Universidad de las Ciencias Informáticas
Facultad 6

DESARROLLO DE UNA HERRAMIENTA DE GESTIÓN DE ARCHIVOS CMAKE PARA EL PROYECTO SIG-DESKTOP DEL DEPARTAMENTO DE GEOINFORMÁTICA

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

Autor: Lisandra Fuentes Oliveros

Tutor: Ing. Alberto Menendez Romero

Junio 2012

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Lisandra Fuentes Oliveros

_____

Firma del Autor

Ing. Alberto Menendez Romero

_____

Firma del Tutor

DATOS DE CONTACTO

Nombre y apellidos: Ing. Alberto Menendez Romero.

Correo electrónico: amenedez@uci.cu

Categoría docente: Instructor Recién Graduado.

Año de graduación: 2010.

Profesión: Ingeniero en Ciencias Informáticas.

Breve descripción: Graduado en la Universidad de las Ciencias Informáticas.

AGRADECIMIENTOS

Le agradezco a mi tutor Alberto Menendez Romero por haber sido tan paciente y exigente conmigo, además de un buen amigo, y a su novia Yeni que también lo fue, a mis padres por haberme apoyado en todo momento durante la carrera, a Rodolfo que además de ser mi compañero durante la carrera fue gran apoyo en los estudios y en los momentos difíciles, a Ivis y a Daniel por ser tan buenos amigos y estar ahí para mí en todo momento, a los amigos que me inspiraron para que no me detuviera y venciera todas las dificultades hasta el final de toda mi carrera, a todo aquel que aportó su granito de arena durante mi formación en la universidad.

DEDICATORIA

Le dedico este trabajo de diploma a mi familia, en especial a mis padres y a mis hermanos, y a mis amigos.

RESUMEN

Como parte de la evolución de las Tecnologías de la Información y la Comunicación, surgen los Sistemas de Información Geográfica (SIG), los cuales son más frecuentes en el uso cotidiano de las más variadas empresas. El proyecto SIG-Desktop de la universidad de las Ciencias Informáticas (UCI), desarrolla un SIG en forma de plataforma llamada GeoQ, cuyo objetivo es la representación y procesamiento espacial de la información en un entorno de escritorio. En la actualidad el equipo de desarrollo de GeoQ presenta dificultades en la configuración y compilación de la plataforma, proceso que se realiza especificando un conjunto de reglas en el lenguaje CMake con el objetivo de lograr un producto multiplataforma.

La presente investigación propone una solución informática para gestionar el proceso de configuración e instalación de los proyectos que se desarrollan con CMake en el proyecto SIG-Desktop, automatizando los pasos y reglas más comunes, además posibilita la edición de archivos CMake de una forma más intuitiva y fácil de entender. La solución es aplicable a otros proyectos realizados con CMake y en lenguaje de programación C++. La puesta en práctica del sistema propuesto deriva algunas ventajas importantes como la gestión de los archivos CMake de forma sencilla sin requerir amplios conocimientos y la reducción del tiempo empleado para configurar los pasos de compilación de los proyectos.

PALABRAS CLAVE

Configuración y compilación, CMake, CMakeLists, multiplataforma, Sistemas de Información Geográfica.

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

TABLA DE CONTENIDOS

INTRODUCCIÓN

A finales del siglo XV la imprenta fue la innovación tecnológica que revolucionó la comunicación e hizo posible la reproducción más eficiente de textos que permitieron compartir el conocimiento y trascender en el tiempo y el espacio, así como divulgar información a una velocidad jamás alcanzada antes por la humanidad. No tardaron en aparecer publicaciones regulares, ideadas a partir del afán del ser humano de mantenerse actualizado. Los medios para lograr dichos objetivos, fueron fundamentalmente la prensa, el telégrafo y el teléfono. En los últimos años, el uso de las llamadas Tecnologías de Información y Comunicación (TIC), que engloban a la prensa, la radio, la televisión, el cine y la red mundial; se ha incrementado. En especial cabe destacar el explosivo desarrollo de la Internet que permite la comunicación diferida o en tiempo real y es un servicio más que ofrece la World Wide Web (WWW).

Las TIC son aquellas herramientas que procesan, almacenan, sintetizan, recuperan y presentan información representada de la más variada forma. Son incuestionables y están siempre presentes, formando parte de la cultura tecnológica a que el ser humano está sujeto, además de que amplían las capacidades físicas y mentales y las posibilidades de desarrollo social. El constante desarrollo de las TIC, ha potenciado, entre otros, la creación de software que permiten interactuar de forma precisa con los mapas geográficos, surgiendo así lo que hoy se conoce como Sistemas de Información Geográfica (SIG o GIS por sus siglas en inglés).

Los SIG surgieron y se desarrollaron debido a la necesidad de almacenar, manipular y visualizar datos que tengan un componente geográfico. Desde su aparición tuvieron como objetivo permitir a los usuarios crear consultas interactivas, analizar la información espacial, editar mapas y presentar los resultados de todas estas operaciones; en la actualidad estos sistemas son utilizados por muchas empresas debido a su gran utilidad.

Los SIG son sistemas que brindan la posibilidad de realizar un conjunto de funcionalidades ya sea realizar búsquedas, consultas a la cartografía, determinar caminos mínimos entre dos o varios puntos de interés, cálculo de distancia y otras operaciones. Estos aportan una perspectiva totalmente nueva y dinámica a la información.

Cuba desde hace años ha mantenido una política de desarrollo y expansión de estas tecnologías, en la búsqueda de soluciones que permitan incrementar los factores de eficiencia y eficacia en las organizaciones y empresas, logrando grandes avances en el desarrollo de software, principalmente con tecnologías libres. Diversos organismos e instituciones dirigen sus esfuerzos a potenciar el uso de los SIG priorizando así diferentes puntos estratégicos, como son: el transporte, la telefonía, la minería, la meteorología, incluso en la salud.

La Universidad de las Ciencias Informáticas (UCI) como parte del Programa de Informatización de la Sociedad incluyó desde el curso 2006–2007 en su ámbito de trabajo los SIG, pretendiendo poner en el mercado una infraestructura que garantice la representación de datos espaciales en diferentes escenarios. De este esfuerzo surgieron varias soluciones que dan cumplimiento a las necesidades actuales en distintos ámbitos, ejemplo de ello la plataforma GeneSIG y la plataforma GeoQ.

La plataforma GeoQ es un Sistema de Información Geográfica tipo Escritorio desarrollado por el proyecto SIG-Desktop en el departamento de Geoinformática del Centro de Desarrollo GEySED y ha sido elaborada tomando como base al software Quantum GIS, el cual es una aplicación de código abierto y con amplio soporte por la comunidad internacional que se encarga de las actualizaciones y mejoras constantes, así como de la incorporación de nuevas funcionalidades.

El equipo de desarrollo de GeoQ se encarga de dar respuestas a los más variados negocios adaptando las funcionalidades ya implementadas a las necesidades del cliente, logrando de este modo vincular las funciones SIG con las operaciones específicas del negocio. Para ello, el equipo de desarrollo realiza un conjunto de modificaciones a la plataforma como pueden ser, eliminar parte de sus funcionalidades que no necesite el cliente, añadir nuevas funcionalidades como parte del negocio, realizar modificaciones a la interfaz gráfica con cambios de colores, reordenamiento e inclusión de nuevas ventanas, adaptar el proceso de configuración y compilación de la plataforma para la arquitectura requerida por el cliente, entre otras variadas.

La configuración y compilación de la plataforma GeoQ se maneja con la ayuda de herramientas disponibles para la creación y compilación de proyectos, tal es el caso de CMake, el cual es una herramienta multiplataforma para la generación y automatización de código. Utilizar a CMake como mecanismo para la compilación garantiza la portabilidad del código y la independencia del sistema

operativo del usuario, lo cual es factible para poder llegar a un rango de cliente más variado. (Molina, 2009)

Con la ayuda de CMake, el equipo de desarrollo puede incluir dependencias de componentes de terceros ya desarrollados, introducir nuevas funcionalidades en forma de módulo a la plataforma, regular el proceso de compilación creando variables, definiendo restricciones de la arquitectura, especificando bibliotecas externas, así como otras tareas de rutina. El uso de la herramienta CMake le permite al equipo de desarrollo de GeoQ centrarse únicamente en implementar nuevas funcionalidades y en la demanda de los clientes, ya que el código no necesita “preocuparse” por especificidades de la arquitectura ni dar soporte para alguna en específico.

