del negocio basándose en la capacidad del equipo para entregar el producto a tiempo. En primer lugar están las fases de Exploración, Planificación de la Entrega, en las cuales se describirán cada una de las funcionalidades mediante las historias de usuarios confeccionadas por el cliente, además de definir la prioridad y la estimación del esfuerzo necesario para cada una. Se realizará además un plan de entrega y una planificación de las historias a realizar en cada iteración.

Una vez finalizadas estas etapas, le siguen las fases de Implementación y Pruebas, en las que se aplicaran las pruebas adicionales además de tomar las decisiones necesarias en relación a los cambios introducidos en la versión actual. Además el sistema se mantiene funcionando, mientras que se producen nuevas versiones, hasta se genera la documentación final pues el cliente no presenta más historias de usuarios para incluir en el sistema (wesley 1999).

Roles de XP:

- **Programador:** Es el encargado de confeccionar el código del sistema y encargarse de las pruebas unitarias correspondientes.
- **Cliente:** Describe cada una de las historias de usuarios y de asignar su prioridad decidiendo cual implementar en cada iteración en correspondencia con su valor.

- **Encargado de pruebas (Tester):** Ejecuta las pruebas regularmente, ayudando al cliente. Es el encargado de las herramientas de soporte para las pruebas así como de difundir los resultados en el equipo.
- **Encargado de seguimiento (Trackter):** Verifica el grado de acierto entre las estimaciones y el tiempo real dedicado.
- **Entrenador (Coach):** Provee las guías y es el encargado de que las prácticas se apliquen correctamente.
- **Gestor (Big Boss):** Es el vínculo entre clientes y programadores y su labor esencial es la coordinación.

3.3 Exploración.

Esta fase se caracteriza por una completa retroalimentación con el cliente, pues es el encargado de definir la prioridad de cada uno de los requisitos funcionales del sistema, mediante las historias de usuarios. El tiempo empleado para la culminación de este período depende directamente de la familiaridad que tenga el equipo de desarrollo con las tecnologías y herramientas a utilizar.

3.3.1 Requisitos no funcionales.

Determinar los requisitos no funcionales de un sistema implica realizar la correcta identificación de las características generales que la aplicación debe tener, con el fin de garantizar la usabilidad y aceptación por parte del usuario. Cada una de estas propiedades o cualidades pueden abarcar diferentes aspectos relacionados con el hardware y software necesarios, o con el diseño, soporte y usabilidad. Siempre con la intención de asegurar la calidad del producto final. En relación con este trabajo de igual manera se determinan los requisitos no funcionales indispensables para la implementación del componente visual propuesto.

RNF1 Reusabilidad.

El sistema se desarrollará de forma que genere un componente visual para Qt, con el objetivo de lograr la reutilización en la futura implementación de cualquier aplicación CAD que necesite de las propiedades que el mismo posee.

RNF2 Soporte.

El componente debe de generar la documentación necesaria que facilite la comprensión de los desarrolladores a la hora de interactuar con el. Además debe de proporcionar el código fuente con el fin de mejorar la configuración, adaptación y funcionamiento del mismo.

RNF3 Hardware.

Requisitos mínimos: Procesador Pentium IV a 3.0 GHz, 1Gb RAM, disco duro: 80 Gb.
Tarjeta gráfica: NVidia GeForce de 1Gb de memoria ó superior.

RNF4 Software.

Sistema operativo: Ubuntu 10.10.
Framework de visualización: OpenCascade 6.3.
Framework para el desarrollo de aplicaciones: Qt 4.7.
IDE de desarrollo: QtCreator.

3.3.2 Historia de usuarios.

Es una técnica utilizada para describir cada uno de los requisitos funcionales y no funcionales, en ella el cliente especifica las características de la aplicación, es decir es una representación de las necesidades del sistema desde el punto de vista del cliente. Las historias de usuarios es un recurso muy flexible pues cada una se caracteriza por ser muy comprensible y delimitada, de modo que el implementador pueda desarrollarla en pocas semanas. A modo de planificación, cada historia de usuario puede ser de una a tres semanas, a al mismo tiempo pueden ser descompuestas en varias tareas de programación, para posteriormente ser asignadas a los programadores en cada iteración. En esta metodología no hay que preocuparse si no se identifican todas las historias de usuarios necesarias, pues en cada iteración, se registraran los cambios y según estos cambios se planifica la siguiente iteración. Es necesario mencionar que las historias de usuarios serán usadas posteriormente tanto en la implementación como en las pruebas del producto, para confirmar si el programa cumple con lo descrito en cada una de ellas (Escribano 2002).

A continuación se describen cada una de las historias de usuarios:

Tabla 1: HU Integración con el framework OpenCascade.

Historia de Usuario.		
Número: 1.	**Nombre:** Integración con el framework OpenCascade.	
Usuario: Sistema.		**Tiempo estimado:** 2 semanas.
Riesgo en desarrollo: Alto.	**Prioridad en negocio:** Alta.	
Descripción: EL sistema compila e incluye cada una de las clases que integran el		

framework OpenCascade.

Tabla 2: HU Generar Interfaz de Visualización.

Historia de Usuario.	
Número: 2.	**Nombre:** Generar Interfaz de Visualización.
Usuario: Sistema.	**Tiempo estimado:** 3 semanas.
Riesgo en desarrollo: Alto.	**Prioridad en negocio:** Alta.
Descripción: EL sistema utiliza las clases de OpenCascade y Qt, para generar una interfaz donde se representaran los objetos en 3D.	

Tabla 3: Generar el componente que se integre a Qt.

Historia de Usuario.	
Número: 3.	**Nombre:** Generar el componente que se integre a Qt.
Usuario: Sistema.	**Tiempo estimado:** 2 semanas.
Riesgo en desarrollo: Alto.	**Prioridad en negocio:** Alta.
Descripción: EL sistema compila todo el proyecto como un componente de visualización que se pueda integrar a QtCreator.	