De forma general, se considera a CMake como una familia de herramientas diseñada para construir, probar y empaquetar el software que utilizando un lenguaje de descripción de proyectos se puede añadir, modificar y eliminar dependencias con el mínimo esfuerzo necesario, brindando al equipo de desarrollo el control sobre todo el proceso de producción de software. (Garrido, 2012)

La descripción de un proyecto CMake se establece a través de un conjunto de archivos CMakeLists.txt los cuales estarán presente en cada directorio del código fuente a compilar. Cada archivo CMakeLists.txt contiene definiciones en forma de comandos que indican o describen cómo debe proceder la compilación del proyecto. El lenguaje utilizado por CMake es muy rico y descriptivo ya que cuenta con los elementos necesarios para definir cualquier proyecto. En proyectos pequeños realizar la descripción en el lenguaje CMake no resulta complejo; sin embargo la configuración de la plataforma GeoQ y adaptación al sistema operativo final puede resultar muy complejo debido al gran volumen de funcionalidades implementadas y los cambios que se necesitan efectuar para adaptar GeoQ a un cliente.

En la actualidad, el equipo de desarrollo de GeoQ debe llevar el proceso de configuración y empaquetado con CMake de forma manual lo cual resulta un proceso complejo por la dimensión de GeoQ; además se debe tener parte del equipo de desarrollo especializado en el lenguaje CMake por lo que se requiere una utilización extra del personal. El proceso de configuración manual acarrea en gran medida que el proceso de desarrollo se alargue ya que en ocasiones es lento y puede dar lugar a errores que son complejos de depurar, tales como: errores al incluir dependencias a bibliotecas que cuentan con varias versiones instaladas, errores lógicos de expresiones condicionales, definiciones de

bloques únicos para determinadas plataformas, inclusión de archivo para compilación cuyo nombre debe ser escrito con exactitud, y otros errores. Es por tales motivos, que surge la necesidad de buscar alternativas que permitan automatizar este tipo de tareas y reduzcan el tiempo necesario para configurar y empaquetar la plataforma GeoQ.

De lo expuesto anteriormente, se identificó el siguiente **problema a resolver** el cual origina esta investigación:

La complejidad en la organización manual de archivos CMake dificulta el proceso de compilación en el proyecto SIG-Desktop.

Como resultado del análisis del problema se definió como **objeto de estudio** el proceso de gestión de archivos CMake en el proyecto SIG-Desktop, específicamente en la automatización del proceso de gestión de archivos CMake en el proyecto SIG-Desktop, lo que constituye el **campo de acción** de la investigación.

Se define como **objetivo general** del presente trabajo de diploma desarrollar una herramienta que permita automatizar la gestión de los archivos CMake en el proyecto SIG-Desktop.

Como parte de la investigación se propone la siguiente **idea a defender**:

El desarrollo de una herramienta para automatizar la gestión de archivos CMake en el proyecto SIG-Desktop permitirá mejorar la configuración y compilación de sus proyectos.

Para darle cumplimiento a los objetivos trazados se desarrollarán las siguientes **tareas** durante la investigación:

1. Caracterizar el proceso de gestión de archivos CMake en el proyecto SIG-Desktop.
2. Realizar una revisión de las tendencias y tecnologías actuales para desarrollar la solución propuesta.
3. Definir y especificar los requisitos funcionales del software.
4. Modelar los procesos de negocio, casos de uso del sistema, diagramas de análisis, diseño e implementación.
5. Implementar funcionalidades definidas

6. Desarrollar los casos de pruebas que certifiquen la veracidad de los algoritmos empleados.

Como parte del cumplimento de las tareas investigativas se espera obtener los siguientes **resultados**:

1. Una herramienta informática para automatizar la gestión de los archivos CMake.
2. La documentación técnica asociada a la construcción de la herramienta.
3. Manual de usuario para el uso de la herramienta.
4. Código fuente de la herramienta implementada.

En el trascurso y desarrollo de esta investigación científica se tendrán en cuenta una serie de **métodos científicos** para la obtención, procesamiento y llegada a conclusiones, los cuales se exponen a continuación:

Métodos teóricos

Este tipo de método permite estudiar las características del objeto de investigación que no son observables directamente, facilitan la construcción de modelos e hipótesis de investigación y crean las condiciones para ir más allá de las características fenomenológicas y superficiales de la realidad, contribuyendo al desarrollo de las teorías científicas y para su ejecución se apoyan en el proceso de análisis y síntesis (Herández León, y otros, 2011).

- **Analítico-Sintético**: Se utilizará para el estudio de la bibliografía y la selección de la más adecuada para el desarrollo del presente trabajo; además, para dividir el problema de investigación en elementos por separado, profundizar en el estudio de cada elemento y luego sintetizarlos en la confección de la solución propuesta.
- **Histórico-Lógico**: Se utilizará para el estudio de los procesos de gestión de archivos CMake, trabajos e investigaciones anteriores y partir de ellos como base para el desarrollo de la presente investigación.
- **Modelación**: La modelación es el método mediante el cual se crean abstracciones con el objetivo de explicar la realidad. Se utilizará en la modelación de los diagramas dentro de la metodología de desarrollo de software seleccionada para llevar a cabo la solución.

El presente trabajo consta de **4 capítulos** que están divididos en epígrafes y subepígrafes.

Capítulo 1. Fundamentación Teórica: En este capítulo se analizan conceptos asociados al campo de acción de la investigación, la descripción del objeto de estudio identificado y los posibles resultados asociados al tema en cuestión, para así poder llegar a las posibles vías de solución del problema a resolver.

Capítulo 2. Características del Sistema: En este capítulo se explican las principales tecnologías, lenguajes de programación y herramientas que se utilizarán para la construcción de la solución propuesta, así como las ventajas de utilizarla.

Capítulo 3. Análisis y Diseño del Sistema: Se describe la propuesta de solución, la cual incluye el modelo de dominio, la identificación de los requisitos funcionales y no funcionales, la modelación de los diagramas de casos de uso del sistema, de los diagramas de clase del diseño y los diagramas de implementación.

Capítulo 4. Implementación y Pruebas: Se describe todo el proceso de construcción de la propuesta de solución. Se realizan todo el diseño de caso de prueba por cada caso de uso y posteriormente se ejecutan las pruebas necesarias para verificar los resultados alcanzados.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En este capítulo se analizan conceptos asociados al campo de acción de la investigación, se realiza una descripción del objeto de estudio identificado, posibles resultados asociados al tema en cuestión para así poder llegar a las posibles vías de solución del problema a resolver. Al finalizar el capítulo se dejan sentadas las bases necesarias para un correcto análisis durante el resto de la investigación.

1.1 Conceptos asociados al dominio del problema

Los conceptos que a continuación se exponen tienen como objetivo establecer un lenguaje común y un mejor entendimiento de la investigación.

Gestión de archivos

En informática, un sistema de gestión de archivos es aquel sistema de software que provee servicios a los usuarios y aplicaciones en el uso de archivos. El único camino que tiene el usuario o la aplicación para acceder a los archivos es a través de un sistema de gestión de archivos. Entre los objetivos de un sistema de gestión de archivos se pueden mencionar (UNE-ISO-15489-1, 2005):

- Cumplir con las necesidades de gestión de datos y con los requisitos del usuario, que incluye el almacenamiento de datos y la capacidad de ejecutar las operaciones en la lista precedente.
- Garantizar, en la medida de lo posible que el dato en el archivo es válido.
- Optimizar el rendimiento desde el punto de vista del sistema en términos de productividad global y como punto de vista del usuario en tiempos de respuesta.
- Minimizar o eliminar la posibilidad de pérdida o destrucción de datos.

Este trabajo define un sistema de gestión de archivos como el software que proporciona tanto a los usuarios como a las aplicaciones servicios para el uso y acceso de los archivos. Este brinda la posibilidad de abrir, crear, modificar o eliminar un archivo de un proyecto.

Multiplataforma

La Real Academia Española en su vigésima tercera edición, define el concepto de multiplataforma para referirse a una aplicación o producto informático que puede ser utilizado por distintos sistemas o entornos. (RAE, 2001)

En términos informáticos, Alegsa define multiplataforma como la capacidad para soportar y funcionar de forma similar múltiples plataformas (Alegsa, 2012).

El autor del presente trabajo asume el concepto de multiplataforma para referirse al software o sistema informático que tiene la característica y habilidad para ejecutarse en distintas plataformas o sistemas operativos de forma similar sin requerir muchos cambios.

CMake

CMake es una herramienta desarrollada inicialmente para poder compilar de forma multiplataforma la biblioteca VTK como un desarrollo interno, pero que ha sido publicada como herramienta de Software Libre (Molina, 2009). Se puede considerar como un sistema de generación de scripts de compilación (build system) que utiliza ficheros de configuración independientes de la plataforma y del compilador para generar proyectos. CMake no sólo permite la compilación de proyectos, sino que cubre todo el proceso de desarrollo de software, desde creación del proyecto, compilación, lanzamiento de pruebas, y empaquetado (Garrido, 2012).

CMakeLists.txt

El CMakeLists.txt es un archivo que describe la configuración y define la compilación de un proyecto CMake. El proceso de construcción se controla creando uno o más ficheros CMakeLists.txt en cada directorio (incluyendo subdirectorios), donde cada archivo está compuesto por varios comandos y estos comandos a su vez contienen un conjunto de argumentos. Los comandos en un archivo CMakeLists.txt definen las instrucciones necesarias para compilar la porción de código especificada.

A grandes rasgos, los archivos CMakeLists.txt son los encargados de brindar toda la información necesaria para la correcta compilación del proyecto. En un archivo CMakeLists.txt se puede encontrar: (Molina, 2009)

- El listado de ficheros de código fuente a compilar.

- La lista de bibliotecas y ejecutables que genera el proyecto.
- Dependencias con bibliotecas.
- Programas externos a utilizar.

Traductor

Un traductor se define como un programa que traduce o convierte desde un texto o programa escrito en un lenguaje fuente hasta un texto o programa equivalente escrito en un lenguaje destino produciendo, si cabe, mensajes de error. Los traductores engloban a los compiladores como a los intérpretes. (Álvares Rojas, y otros, 2005)

Compilador

El término compilador se emplea en el ámbito informático para referirse a un intérprete que tiene como entrada un secuencia en lenguaje formal y como salida tiene un fichero ejecutable, es decir, realiza una traducción del código de alto nivel a un código máquina (también se entiende por compilador aquel programa que proporciona un fichero objeto en lugar del ejecutable final). (Álvares Rojas, y otros, 2005)

Otro concepto de compilar aceptado por esta investigación es el siguiente: programa informático que permite traducir el código fuente de un programa en lenguaje de alto nivel, a otro lenguaje de nivel inferior (típicamente lenguaje máquina). El trabajo de un compilador consiste en tomar la cadena fuente del programa, determinar si es válida desde el punto de vista sintáctico y, a la vez, generar un programa equivalente en un lenguaje que la computadora entienda. (Acevedo Martínez, y otros, 2011)

El trabajo del compilador se puede dividir varias partes conocidas como fases del proceso de compilación, entre las más importantes destacan el análisis léxico, análisis sintáctico y análisis semántico.

Analizador léxico o Scanner

Un analizador léxico puede definirse como un traductor cuya entrada es una cadena de símbolos (programa fuente) y cuya salida es una secuencia de estructuras lexicológicas o tokens. La entrada de un compilador es el código de un programa escrito en un lenguaje de programación compuesto por un

conjunto de instrucciones del lenguaje. Al analizar el código fuente como una secuencia de símbolos (caracteres), la función del analizador léxico o scanner, seria tomar cada uno de esos símbolos y agruparlos en entidades sintácticas simples o elementales denominadas tokens. (Acevedo Martínez, y otros, 2011)

Las categorías de tokens pueden variar de un lenguaje a otro, pero en general se distinguen las siguientes:

- Palabras reservadas
- Identificadores
- Constantes numéricas y literales
- Operadores
- Delimitadores

Analizador sintáctico o Parser

El análisis sintáctico es el proceso en el cual se examina una secuencia de tokens para determinar si el orden de esa secuencia es correcto de acuerdo a ciertas convenciones estructurales (reglas) de la definición sintáctica del lenguaje. La entrada del analizador sintáctico o parser es la secuencia de tokens generada por el scanner y la salida es, por un lado, la indicación del cumplimiento de las reglas gramaticales que definen al lenguaje, y por el otro, producir una representación intermedia que permita la ejecución de las fases restantes. (Acevedo Martínez, y otros, 2011)

Analizador semántico

El analizador semántico recibe la información resultado del análisis sintáctico que puede ser un árbol con la información relativa a la organización jerárquica gramatical de los tokens en la instrucción que se analiza. En la fase de Análisis Semántico se detectan errores relacionados con la validez del programa. Se puede decir que estos errores son de tipo sintáctico-semántico, pero no pueden ser detectados por el analizador sintáctico, ya que se relacionan con interdependencias entre las diferentes partes de un programa que no son reflejadas en un análisis gramatical. (Acevedo Martínez, y otros, 2011)

Intérprete

El intérprete es un programa que se comporta de forma similar a un compilador a diferencia de que la salida producida es la ejecución del programa o lenguaje que reconoce, donde el programa de entrada es reconocido y ejecuta a la vez. No se produce resultado físico, sino lógico. La principal ventaja de un intérprete es que permite una fácil depuración durante la construcción. Como principal desventaja es que en ocasiones pueden llegar a ser lentos y consumir gran volumen de memoria. (Álvares Rojas, y otros, 2005)

SIG Desktop

SIG-Desktop es un proyecto del Centro de Investigación y Desarrollo GEySED, en la facultad 6 de la Universidad de las Ciencias Informáticas. La misión principal de este proyecto es la creación de una plataforma SIG que permita la rápida respuesta a los negocios que exigen un componente SIG en sus sistemas.

GeoQ

Es el Sistema de Información Geográfica desarrollado por el proyecto SIG-Desktop y tiene como objetivo brindar variadas soluciones a los negocios a través de la creación de personalizaciones. GeoQ basa su desarrollo en el sistema Quantum GIS[1], el cual es un SIG de código abierto desarrollado en C++ con Qt. En la actualidad GeoQ puede ser ejecutado en plataformas Linux, Windows y MacOsX y su equipo de desarrollo centra los esfuerzos en la mejora de la usabilidad del sistema y la integración de los conceptos de negocios de sus clientes.

1.2 Objeto de Estudio

1.2.1 Descripción General

El proyecto SIG-Desktop del centro de desarrollo GEySED en la Universidad de las Ciencias Informáticas desarrolla el Sistema de Información Geográfica GeoQ el cual es considerado una

[1]Sistema de Información Geográfica de carácter general, libre y de código abierto con amplio soporte por la comunidad internacional.

plataforma SIG ya que a partir de este se realizan adaptaciones para dar respuesta a distintas necesidades de negocio.

La plataforma GeoQ está basada en Quantum GIS, el cual es un Sistema de Información Geográfica libre y de código abierto mantenido internacionalmente por una comunidad encargada de la implementación de nuevas funcionalidades.

Como la mayoría de los proyectos de código abierto en el mundo, se hace necesario el uso de herramientas que permitan la configuración y compilación del sistema. Para el caso de Quantum GIS se utiliza la herramienta CMake la cual establece un conjunto de reglas que indican cómo debe configurarse y compilarse un proyecto. Por tal motivo, los desarrolladores de GeoQ deben hacer uso de CMake para configurar y compilar cada una de las soluciones SIG que se obtienen a partir de la plataforma, haciendo uso de los archivos CMakeLists.txt.