3.4 Planificación.

En esta fase el cliente establece la prioridad de cada historia de usuario y se realiza realizan una estimación del esfuerzo necesario para su futuro desarrollo. Se diseña un cronograma en conjunto con el cliente, programando cada una de las entregas. Los desarrolladores establecen la estimación para la implementación de las historias de usuarios basándose en el tiempo o el alcance.

3.4.1 Estimación del esfuerzo.

En este aspecto de la planificación el equipo de desarrollo realiza una estimación del esfuerzo de cada una de las historias de usuarios, de modo que se pueda valorar las prioridades conjuntamente con el cliente. Es importante tener en cuenta que para esta fase se hace necesario realizar un análisis profundo de los requerimientos con el fin de determinar correctamente el tiempo de desarrollo.

Tabla 4: Estimación del esfuerzo.

NO	Historia de Usuario	Puntos de estimación (semanas).
1	Integración con el framework OpenCascade.	2
2	Generar Interfaz de Visualización.	3
3	Generar el componente que se integre a Qt.	2

3.4.2 Plan de Iteraciones.

Luego de diseñar cada una de las historias de usuarios se realizarán varias iteraciones por no más de 3 semanas cada una, antes de entregarle el producto. El cliente decide qué Historia de Usuario se implementarán en cada iteración, de modo que en la última iteración el sistema estará listo para entrar en la fase de producción.

A continuación se describen las 3 iteraciones correspondientes a la aplicación:

Iteración 1.

En esta primera iteración se procederá a implementar la historia de usuario 1, pues se considera de vital importancia para el desarrollo del componente. En esta funcionalidad se integrarán los frameworks Qt y OpenCascade, mediante la

compilación de cada una de las librerías de OpenCascade en un proyecto creado en Qt.

Iteración 2.

En una segunda iteración se implementara la segunda historia de usuario pues en ella se define la interfaz gráfica del componente. Para el desarrollo de esta funcionalidad se hace necesario el estudio de la vinculación entre Qt, OpenCascade y OpenGL a la hora de generar un panel de representación 3D.

Iteración 3.

En esta iteración se desarrolla la última historia de usuario, la cual generará el componente visual que obtendrá el cliente. El componente podrá incluirse con Qt y a la vez servirá de soporte para generar cualquier tipo de aplicación CAD.

3.4.3 Plan de duración de iteraciones.

Como parte de la técnica establecida por la metodología de desarrollo XP se crea un plan de duración de las iteraciones que se llevaran cabo durante el proceso de desarrollo. En dicho plan se estará mostrando la duración y de cada iteración en el orden en que serán implementadas.

Tabla 5: Plan de duración de iteraciones.

Iteraciones	HU en el Orden Implementar.	Duración.
Iteración 1	Integración con el framework OpenCascade.	2 Semanas.
Iteración 2	Generar Interfaz de Visualización.	3 Semanas.
Iteración 3	Generar el componente que se integre a Qt.	2 Semanas.

3.4.4 Plan de entrega.

El Plan de Entrega establece que las historias de usuarios serán agrupadas para conformar una entrega, donde el cliente ordenará según sus prioridades.

Tabla 6: Plan de entrega.

Entregas	HU

Entrega 1	Integración con el framework OpenCascade.
Entrega 2	Generar Interfaz de Visualización.
Entrega 3	Generar el componente que se integre a Qt.

3.5 Diseño de la propuesta.

La metodología seleccionada para el desarrollo de este trabajo se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, por lo que el diseño del mismo no requiere de la utilización de extensos diagramas de modelado. En su lugar se establecen las tarjetas CRC (Contenido, Responsabilidad y Colaboración), las cuales son las en cargadas de establecer la relación entre los objetos, usuarios y cada una de las cases que componen el sistema, aunque se puede considerar el uso de algún diagrama siempre que contribuya a la integración del equipo de desarrollo.

3.5.1Tarjetas CRC.

El principal objetivo que presentan las tarjetas CRC, es permitir que todo el equipo participe en el diseño de la aplicación, además de que permite el trabajo con una metodología basada en objetos. Las CRC están constituidas por 3 secciones en las que se mostraran el nombre de la clase, las responsabilidades de la clase las que constituyen las principales funcionalidades que debe cumplir. En una tercera sección se representan sus colaboraciones, las que no son más que las otras clases con las que se relaciona para cumplir sus responsabilidades.

A continuación se muestran cada una de las tarjetas correspondientes a este trabajo:

Tabla 7: Tarjeta CRC Clase MyWidgetPlugin.

Tarjeta CRC
Clase: MyWidgetPlugin
Súper Clase: QObject
Sub Clase(s)

Responsabilidades:	**Colaboraciones:**
isInitialized() includeFile() const; createWidget() initialize()	MyWidget

Tabla 8: Tarjeta CRC Clase MyVisorContex.