Como se mencionó anteriormente en este trabajo, un archivo CMakeLists.txt define a través de comandos el proceso de compilación de un proyecto; donde se tiene un archivo CMakeLists.txt general que engloba a todo el proyecto y varios archivos CMakeLists.txt distribuidos por cada directorio que conforma al proyecto. A continuación se expone la estructura y organización de los archivos CMake para un proyecto de ejemplo llamado Hello.

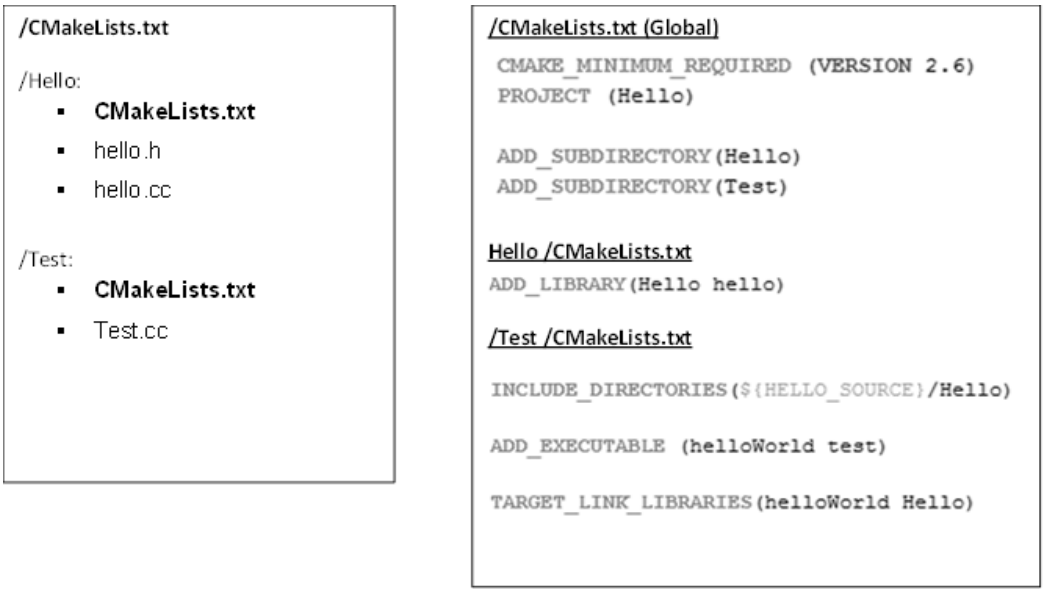


Imagen 1. Estructura y organización de un proyecto de ejemplo desarrollado con CMake

En el fragmento de código anterior se puede apreciar que el proyecto se encuentra estructurado en 2 directorios principales Hello y Test. El uso del comando PROJECT permite especificar el nombre del proyecto de forma global. El comando ADD_SUBDIRECTORY permite la inclusión de los directorios donde se tiene código para la compilación. El comando TARGET_LINK_LIBRARIES se utiliza para la compilación de una biblioteca.

1.2.2 Situación problemática

El nivel de uso de los comandos de CMake depende en gran medida de las dimensiones del sistema que se configura y puede llegar a ser una tarea tediosa, que ocupe mucho tiempo. Para el caso de la plataforma GeoQ se cuenta con un elevado número de archivos fuentes, dependencias y estructura de directorio donde la configuración del proyecto con CMake se torna complejo.

Cuando se realiza una personalización de GeoQ para un cliente, se crea una copia local de la plataforma base de GeoQ y a partir de dicha copia se comienza la configuración de la solución a la medida para el cliente. Configurar el proyecto significa en determinadas ocasiones quitar un conjunto de archivos con funcionalidades que no necesita el cliente, en ese caso, debe especificarse en el archivo CMakeLists.txt correspondiente para que no se realice la compilación de dichos archivos. La inclusión de nuevas funcionalidades y bibliotecas de clases para la solución que se realiza requieren la modificación de los archivos de CMake para especificar dichas dependencias; igualmente se debe adaptar la configuración del proyecto para el sistema operativo final, incluyendo un conjunto de instrucciones condicionales que ajustan el comportamiento de la plataforma para dicho sistema operativo.

La configuración manual de los archivos CMake por parte del grupo de desarrollo de GeoQ se puede caracterizar como un proceso lento, tedioso y propenso a la ocurrencia de errores que en ocasiones resulta complejo detectarlos. Las soluciones brindadas por la plataforma a sus clientes, deben ser configuradas manualmente para su posterior compilación; por tal motivo se hace necesaria la búsqueda de una herramienta que permita gestionar los archivos CMake de forma automatizada y que elimine las deficiencias mencionadas.

La presente investigación centra sus esfuerzos en la búsqueda de una solución desde el punto de vista informático que permita la automatización de las tareas más comunes y frecuentes ejecutadas en la

fase de configuración de la plataforma y por ende en la edición de los archivos CMake en el proyecto SIG-Desktop.

1.3 Análisis de soluciones existentes o semejantes

CMake Editor

CMake Editor o editor de CMake, es un plugin[2] que se conecta al Entorno de Desarrollo Integrado Eclipse para la manipulación de archivos CMake. Entre las características de esta solución destacan (CMake-Project, 2012):

- La distinción de la sintaxis con la ayuda de diferentes colores.
- Asistente de contexto para el completado de funciones básicas, comandos, variables, etc.
- Funciones de edición básica de los archivos.

Esta solución tiene amplia aceptación en el mundo debido a su libre licencia y soporte por parte de la comunidad. A pesar de las ventajas expuestas anteriormente, se considera que no es una solución factible para la investigación debido a su dependencia con el Entorno de Desarrollo Integrado para el cual se diseñó y por otro lado a que sus prestaciones sólo se limitan a las funcionalidades básicas de edición (resaltar sintaxis, copiar, pegar, buscar y reemplazar, abrir, guardar, etc.), donde los comandos de CMake deben ser escritos por el usuario y no se automatizan las tareas más comunes o cotidianas en un proceso de configuración.

Extensión para la edición de CMake en Visual Studio

Esta solución es una extensión o plugin diseñada específicamente para el Entorno de Desarrollo Integrado Visual Studio a partir de su versión 2010. Posee características básicas como el resaltado de sintaxis, el completamiento de comandos, operadores y funciones de CMake. En adición, la extensión permite la inclusión de módulos definidos que son ampliamente utilizados en proyectos construidos con CMake, facilitando de esto modo la especificación de dependencias. (vissemee, 2012)

[2]Componente informático que adiciona funcionalidades específicas a la aplicación para el cual se diseñó.

Como principal desventaja de la solución destaca la necesidad de comprar una licencia para la versión de pago de Visual Studio, ya que la extensión no trabaja correctamente en una versión libre o express de este Entorno. Otra desventaja de la solución la constituyen las pocas funcionalidades que brinda para le edición de los archivos CMake, limitándose solo a la edición en texto.

Por estas desventajas, la solución solo se tendrá en cuenta para un estudio e identificación de posibles prestaciones en la solución que se propone al problema identificado en la investigación.

Editores de CMake

Existen múltiples programas de edición que tienen soporte para la edición de archivos CMake, se pueden mencionar editores populares como Notepad++ para Windows y GEdit para plataformas GNU/Linux. De forma general estos editores se limitan solo a resaltar la sintaxis con colores y en pocas ocasiones al completamiento de funciones básicas. De forma general, estos editores no conciben el concepto “proyecto CMake” por lo que su tratamiento es puramente textual.

1.4 Conclusiones parciales

En este capítulo se expusieron los principales conceptos y términos por los cuales se rige la investigación, en muchos casos se mencionaron otros conceptos definidos y que se consideran válidos. Se dejó plasmado un análisis del problema identificado por la investigación y la necesidad de proponer una solución a dicho problema.

Durante la investigación se realizaron estudios en búsqueda de soluciones existentes que pudieran aportar de forma general una respuesta o punto de partida para el problema identificado. El autor del presente trabajo concluye en que la investigación actual constituye la primera solución real con vista a resolver el problema identificado y de forma general la primera de su tipo aplicada a otros contextos.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

El presente capítulo describe las tecnológicas a utilizar para la construcción de la herramienta propuesta, que incluye: tipos de aplicaciones, metodologías de desarrollo de software, herramientas de modelado y lenguajes de programación; las principales ventajas y la opinión del autor de la actual investigación en cada caso.

2.1. Aplicación Informática

En un ordenador el software representa las instrucciones responsables para que el hardware (la máquina) realice su tarea. Como concepto general, el software puede dividirse en varias categorías basadas en el tipo de trabajo realizado: por una parte, el sistema operativo que controla los trabajos del sistema, y por otra, el software de aplicación que dirige las distintas tareas que se requiera.

Un programa informático, sinónimo de software, es el conjunto de instrucciones que ejecuta un ordenador o computadora donde una vez ejecutadas realizarán una o varias tareas (STAIR, 2003); también suele referirse al código fuente original o a la versión ejecutable.

Una aplicación informática se puede definir como un programa informático diseñado para facilitar al usuario la realización de un determinado tipo de trabajo. Generalmente resultan ser una solución informática para la automatización de ciertas tareas complicadas como pueden ser la contabilidad, la redacción de documentos, la gestión de un almacén, la comunicación de datos, entre otros.

2.1.2 Aplicación de escritorio

Cuando se refiere al término aplicación de escritorio se está en presencia de una aplicación informática que se ejecuta en una computadora de escritorio o en una portátil (laptop) de manera local, en contraste a las aplicaciones web (Menendez Romero, 2010).

Este tipo de aplicación informática se ejecuta en el ordenador en su propia dirección de memoria física o virtual y no requiere cargarse en otras aplicaciones como navegadores para poder funcionar. No obstante, una aplicación de escritorio puede encontrarse distribuida en varios ordenadores y comunicarse a través de diferentes mecanismos o protocolos de la red. (Menendez Romero, 2010)

2.2 Arquitectura de software

La arquitectura de software, ha emergido como una disciplina de gran importancia dentro de la ingeniería de software y su adecuada selección es la pieza clave para lograr el cumplimiento de los requerimientos funcionales como no funcionales de un sistema; su representación y diseño se han convertido en temas dominantes de la ingeniería de software; donde encontrar una definición aceptada universalmente no es tarea fácil.

Toda aplicación contiene código de presentación, código de procesamiento de datos y código de almacenamiento de datos. La arquitectura de las aplicaciones difiere según como está distribuido este código (Cornejo, 2001). En los sistemas grandes y complejos la arquitectura de software brinda al equipo de desarrollo una visión común, ayuda a comprender la estructura del sistema, organizar el desarrollo del mismo, así como fomentar la reutilización y tomar en cuenta las posibles evoluciones del sistema.

La definición de la IEEE[3]Std 1471-2000 sobre el término arquitectura de software plantea que "…es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución".

2.2.1 Arquitectura en 3 Capas

La programación en capa es un rasgo innovador de las aplicaciones informáticas actuales que buscan un diseño y desempeño robustos. Este tipo de programación permite la obtención de un repositorio de objetos para futuros desarrollos posibilitando la reducción de tiempo, coste y mantenimiento. Una arquitectura común de los sistemas informáticos que abarca una interfaz para el usuario y el almacenamiento de datos es la arquitectura de tres capas (Schulte, 1995). Este tipo de arquitectura separa los datos de una aplicación, la interfaz de usuario y la lógica en tres componentes o capas distintas.

[3]Institute of Electrical and Electronics Engineers.

La calidad tan espacial de la arquitectura de tres capas consiste en aislar la lógica de la aplicación y en convertirla en una capa intermedia bien definida y lógica del software. En la capa de presentación se realiza relativamente poco procesamiento de la aplicación (Schulte, 1995). A su vez, la capa de la lógica de la aplicación está constituida por otras capas menos densas y su división depende de las características propias del sistema que se pretende construir; sin embargo, es común contar con las capas de Objetos del Domino donde se representan los conceptos del dominio y la capa de Servicio donde se representan las funciones que brinda el sistema como: reportes, control del flujo de la información, etc.

De forma general se identifican tres capas bien definidas:

La capa de presentación: Es la capa encargada de encerrar todas las interacciones con el usuario u otro sistema informático; recibe las peticiones y solicitudes que se realiza al sistema y mantiene la gestión de cómo se representan los datos.

La capa de la lógica: Como se explicó anteriormente, es la capa encargada de dar respuesta a las peticiones del usuario, recibe las solicitudes realizadas por la capa de presentación. Gestiona el flujo de la información del sistema y realiza los cálculos pertinentes para arribar a resultados.

La capa de datos: La capa de datos, también conocida como capa de acceso a datos; mantiene el control sobre el almacenamiento de la información que puede ser persistente o no. En esta capa se tienen los mecanismos necesarios para acceder a los datos de la aplicación y la gestión de estos.

La investigación actual se proyecta por el uso de una arquitectura de tres capas debido a las ventajas que aporta esta arquitectura en la construcción de la solución, entre ellas:

- Las llamadas a la interfaz y la lógica de la aplicación es flexible pues solo necesita transferir parámetros a la capa intermedia.
- La lógica de la aplicación puede ser dividida en sub-capas para aislar los conceptos del dominio y la gestión de dichos conceptos.
- El reemplazo de una capa o parte del sistema no afecta a todas las capas del sistema. Además esta arquitectura posibilita la inclusión de nuevas funcionalidades sin afectar el comportamiento actual del sistema.

2.3 Lenguajes de programación

La Real Academia Española define el término lenguaje como el “conjunto de sonidos articulados con que el hombre manifiesta lo que piensa o siente” o al “conjunto de señales que dan a entender algo” y se define para el ámbito informático como el “conjunto de signos y reglas que permite la comunicación con un ordenador.” (RAE, 2008)

Un lenguaje de programación se puede definir como un idioma artificial diseñado para expresar tareas a realizar por máquinas como las computadoras, que puede usarse para crear programas que controlen el comportamiento físico y lógico de la máquina, para expresar algoritmos con precisión, entre otros.

Los lenguajes de programación son las herramientas necesarias que permiten la creación de los programas informáticos; están formados por un conjunto de símbolos y reglas que definen su estructura y el significado de sus elementos y expresiones.

Un lenguaje de programación puede clasificarse en alto nivel o bajo nivel. En los de bajo nivel las instrucciones son simples y cercanas al funcionamiento de la máquina, como por ejemplo el código máquina y el ensamblador. En los lenguajes de alto nivel hay un alto grado de abstracción y el lenguaje es más próximo a los humanos, como por ejemplo PASCAL, Cobol o Java.

2.3.1 Lenguaje de programación CSharp (C#)

El lenguaje de programación CSharp (conocido como C#) fue creado por Microsoft y lanzado a mediados del año 2000 como parte estratégica de la plataforma de desarrollo .NET, donde se considera el más depurado y optimizado de esta plataforma.

Este lenguaje incorpora las mejores características de lenguajes populares como C, C++ y Java; eliminando los problemas que pudieran acarrear el desarrollo de sistemas en uno de estos lenguajes, por lo que su aceptación y evolución se produjo con rapidez. Es considerado un lenguaje de alto nivel incorporando nuevos conceptos que lo caracterizan como un lenguaje orientado a componentes ya que soporta completamente el desarrollo de sistemas basados y distribuidos en componentes.

Como todo lenguaje de programación de propósito general actual, C# es un lenguaje orientado a objetos. Una diferencia de este enfoque orientado a objetos respecto al de otros lenguajes como C++ es que el de C# es más puro en tanto que no admiten ni funciones ni variables globales sino que todo el código y datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código (Microsoft, 2011).

Los programas realizados en C# son compilados internamente a código manejado, el cual es un lenguaje intermedio caracterizado por estar entre un alto nivel y el bajo nivel, para posteriormente ser compilado a la arquitectura específica de la plataforma donde se desarrolla; de este modo, los programas realizados podrán ser portados a distintas plataformas sin necesidad de cambios (Microsoft, 2011).