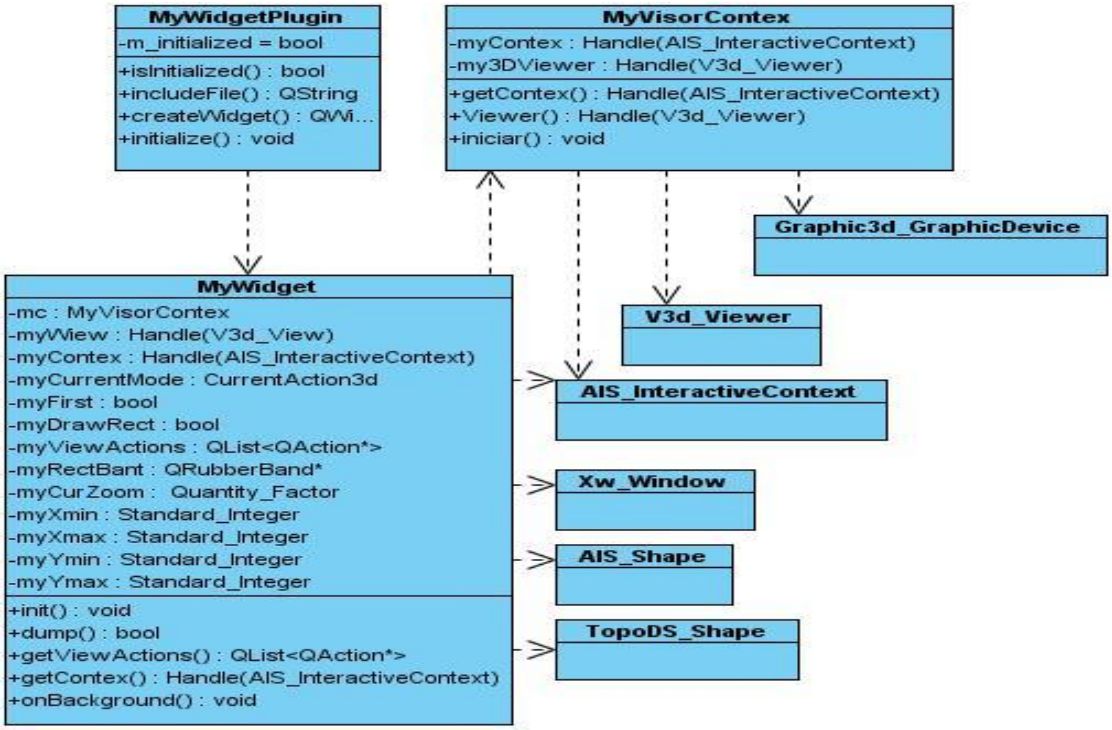
Tarjeta CRC	
Clase: MyVisorContex	
Súper Clase:	
Sub Clase(s)	
Responsabilidades: getContext() iniciar() Viewer()	**Colaboraciones:** AIS_InteractiveContext Graphic3d_GraphicDevice V3d_Viewer

Tabla 9: Tarjeta CRC Clase MyWidget.

Tarjeta CRC
Clase: MyWidget
Súper Clase: QWidget
Sub Clase(s)

Responsabilidades:	**Colaboraciones:**
init()	MyVisorContext
dump()	Xw_Window
getViewActions()	TopoDS_Shape
getContex()	AIS_Shape
onBackground()	AIS_InteractiveContext

Ilustración 1 Diagrama del diseño descrito por las tarjetas CRC.



3.5.2 Arquitectura del Sistema

“La arquitectura de software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos, el ambiente, los principios que orientan su diseño y evolución” (Reynoso 2004).

En pocas palabras la arquitectura de software es una vista del sistema que se encarga de describir aspectos globales del sistema, además de incluir y reflejar la forma en que los componentes del mismo interactúan y colaboran entre sí. La arquitectura de software forma parte del diseño del sistema por lo que está afectada no solo por la estructura y el comportamiento de la aplicación sino que también depende de la funcionalidad el rendimiento y la flexibilidad de comprensión, entre otros aspectos.

Arquitectura Basada en Componentes

Una arquitectura basada en componentes describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. 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.

El estilo de arquitectura basado en componentes tiene las siguientes características:

- Es un estilo de diseño para aplicaciones compuestas de componentes individuales.
- Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas.
- Define una aproximación de diseño que usa componentes discretos, los que se comunican a través de interfaces que contienen métodos, eventos y propiedades.

La estructura de la arquitectura basada en componentes contempla 3 partes:

- El nombre de los componentes: el nombre de un componente debe ser la identificación de la funcionalidad y uso que tiene como software. Generalmente, los desarrolladores usan algún tipo de convención que facilite la identificación de componentes, especialmente, cuando se trabaja en proyectos de gran envergadura.
- La interfaz de los componentes: es el área de intercambio (input-output) entre el interior y el exterior de un componente de software permitiendo el acceso a los datos y funcionalidades que estén especificadas en el interior del componente (acceder funcionalmente, no a su especificación). Adicional a la interface se encuentra la documentación que muestra la información sobre cómo utilizar un componente.
- Cuerpo y código de implementación: es la parte del componente que provee la forma (implementación) sobre la cual un fragmento del componente realiza sus servicios y funcionalidades. Este es el área que debe cumplir con el principio de encapsulación.

Estructura de la arquitectura aplicada.

El framework de desarrollo Qt esta constituido por diferentes patrones arquitectónicos, en dependencia del tipo de información que manipule. En relación al manejo interno de la información y con algunos aspectos relacionados a la interfaz de usuario se utilizan los patrones por capas y modelo vista controlador respectivamente, pero todo el negocio vinculado con cada uno de los módulos, componentes o plugins del framework son controlados por el patrón arquitectónico basado en componentes. Por esta razón se selecciona un diseño basado en componentes como guía para esta investigación, ya que lo que se espera generar con este trabajo es un componente o puglin que se integre con Qt, aprovechando esta parte de su arquitectura que facilita la incorporación de módulos sin necesidad de tener acceso al código del framework.

La organización de Qt al incorporar un componente varia en dependencia del nivel de acceso al framework, de la propia composición del componente. El objetivo perseguido por este trabajo es el obtener un componente de visualización en tres dimensiones, el cual se integraría al QtDesigner ya que es el encargado de manejar los componentes relacionados con la interfaz de usuario. De igual manera la organización particular del componente propuesto respeta las pautas establecidas por el patrón arquitectónico correspondiente, pues cuenta con clases que contienen las responsabilidades requeridas (en este caso MyVisorContex y MyWidget), además de una clase interfaz (en este caso MyWidgetPlugin) que se encarga de exportar el plugin o componente, estableciendo cada una de las propiedades especificas como son el nombre, el ícono, la dirección del proyecto entre otros.

Ilustración 2 Estructura del Componente Propuesto.

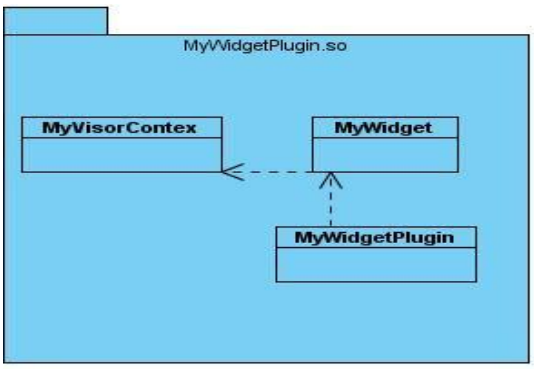
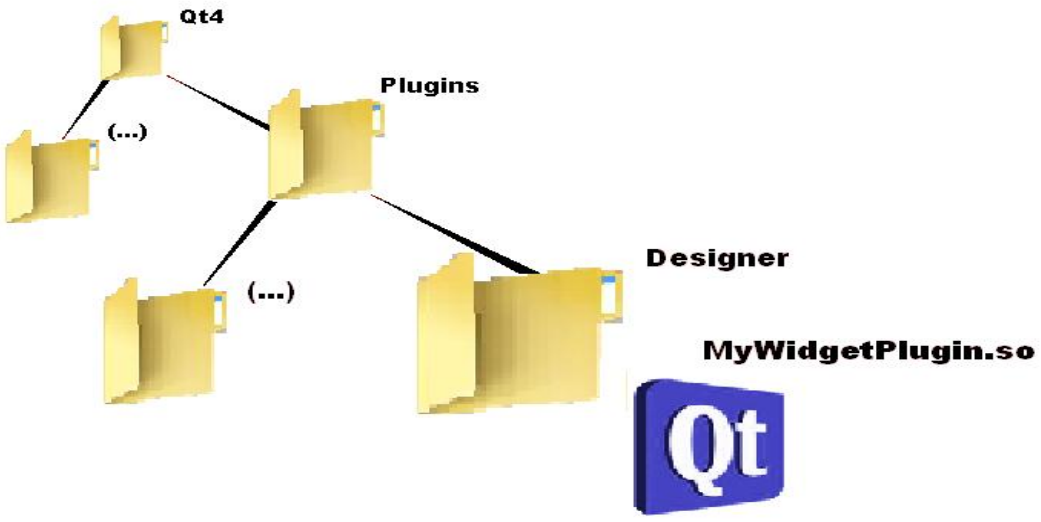


Ilustración 3Estructura de la integración del componente propuesto.



3.5.3 Patrones de diseño.

Los patrones de diseño nacen producto a la experiencia adquirida durante el desarrollo de sistemas informáticos, por lo que constituyen una respuesta a las problemáticas de comunes encontradas en la implementación de un software. Cada uno de estos patrones de alguna manera ha contribuido a evolucionar el diseño orientado a objetos por lo que el utilizarlos constituye una buena práctica de programación (Gracia 2005).

Objetivos de los patrones de diseño

- Facilitar el aprendizaje de las nuevas generaciones de diseñadores a partir de un conocimiento ya existente.
- Generar estándar a la hora de crear un diseño de software.
- Proporcionar un elemento reusable en el diseño de sistemas de software.

Patrón Bajo Acoplamiento.

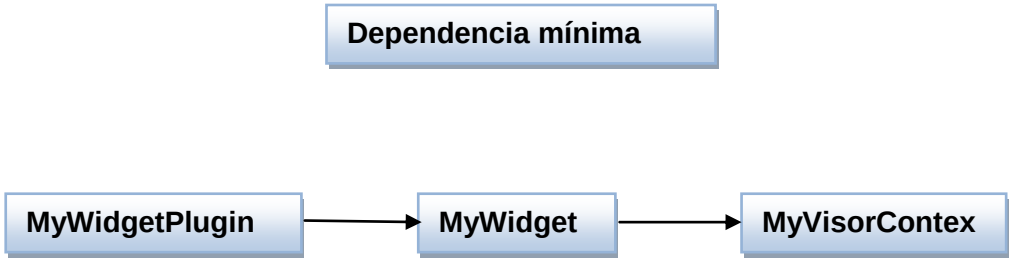
El patrón Bajo Acoplamiento se plantea el dar soporte a una escasa dependencia asegurando una alta reutilización del código, “por lo que el acoplamiento es una medida de la fuerza con que una clase esta conectada a otras clases” (Visconti and Astudillo 2011). Un sistema sustentado por un alto acoplamiento en general constituye un síntoma de un mal diseño pues puede presenta problemas al reutilizar el código además de que un cambio en alguna de las clases supone generar un cambio global que dificulta el entendimiento de las mismas cuando están aisladas. Por tanto resulta una mejor alternativa el asignar las responsabilidades tratando de mantener un grado de acoplamiento bajo (Visconti and Astudillo 2011).

Beneficios del Bajo Acoplamiento:

- Las clases no se afectan por cambios en otros componentes.
- Al ganar en independencia, facilita el entendimiento del sistema.
- Facilita la reutilización.

La herramienta que da origen a esta investigación establece a este patrón de diseño, como una de las guías fundamentales para estructurar su sistema, pues establece una dependencia mínima para cada una de las clases que lo conforman.

Ilustración 4 Bajo Acoplamiento.



Patrón Alta Cohesión.