2.4 La plataforma Microsoft .NET

Microsoft. Net constituye el conjunto de tecnologías que ha desarrollado Microsoft en los últimos años con el objetivo de crear una plataforma sencilla y potente que permita distribuir software en forma de servicios, garantizando que puedan comunicarse y combinarse unos con otros de manera totalmente independiente de la plataforma, del lenguaje de programación y del modelo de componentes con los que hayan sido desarrollados (Mamone, 2006).

Microsoft .NET es considerada una infraestructura de varias tecnologías que conforman un entorno para desarrollar (tanto servicios Web como aplicaciones tradicionales, aplicaciones de consola, aplicaciones de ventanas, servicios de Windows NT, etc.), y ejecutar aplicaciones escalables y distribuidas. En la actualidad es considerada como la competencia de plataforma J2EE incorporando nuevas técnicas y brindando soporte a más de 40 lenguajes de programación (C#.NET, C++.NET, Python.NET, Java.NET, ASP.NET, Ruby.NET, etc.) (Mamone, 2006).

En esta plataforma el programador no trabaja directamente contra un sistema operativo, sino que lo hace frente al Common Language Runtime (CLR) o Máquina Virtual Común para garantizar la portabilidad de la aplicación y la interoperabilidad de los lenguajes; ya que es el CRL el encargado de adaptar y ejecutar el programa de acuerdo a la arquitectura. El CLR ofrece vario servicios durante el desarrollo de las aplicaciones, entre ellas (Rammer, y otros, 2005):

- Tratamiento homogéneo de errores mediante excepciones.
- Desarrollo interlenguaje e interoperabilidad con código antiguo permitiendo la reutilización de componentes realizados en otros lenguajes.
- Ejecución multiplataforma donde programas para una plataforma genérica y en fase de compilación se traducen las especificaciones correspondientes.
- El CLR proporciona un sistema automático de administración de memoria denominado recolector de basura (Garbage Collector) que detecta cuándo el programa deja de utilizar la memoria y la libera automáticamente. De esta forma el programador no tiene que liberar la memoria de forma explícita y no preocuparse por errores comunes en la programación como: volcados de memoria, desbordamientos de la pila del sistema, invasión de marcos de memoria, etc.
- Soporte multihilo para la programación en paralelo de procedimientos complejos.
- Seguridad avanzada basada y protección de los componentes que certifiquen el origen de procedencia.

Una de las características más importantes de esta plataforma es la Biblioteca de Clases, la cual provee acceso a las operaciones más comunes de Entrada/Salida, intercambio de mensajes con el Sistema Operativo, comunicación con sistemas externos, etc., abstrayendo al programador de las especificaciones del sistema operativo.

2.5 La plataforma Mono

Lanzado en el año 2005 por Miguel de Icaza, la plataforma Mono puede considerarse como un intento de traer la plataforma .NET de Microsoft a un entorno libre y de código abierto. En estos momentos la plataforma cuenta con soporte para variadas distribuciones de GNU/Linux, Microsoft y Mac OsX, donde su desarrollo se mantiene a través de la comunidad internacional.

El núcleo de la plataforma Mono ha sido diseñado para mantener y poder ejecutar las aplicaciones desarrolladas en .NET, a su vez cuenta con la implementación de un Common Languaje Runtime (CLR) que simula el funcionamiento del CLR de Microsoft (Mamone, 2006). El ambiente de desarrollo de Mono incluye además un conjunto de librerías que constituye la biblioteca de clases de .NET manteniendo uniformidad de los espacios de nombre y el nombre de las clases para garantizar de este modo la portabilidad de la aplicación.

La evolución de la plataforma y el apoyo recibido por la comunidad ha contribuido a que actualmente Mono sea considerado una plataforma seria y robusta para el desarrollo de aplicaciones de escritorio y web, incluyendo conceptos avanzados como Remoting, ASP.NET Forms, XML y Serialización. Sin embargo, en la actualidad Mono solo es capaz de ejecutar con toda seguridad aplicaciones realizadas con la versión 2.0 de la plataforma .NET (Mamone, 2006).

2.6 Entorno de Desarrollo Integrado

Un entorno de desarrollo integrado (IDE, del inglés Integrated Development Environment), es un programa informático compuesto por un conjunto de herramientas de programación, que puede dedicarse a uno o varios lenguajes de programación y ha sido empaquetado como un programa de aplicación compuesto por un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de forma amigable. (Nourie, 2005)

2.6.1 Microsoft Visual Studio como Entorno de Desarrollo Integrado

Microsoft Visual Studio es un Entorno de Desarrollo Integrado para sistemas operativos Windows que soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J# y permite el desarrollo de extensiones para incorporar nuevos lenguajes.

Los orígenes de este IDE se remontan al año 1998 en su versión 6, pero no es hasta el año 2002 con la integración de la plataforma .NET que alcanza el verdadero auge a nivel internacional. En la actualidad permite a los desarrolladores crear aplicaciones de escritorio (Windows Forms), aplicaciones web y servicios web en cualquier entorno que soporte la plataforma .NET dando posibilidad a crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

A partir del año 2005 Visual Studio se distribuye para plataformas Intel, AMD en x86 y x64 en varias versiones: Express, Standard Professional, Tools for Office y Visual Studio Team System dando posibilidad a los equipos de desarrollo elegir la más cómoda.

En cualquier edición de Visual Studio pueden encontrarse los siguientes componentes (Microsoft, 2005):

- Un editor de código para la escritura del código fuente, con resaltado de sintaxis y completamiento.
- Herramientas para la ejecución, compilación y depuración del sistema.
- El cuadro de herramientas y el diseñador de formularios y componentes, para desarrollar rápidamente las interfaces de usuario.
- El explorador de soluciones, para ver y administrar archivos de proyecto y las configuraciones.
- Una vista de clases para desplazarse por el código fuente según los tipos de datos y dependencias.
- La ventana propiedades para ajustar las propiedades y eventos en los controles de la interfaz de usuario.

El completamiento de código ofrecido por el editor, la ayuda en contexto y la integración de la ayuda MSDN acelera el proceso de desarrollo de los sistemas. Además Visual Studio puede configurarse cubriendo 4 roles principales en cualquier proceso de desarrollo: Arquitecto de Software, Desarrollador, Probador y Diseñador de la base de dato.

2.7 Herramientas para la creación y compilación de proyectos

Las herramientas de compilación de proyectos como automake y autoconf se encuentran disponibles en la mayoría de los proyectos de código abierto de hoy en día. La mayor ventaja en el uso de estas herramientas se debe a que ayudan a la portabilidad de las aplicaciones a nivel de código fuente, abstrayéndose en la medida de lo posible, de las versiones de las herramientas tradicionales disponibles en cada sistema operativo. Este tipo de herramientas empaquetan la aplicación en una estructura estándar para su distribución, la cual cuenta con un archivo de configuración que será el punto de entrada para instalar la aplicación.

2.7.1 CMake

CMake es una herramienta multiplataforma de generación o automatización de código, su nombre procede de la abreviatura para "cross platform make" (make multiplataforma). Más allá del uso de "make" en el nombre, CMake es una suite separada y de más alto nivel que el sistema make común de sistemas Unix y Windows, siendo similar a otras herramientas como las autotools.

Se puede definir CMake como una familia de herramientas diseñada para construir, probar y empaquetar software, así como controlar el proceso de compilación del software usando ficheros de configuración sencillos e independientes de la plataforma (Martin, y otros, 2008).

La necesidad de disponer de un entorno multiplataforma apropiado para la construcción de sistemas y la portabilidad de estos, dio lugar a la creación de CMake, donde se pueden destacar como principales características (Martin, y otros, 2008):

- La existencia de un fichero de configuración escrito en un lenguaje de scripting específico para CMake.
- Análisis automático de dependencias para C, C++, Fortran, y Java.
- Soporte para varias versiones de Microsoft Visual Studio, incluyendo la 6, 7, 7.1, 8.0, 9.0 y 10.0.
- Detección de cambios en ficheros usando timestamps tradicionales.
- Soporte para la construcción multiplataforma: Linux y otros sistemas POSIX, Mac OS X, Windows 95/98/NT/2000/XP, Windows Vista y Windows 7.

El proceso de construcción del proyecto se controla creando uno o más ficheros CMakeLists.txt en cada directorio (incluyendo subdirectorios) los cuales son el punto clave del funcionamiento de CMake, donde cada CMakeLists.txt consiste en uno o más comandos de la forma COMANDO (argumentos...).

En la actualidad se cuenta con generadores makefile para Unix, Borland make, Watcom make, MinGW, MSYS y Microsoft NMake. También es posible generar ficheros de proyecto para Microsoft Visual Studio de la versión 6 a la 10 incluyendo la arquitectura de 64 bits.

2.8 Metodologías para el desarrollo del software

En la actualidad la construcción de una aplicación informática eficiente que cumpla con los requerimientos planteados es una tarea intensa y difícil de cumplir. Las metodologías para el desarrollo del software imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente.

Una metodología tiene como principal objetivo aumentar la calidad del software que se produce en todas y cada una de sus fases de desarrollo. Hoy en día existen numerosas propuestas metodológicas

que inciden en distintas dimensiones del proceso de desarrollo y seleccionar la adecuada que posibilite obtener los resultados óptimos es uno de los principales problemas a que se enfrentan los equipos de desarrollo.

2.8.1 El Proceso Unificado de Desarrollo de Software (RUP)

El Proceso Unificado de Desarrollo (conocido como RUP por sus siglas en inglés) es un proceso de desarrollo de software que junto al Lenguaje Unificado de Modelado (UML) constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

RUP mejora la productividad del equipo de trabajo y entrega las mejores prácticas del software a todos los miembros del mismo. Es, además, una metodología de desarrollo de software que intenta integrar todos los aspectos a tener en cuenta durante el ciclo de vida del software, con el objetivo de abarcar tanto pequeños como grandes proyectos. (KRUCHTEN, 2001)

RUP posee tres características esenciales: (Jacobson, y otros, 2000)

- **Dirigido por casos de uso**: Los casos de usos representan el hilo conductor que orienta las actividades de desarrollo. Se centra en la funcionalidad que el sistema debe poseer para satisfacer las necesidades del usuario.
- **Centrado en la arquitectura**: Establecer una arquitectura candidata al inicio es un paso muy importante, pues será ella la que guíe el desarrollo del sistema. Esto permitirá el desarrollo en paralelo, minimizar la repetición de trabajos e incrementar la probabilidad de reutilización de componentes.
- **Iterativo e Incremental**: El desarrollo iterativo brinda la posibilidad de que los elementos sean integrados progresivamente, facilita el rehúso y resulta un producto más robusto pues los errores se van corrigiendo en cada iteración.

RUP se repite a lo largo de una serie de ciclos de desarrollo que constituyen la vida de un sistema, donde cada ciclo concluye con una versión del producto. Cada ciclo consta de cuatro fases: Inicio, Elaboración, Construcción y Transición. Todas las fases se subdividen en N iteraciones y terminan con un hito.

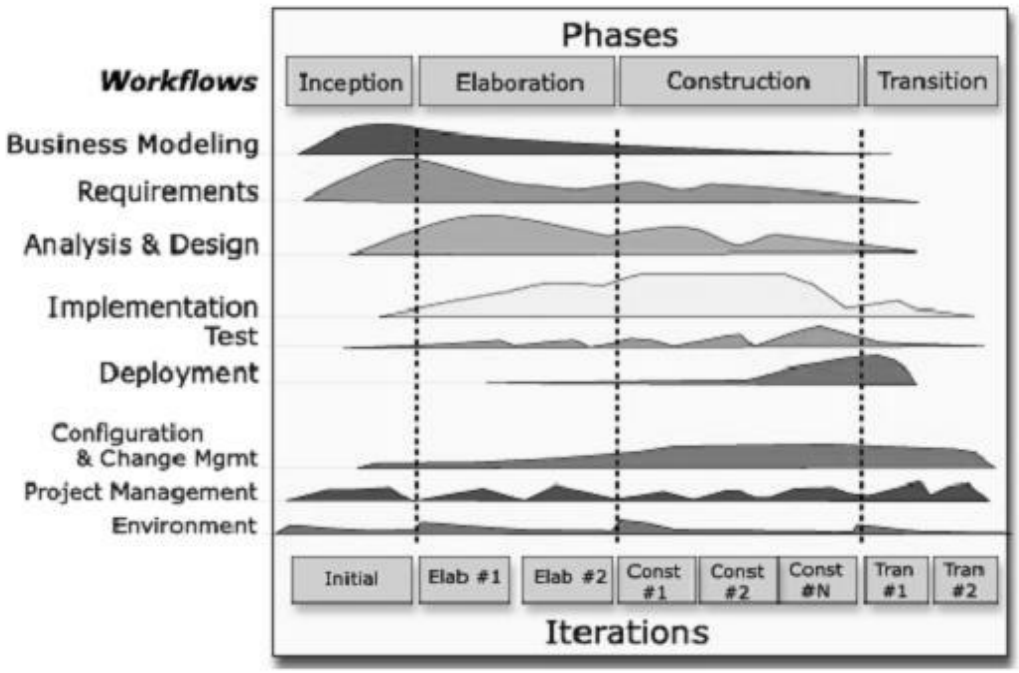


Imagen 2 Fases y flujos de trabajo de RUP (Jacobson, y otros, 2000)

Las nociones de casos de uso y de escenarios utilizadas en RUP han demostrado ser una manera excelente de capturar los requerimientos funcionales y asegurarse que direccionan el diseño, la implementación y la prueba del sistema, logrando así que el sistema satisfaga las necesidades del usuario. Una característica a destacar de RUP es que describe cómo controlar, rastrear y monitorear los cambios, en ambientes en los cuales el cambio es inevitable, para permitir un desarrollo iterativo exitoso.

2.9 El Lenguaje Unificado de Modelado (UML)

La falta de estandarización en la forma de representar gráficamente un modelo impedía que los diseños gráficos realizados se pudieran compartir fácilmente entre distintos diseñadores. Se necesitaba por tanto un lenguaje no sólo para comunicar las ideas a otros desarrolladores sino también para servir de apoyo en los procesos de análisis de un problema.

UML (del inglés Unified Modeling Language) se define como un lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software. Es un sistema de notaciones (que,

entre otras cosas, incluye el significado de sus notaciones) destinado a los sistemas de modelado que utilizan conceptos orientados a objetos (Larman, 1999).

UML ayuda a los usuarios a entender la realidad desde un punto de vista de la tecnología y la posibilidad de que reflexione antes de invertir y gastar grandes cantidades de dinero en proyectos que no estén seguros en su desarrollo, reduciendo el costo y el tiempo empleado en la construcción de los módulos que construirán el software. (Jacobson, y otros, 2000)

Entre los rasgos principales que han contribuido a hacer de UML el estándar de la industria en la actualidad, se puede mencionar (Jacobson, y otros, 2000):

- Permite modelar sistemas utilizando técnicas orientadas a objetos.
- Adecuado a las necesidades de conectividades actuales y futuras.
- Es un lenguaje muy expresivo que cubre las vistas necesarias para desarrollar y luego desplegar los sistemas.
- Ampliamente utilizado por la industria del software.
- Reemplaza a decenas de notaciones empleadas por otros lenguajes y modela estructuras complejas.
- Comportamiento del sistema: casos de usos, diagramas de secuencia, de colaboración, que sirve para evaluar el estado de las máquinas.

2.10 Herramienta Case Visual Paradigm

Las herramientas CASE (Ingeniería de Software Asistida por Ordenador, del inglés Computer Aided Software Engineering) son aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Entre sus principales objetivos se encuentra: (Zapata, y otros, 2005)

- Mejorar la productividad en el proceso de desarrollo del software.
- Aumentar el nivel de calidad del sistema construido.
- Reducir el tiempo y coste de desarrollo.
- Automatizar el desarrollo del software, la documentación, la generación de código, las pruebas de errores y la gestión del proyecto.