Este patrón de diseño resuelve el problema de mantener la complejidad dentro de los límites manejables, asumiendo como principal objetivo el que cada elemento debe de realizar una labor única dentro del sistema. El establecer una alta cohesión asegura que una clase no se sature con tareas, por lo que reparte cada una de las responsabilidades especificas en diferentes entidades. Por tanto se puede comprender como cohesión a la medida de la fuerza con que se relacionan los objetos o de la cantidad de trabajo que realizan (Visconti and Astudillo 2011).

Beneficios de una alta cohesión.

- Facilita la claridad con que se comprende el diseño.
- Simplifica en gran medida el mantenimiento y las mejoras de funcionalidad.
- Soporta mayor capacidad de reutilización.

Dentro del proceso de desarrollo de este trabajo se hace imprescindible establecer una alta cohesión, asegurando que cada clase tenga una responsabilidad específica, contribuyendo a la comprensión y mantenimiento del sistema propuesto.

Ilustración 5 Alta Cohesión.

(MyWidgetPlugin)	(MyWidget)	(MyVisorContex)
Se encarga de exportar el componente.	Se encarga de generar la interfaz de visualización.	Se encarga de inicializar el visor y el contexto

Patrón Creador.

El problema que trata este patrón es el de establecer quien es el responsable de crear una nueva instancia de una clase, de modo que este patrón es asigna responsabilidades relacionadas con la creación de objetos, definiendo como objeto creador aquel que necesite conectarse al objeto creado (Visconti and Astudillo 2011).

Este patrón asigna a la clase B la responsabilidad de crear una instancia de la clase A (B es creador de los objetos A) si:

- B contiene objetos de A.
- B agrega objetos de A.
- B registra instancias de objetos de A.
- B tiene los datos de inicialización de A (datos que requiere su constructor).
- B utiliza más estrechamente datos de A (Visconti and Astudillo 2011).

Beneficios del patrón Creador.

- Proporciona un diseño que puede soportar un bajo acoplamiento y una mayor claridad
- Facilita el encapsulamiento y la reutilización del código.

En la implementación que ocupa a esta investigación, este patrón es evidenciado a través de la clase MyWidget, ya que esta es la encargada de instanciar un objeto de tipo MyVisorContext, pues contiene a este objeto dentro de sus parámetros y necesita de sus datos para realizar las funcionalidades establecidas en ella (Visconti and Astudillo 2011).

Ilustración 6 Creador.

(MyWidget)	**(MyWidget)**
Es el responsable de instanciar el objeto Contexto pues requiere de su información.	**Es el responsable de instanciar el objeto Widget, para exportar el componente.**

Patrón Experto.

Este patrón se especializa en la asignación de responsabilidades siguiendo el principio fundamental del diseño orientado a objetos. Según este patrón se le asigna la responsabilidad de realizar una tarea determinada al objeto que cuenta con toda la información necesaria para ello, buscando como objetivo el generar un sistema de fácil mantenimiento, entendimiento y ampliación. Hay que tener en cuenta que el Patrón Experto es oportuno siempre y cuando se mantengan controlados los parámetros de cohesión y acoplamiento.

Beneficios del Patrón Experto.

- Se encarga de mantener el encapsulamiento, ya que los objetos utilizan su propia información para realizar sus responsabilidades.
- Contribuye a estructurar sistemas más robustos y de fácil mantenimiento.
- Brinda un soporte de alta cohesión al repartir las responsabilidades entre las clases que contengan la información necesaria (Visconti and Astudillo 2011).

La utilización del Patrón Experto en este trabajo contribuye a no dispersar la información y las responsabilidades en demasiadas clases, pues se identifica a un objeto experto que cuente con la información necesaria para realizar terminada tarea.

3.5.4 Estándar de Codificación.

Los estándares de codificación proveen un estilo de programación uniforme, posibilitando que haya un entendimiento común en el equipo de desarrollo, pues cada uno de sus integrantes estará escribiendo un mismo lenguaje. Además constituyen pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. También son conocidos por estilos de programación o convenciones de código; no son más que convenios para escribir código fuente en ciertos lenguajes de programación, manteniendo un estilo lo cual permite mejorar el proceso de codificación, haciéndolo mucho más eficiente.

A continuación se muestra el estándar de codificación escogido para este trabajo.

Declaraciones.

Número de declaraciones por línea: Por definición, nunca se utilizará una misma línea para más de una definición, la cual cosa facilita los comentarios relativos al elemento declarado.

Inicialización: Siempre que sea posible las variables se inicializarán en la misma línea de declaración.

Declaración de clases e interfaces: Para definir las clases e interfaces, se seguirán las siguientes reglas:

- No se usará ningún espacio entre el nombre de un método y el paréntesis de abertura de la lista de parámetros.
- La llave de abertura que contiene el código se escribe sola en la línea siguiente a la definición del prototipo. Igualmente, la llave de cierre correspondiente se escribe sola en la última línea.

Sentencias.

- Sentencias simples: Cada línea contendrá no más de una sentencia, aunque, una sentencia puede estar en más de una línea.

- Sentencias de retorno: la sentencia return no utiliza paréntesis.

```
return myViewer;
```

- Sentencia de selección básica (**if/if...else/if...else if...else**): Las llaves de inicio de un bloque de código se pone al final de la sentencia **if**, **else** o **else if**.

```
if (!fileName.isEmpty())
        {
          …..
        }
```

- Sentencias de bucle for o foreach: De forma similar a las sentencias de selección, la llave de abertura de bloque se colocará tras la sentencia de definición del **for** o **foreach** y la de cierre en una línea independiente.

```
for(int i,i<0,i++)
{
   ...
}
```

- Sentencias de bucle **while** o **dowhile**: de modo idéntico al **for**, pero en el caso del **dowhile**, el **while** se colocará en la misma línea que la última llave.

```
while(ileName.isEmpty())
{
   ...
}
```

Espacios en blanco.

- Líneas en blanco: Se pueden utilizar líneas en blanco para separar grupos de líneas que tengan cierta relación lógica. Dos líneas en blanco seguidas se usan para: separar secciones de código en un fichero o separar definiciones de clases e interfaces dentro de un fichero.

- Separaciones entre términos: En una lista de parámetros, se pondrán espacios tras las comas, pero nunca tras o antes de los paréntesis. En las asignaciones, se pondrán espacios antes y después del. En las expresiones, se utilizarán espacios sólo cuando sean estrictamente necesarios y para separar las distintas expresiones.

- Separaciones en las declaraciones: Se procurará mantener una estructura tabulada para las líneas de declaraciones de variables, de manera que leer la información sea fácil. En el ejemplo siguiente se puede observar cómo mantener dicha estructura, pero hay que tener en cuenta que para el espaciado, hay que utilizar espacios, no tabuladores:

Nomenclatura.

- Nomenclatura para clases: Nombres deben de escribirse con palabras compuestas, sin espacios, donde cada palabra comience en mayúscula.
  class MyWidget : public QWidget
- Nomenclatura para enumeraciones: Se nombrarán con sustantivos que describan su comportamiento.
  enum Formato{FormatBREP, FormatIGES, FormatSTEP, FormatCSFDB, FormatVRML, FormatSTL}
- Nomenclatura para variables: Se utilizarán sustantivos que describan su comportamiento, con la primera letra en minúscula.
  Handle_AIS_InteractiveContext myContext;
- Nomenclatura para métodos: Se utilizarán sustantivos que describan su comportamiento, comenzando con la primera letra en minúscula.

```
Handle_V3d_Viewer& myVisorContex::getViewer()
{
    return myViewer;
}
```

Práctica de programación.

- Visibilidad: Se codificaran los atributos de clase y de instancia como privados, garantizando un método de acceso para cada uno.
- Magia prohibida: No se utilizarán números que puedan ser sustituidos por constantes, garantizando que en caso de modificación del mismo, sólo se modifica en un lugar.

Comentarios.

Para los comentarios se utilizará el formato establecido por Javadoc con el objetivo de poder generar la ayuda correspondiente al proyecto a través de la herramienta Doxygen.

Nomenclatura:

- Cada bloque de comentarios se abrirá con /** y se cerrará con */.
- Cada línea dentro de un bloque de comentarios se iniciará con *.
- Cada descripción contenida en un bloque de comentarios debe de terminar en punto.

Ejemplo:

```
/**
 * Enumerativo de estado.
 *
 * Este enumerativo se encarga de establecer el estado que.
* tiene el contexto en un momento determinado.
 */
```

3.6 Conclusiones Parciales:

En este capítulo se analizó en detalle la propuesta de solución que se desea implementar, además de exponer el análisis y diseño de la investigación, incluido en las dos primeras fases de la metodología de desarrollo XP. Primeramente en la fase de exploración se diseñaron cada una de las historias de usuarios, logrando estimar el esfuerzo necesario para la implementación de las mismas. Del mismo modo durante la planificación del sistema, se documentaron cada una de las 3 iteraciones, con el fin estimar su prioridad y de pronosticar un plan de entrega coherente con la propuesta presentada. Mediante las tarjetas CRC se conformó el diseño de la aplicación, proponiendo un trabajo basando en una arquitectura por componentes y delimitado el estándar de código óptimo para la futura implementación del sistema.

Capitulo 4 "Implementación y Prueba"

4.1 Introducción.

En el presente capítulo se exponen cada uno los componentes que integran el proceso de implementación y prueba de la aplicación en cuestión. Según las pautas que presenta la metodología de desarrollo XP, el proceso de implementación se caracteriza por ser iterativo, de modo que se describen cada una de las iteraciones comprobando que se cumplen cada uno de los objetivos planteados en el plan de iteraciones. Finalmente se procede a realizar conjuntamente con el cliente y el equipo de desarrollo, las pruebas correspondientes a cada historia de usuario, verificando que el sistema cumple todas las funcionalidades.

4.2 Implementación.

Desarrollo de iteraciones:

Durante la fase de Planificación se detallaron las historias de usuarios correspondientes a cada una de las iteraciones a desarrollar, las cuales tienen como objetivo principal atender las necesidades requeridas por el cliente. En el transcurso de las iteraciones se lleva a cabo una revisión del plan de iteraciones, modificándose en caso de ser preciso. Como parte de este plan, se descomponen las historias de usuarios en tareas de programación o ingeniería, asignando a un equipo de desarrollo (o una persona), responsable de su implementación, aplicando la práctica de la programación en parejas. Estas tareas no tienen que necesariamente ser entendidas por el cliente, pueden ser escritas en lenguaje técnico y son para el uso estricto de los programadores.

Teniendo en cuenta la planificación realizada con anterioridad, se llevó a cabo el desarrollo del sistema en tres iteraciones, obteniéndose como finalidad un producto con todas las restricciones y características deseadas por el cliente. A continuación se detallan cada una de las iteraciones.

4.2.1 Iteración 1.

Tabla 10: Iteración 1.

Historia de Usuario	Tiempo de implementación	
	Estimación	Real
Integración con el framework	2	2

OpenCascade.		

Tabla 11: Iteración 1.

Tarea de ingeniería.	
No de tarea:1	**No de HI:**1
Nombre de la tarea: Integración con el framework OpenCascade	
Tipo de tarea: Desarrollo	**Puntos estimado:**2
Fecha Inicio: 21/03/11	**Fecha Fin:**03/04/11
Programador responsable: Miguel Arcia Salazar	
Descripción.	

4.2.2 Iteración 2.

Tabla 12: Iteración 2.

Historia de Usuario	**Tiempo de implementación**	
	Estimación	**Real**
Generar interfaz de visualización.	3	3

Tabla 13: Iteración 2.

Tarea de ingeniería.	
No de tarea:1	**No de HI:**2
Nombre de la tarea: Generar Interfaz de Visualización	
Tipo de tarea: Desarrollo	**Puntos estimado:**3
Fecha Inicio:04/04/11	**Fecha Fin:**24/04/11
Programador responsable: Miguel Arcia Salazar	

Descripción.

4.2.3 Iteración 3.

Tabla 14: Iteración 3.

Historia de Usuario	Tiempo de implementación	
	Estimación	Real
Generar componente visual que se integre a Qt.	2	2

Tabla 15: Iteración 3.

Tarea de ingeniería.	
No de tarea:1	**No de HI:**3
Nombre de la tarea: Generar componente visual que se integre a Qt	
Tipo de tarea: Desarrollo	**Puntos estimado:**2
Fecha Inicio: 25/04/11	**Fecha Fin:**11/05/11
Programador responsable: Miguel Arcia Salazar	
Descripción.	

4.3 Pruebas.

La fase de pruebas es un aspecto de vital importancia dentro del ciclo de vida de un proyecto que encaminado por la metodología XP, puesto que permiten certificar con exactitud la calidad de un producto. Estas pruebas disminuyen considerablemente el número de errores, reduciendo el tiempo de transcurrido entre la aparición y detección de un error, fortaleciendo del mismo modo la seguridad del producto en cuestión.

En XP el proceso de pruebas se divide en dos ramas, las pruebas de unitarias y las pruebas de aceptación. Las pruebas unitarias son protagonizadas por los implementadores, pues son los encargados de verificar directamente el código del producto. Por otro lado las pruebas de aceptación son ejecutadas por un equipo más numeroso, ya que el cliente es el encargado de diseñarlas; por tanto inciden

directamente en la satisfacción del cliente con el software al final de cada iteración y al principio de la siguiente.

4.3.1 Pruebas Unitarias.

Un proceso de desarrollo dirigido por pruebas se separa en dos actividades: En primer lugar se escriben las pruebas (Test First Development) seguido por la refactorización (Refactoring) del código. Para escribir las pruebas se utilizan las pruebas unitarias, en ellas se procede a escribir las pruebas y se verifica que estas fallen, luego se procede a corregir y refactorizar el código escrito. El objetivo de este diseño manejado por pruebas (Test Driven Design) es lograr un código limpio donde los requerimientos se traduzcan a pruebas, de modo que al pasar las pruebas se certifica el funcionamiento de los mismos (Escribano 2002).

Ciclo del TDD.

- Escribir la prueba: El implementador debe de comprender los requerimientos de los requisitos, comprobando cada uno de los escenarios de pruebas.
- Corregir el código de forma que pase la prueba: A la hora de corregir el código, el desarrollador debe de asumir la perspectiva del cliente desde la interfaz del producto.
- Pruebas automatizadas: El desarrollador debe de garantizar que con el código resuelve cada uno de los casos de pruebas escritos.
- Refactorización y limpieza del código: Se efectúan nuevamente los casos de prueba y se observan los resultados.
- Repetición: Se repite el ciclo y se procede a añadir las funcionalidades adicionales.

4.3.2 Pruebas de Aceptación.

Las pruebas de aceptación son diseñadas a partir de cada iteración y basadas en las Historia de Usurarios, en ella es el cliente el encargado de comprobar que ha sido correctamente implementada. Estas pruebas son reconocidas por su similitud con las llamadas “Pruebas de caja negra”, pues como los clientes son los principales responsables de la inspección, también son los autorizados de diseñar la prioridad o el orden de resolución en caso de que falle alguna funcionalidad (Escribano 2002).

Tabla 16: caso de prueba HU_1.

Caso de Prueba de Aceptación	
Código:UH1_P1	**HU:** 1
Nombre: Integración con el framework OpenCascade	
Descripción: Comprobar que se generó un código que integre ambas tecnologías.	
Condiciones de Ejecución: El cliente debe de comprobar que se integraron ambas tecnologías.	
Entrada / Pasos de Ejecución: Crear un código utilizando ambos frameworks.	
Resultado Esperado: Ambos frameworks se encuentran integrados.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 17: caso de prueba HU_2.

Caso de Prueba de Aceptación	
Código:UH2_P1	**HU:** 2
Nombre: Generar Interfaz de Visualización.	
Descripción: Comprobar que se generó una interfaz de visualización.	
Condiciones de Ejecución: El cliente debe de comprobar que se puede visualizar en la interfaz de visualización.	
Entrada / Pasos de Ejecución: Dibujar un objeto tridimensional en la interfaz.	
Resultado Esperado: Se logró visualizar un objeto tridimensional en la interfaz.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 18: caso de prueba HU_3.

Caso de Prueba de Aceptación	
Código: UH3_P1	**HU:**3
Nombre: Generar el componente que se integre a Qt.	

Descripción: Comprobar que se generó un componente visual para Qt.
Condiciones de Ejecución: El cliente debe de comprobar que el componente visual se integre a Qt.
Entrada / Pasos de Ejecución: Crear un nuevo proyecto a partir del componente visual.
Resultado Esperado: Se logró crear un componente visual que se integre a Qt.
Evaluación de la Prueba: Satisfactoria.

Hay que mencionar que una historia de usuario no se puede considerar terminada, hasta que no pase con éxito las pruebas de aceptación. Del mismo modo como es una inspección colectiva, se hace necesario que todo el equipo de desarrollo esté al tanto de los resultados de las mismas.

4.4 Conclusiones Parciales.

En este capítulo se representó todo el proceso relacionado con la implementación del software, describiendo cada una de las iteraciones planificadas. Se detallaron cada una de las tareas de ingeniería e implementación correspondientes a cada iteración, corroborando que las mismas se desplegaron según lo previsto por el plan correspondiente. Se ejecutaron además cada una de las pruebas propuestas por la metodología XP, con el fin de validar la calidad de la aplicación. En primer lugar se aplicaron las pruebas unitarias seguidas por las pruebas de aceptación donde el cliente comprobó las funcionalidades descritas mediante las Historias de Usuarios. Al finalizar este capítulo se concluye todo el proceso de desarrollo que ocupa a esta investigación.

Conclusiones Generales

Una vez finalizado todo el proceso de desarrollo generado por esta investigación, se hace necesario mencionar las siguientes conclusiones:

- Se consideran cumplidos los objetivos trazados al inicio de la investigación, mediante la realización de los requisitos funcionales y las pruebas establecidas por el proceso ingenieril.
- A partir de la metodología de software seleccionada se elaboró la documentación que recoge todo el proceso de desarrollo, constituyendo este un elemento importante para la revisión o la futura reutilización del componente propuesto.
- El uso de tecnologías guiadas por la filosofía del software libre favoreció en la búsqueda de la documentación y las herramientas que guiaron la realización de este trabajo, además de contribuir a la soberanía informática establecida por la universidad y el país.
- Como principal resultado se obtuvo una herramienta informática que resuelve la problemática relacionada con la integración de los frameworks Qt y OpenCascade respectivamente, además de obtener un soporte genérico que permita la elaboración de cualquier herramienta de tipo CAD a partir de estas tecnologías.

Recomendaciones

A partir de las experiencias acumuladas a lo largo del desarrollo de este trabajo, se proponen las siguientes recomendaciones:

- Implementar una nueva versión de la aplicación, en correspondencia con las nuevas actualizaciones de los frameworks OpenCascade y Qt, respectivamente.

- Estudiar la implementación de la herramienta “Salomé”, con el fin de eliminar la dependencia de una tarjeta de aceleración gráfica en las versiones posteriores de la aplicación.

- Generar una ayuda correspondiente a la aplicación propuesta, apoyándose en la herramienta Doxygen.

Bibliografía Consultada.

- Aik-Siong Koh, P. (2007) freeCAD 3D CAD with Motion Simulation.
- Booch, G., J. Rumbaugh, et al. (2000) El Lenguaje Unificado de Modelado.
- Borroso, A. M. (2009). Uso de la Librería Gráfica Conin3D.
- Carmona, P. R. (2008). Introducción a la Visualización 3D.
- Carracedo, J. d. P. (2010) visualización 3D.
- Culebro Juárez, M., W. G. Gómez Herrera, et al. (2006). "Software libre vs software propietario Ventajas y desventajas."
- Downey, A., J. r. Elkner, et al. (2002). Aprenda a Pensar Como un Programador con Python.
- Escribano, G. F. (2002). "Introducción a Extreme Programming."
- Ferreiro, A. M. F. and J. A. G. Rodríguez (2009). CURSO DE CAD/CAE CON SOFTWARE LIBRE SALOME.
- Galiano, F. B. (2007). Programación en Java.
- Garrido, S. A. (2009). Introducción a Qt. Programación gráfica en C++ con Qt4.
- González, S. G. (2004). Dibujo asistido con ordenador: teoría y prácticas de diseño con Solidworks
- Gracia, J. (2005) Patrones de Diseño. Diseño de Software Orientado a Objetos.
- Jones, N. G. J. M. (2008). Python for Unix and Linux System Administration.
- Méndez, A. V. (2010). "INVESTIGACIÓN DOCUMENTAL METODOLOGÍAS DE DESARROLLO DE SOFTWARE."
- Meyer, L. D. N. P. (2007). Introducción al desarrollo multiplataforma con Qt 4.
- Molpeceres, A. (2002). "Proceso de desarrollo: RUP, XP, FDD."
- opencascade.org (2007) Open CASCADE Technology, 3D modeling & numerical simulation.
- Orallo, E. H. (2006). "El Lenguaje Unificado de Modelado (UML)."
- Prata, S. (2002). C++ Primer Plus.
- Restrepo, C. A. R. (2010).
- Reynoso, C. B. (2004). Introducción a la arquitectura de software.

- Roberth G. Figueroa, C. J. S. and A. A. Cabrera (2010). METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES.
- Sánchez, M. A. M. (2004). Metodologías De Desarrollo De Software.
- Schildt, H. (1995). Guía de autoenseñansa.
- Summerfield, J. B. M. (2008). C++ GUI Programming with Qt 4.
- the_QtOCC_team (2007) QtOPENCASCADE Open Source CAD/CAE Framework.
- Visconti, M. and H. Astudillo (2011). "Fundamentos de Ingeniería de Software."
- wesley, A. (1999). Extreme Programing Emplained.
- wtk.org (2010) Visualization Toolkit (VTK).

Bibliografía Citada.

- Booch, G., J. Rumbaugh, et al. (2000) El Lenguaje Unificado de Modelado.
- Borroso, A. M. (2009). Uso de la Librería Gráfica Conin3D.
- Carracedo, J. d. P. (2010) visualización 3D.
- Méndez, A. V. (2010). "INVESTIGACIÓN DOCUMENTAL METODOLOGÍAS DE DESARROLLO DE SOFTWARE."
- opencascade.org (2007) Open CASCADE Technology, 3D modeling & numerical simulation.
- Reynoso, C. B. (2004). Introducción a la arquitectura de software.
- Summerfield, J. B. M. (2008). C++ GUI Programming with Qt 4.
- Visconti, M. and H. Astudillo (2011). "Fundamentos de Ingeniería de Software."