- Gestionar de forma global todas las fases de desarrollo del software con una misma herramienta.
- Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

Visual Paradigm es una herramienta CASE que utiliza UML como lenguaje de modelado. Está diseñada para una amplia gama de usuarios interesados en construir sistemas fiables con el uso del paradigma orientado a objetos, incluyendo actividades como ingeniería de software, análisis de sistemas y análisis de negocios (Jacobson, y otros, 2000).

Emplea las últimas notaciones del Lenguaje Unificado de Modelado, ingeniería inversa, generación del código, exportación e importación de archivos XML, entre otros. Es una tecnología libre, multiplataforma, fácil de instalar y actualizar que se encuentra disponible en varios idiomas. De forma general, esta herramienta ofrece:

2.10 Conclusiones parciales

Concluido el capítulo se ha decidido desarrollar la solución propuesta en el lenguaje de programación C# debido a las características mencionadas anteriormente, además cuenta con un conjunto de biblioteca de clases para la manipulación de archivos y carpetas, el código generado es gestionado y cuenta con una alternativa libre para los sistemas GNU/Linux. La codificación de la solución se realizará utilizando el IDE Visual Studio versión Express 2005.

El desarrollo de la solución se realizará empleando únicamente las clases de la plataforma .NET en su versión 2.0 con el objetivo de garantizar la portabilidad a la plataforma Mono en las distribuciones GNU/Linux, de forma tal que el resultado obtenido sea multiplataforma. El proceso de desarrollo será guiado por la metodología RUP y documentado a través de diagramas realizados en Visual Paradigm versión 6.4 con UML 2.1. De forma general las tecnologías seleccionadas responden a las necesidades y exigencias del problema identificado y se caracterizan por ser actuales y contar con respaldo internacional.

CAPÍTULO 3: ANÁLISIS Y DISEÑO DEL SISTEMA

En el presente capítulo se describe la solución propuesta, se realiza un estudio del dominio donde se ejecuta la investigación y se identifican sus principales conceptos y relaciones. Se presentan los requisitos funcionales y no funcionales, el modelo de casos de uso del sistema y su descripción textual. Finalmente se describen las especificaciones del diseño y las clases persistentes en el tiempo.

3.1. Modelo de negocio

El modelo de negocio describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización. Tiene como objetivo comprender la estructura y la dinámica de la organización en la cual se va a implantar un sistema, comprende los problemas actuales de la organización e identifica las mejoras potenciales que se pueden realizar.

Durante el desarrollo de la investigación no se logra determinar las fronteras bien establecidas de los procesos de negocios posibles a automatizar, las personas que lo inician para beneficiarse y las que desarrollan las actividades en cada uno de estos procesos; por tal motivo la investigación se centra más en el desarrollo de un Modelo del dominio.

3.1.1 Modelo del dominio

El modelo del domino ayuda en la compresión de los conceptos empleados por el usuario y con los que deberá trabajar la aplicación. Es una representación visual estática del entorno real objeto del proyecto. Se le nombra dominio para hacer una distinción del modelo del negocio. Este modelo se centra en una parte del negocio, la relacionada con el ámbito del proyecto.

El modelo del dominio se debe concebir como un diccionario visual de abstracciones que será utilizado en fases posteriores y cuya función principal es ayudar a comprender el problema a tratar. Para la presente investigación se decide realizar un modelo de dominio donde se representen los conceptos principales relacionados con el trabajo de archivos CMake, la terminología y nombre de objetos empleados está en correspondencia a los elementos que usualmente presentan los proyectos de CMake y que dominan los desarrolladores de forma general.

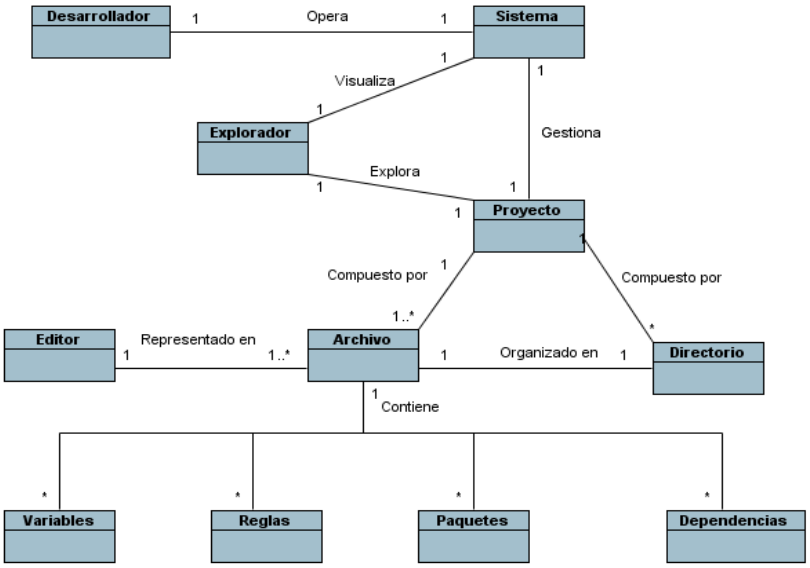


Imagen 3 Representación de las clases del Dominio

3.1.2 Glosario de Términos del Dominio

A continuación se ofrece una breve descripción de cada concepto que interviene en el modelo de dominio presentado con anterioridad.

Desarrollador: Se refiere a la persona que utiliza las prestaciones del sistema. Es el usuario final encargado de gestionar los proyectos con la aplicación.

Sistema: Es la solución como un todo único y funcional que integra las interfaces visuales y las prestaciones del sistema al interactuar con el Desarrollador.

Proyecto: Se refiere a la estructura jerárquica de varios archivos CMakeLists.txt, directorios y archivos de compilación organizados bajo un nombre especificado por el usuario. El proyecto es el objetivo principal del Sistema y representa una aplicación real que requiere ser compilado con CMake.

Explorador: Se refiere a la parte del sistema capaz de organizar y visualizar estructuralmente un proyecto.

Archivo: Es la unidad básica de un proyecto. Puede ser un archivo CMakeLists.txt donde se indican reglas para la compilación, o un archivo de compilación (.cpp, .h, etc.).

Directorio: Es la unidad encargada de estructurar y organizar los archivos del proyecto.

Editor: Se refiere a la parte del sistema que representa el contenido de un archivo CMakeLists.txt y muestra su estructura interna para que pueda ser modificado.

Variables: Contenedor de valores que se pueden definir en un archivo CMakeLists.txt.

Reglas: Se refiere a determinados comandos que se ejecutarán al compilar el proyecto. Generalmente un archivo CMakeLists.txt está compuesto por varias reglas.

Paquetes: Un paquete se refiere a la recopilación de reglas, variables y otros elementos de carácter general que pueden ser utilizados en el proyecto.

Dependencias: Son declaraciones de reglas que se necesitan cumplir al pie de la letra para poder compilar de forma exitosa el proyecto.

3.2 Requisitos funcionales

El término requisito funcional, de acuerdo a RUP, puede definirse como una condición que el sistema debe cumplir o capacidad que debe tener. Mientras más grande e intrincado sea el sistema en desarrollo, más tipos de requisitos aparecen cuando se inicia la recolección de estos. Mediante la identificación de los tipos de requisitos, los equipos de desarrollo de software pueden separar grandes cantidades de requisitos en grupos que faciliten su manejo, también se logra una comunicación más clara entre los miembros del equipo, y en general se mejora el manejo del proyecto en su totalidad.

3.2.1 Estrategia de captura de requisitos

Para la captura de requisitos se propone utilizar como método de análisis la lluvia de ideas. Esta estrategia se basa en expresar ideas acerca del problema y su posible solución donde ningún criterio deberá ser criticado. Además se empleará el análisis de discurso como una alternativa complementaria del método anteriormente planteado; para ello se establecerán un conjunto de conversaciones en tiempo real, donde las preguntas y respuestas deben hacerse de manera rápida incluyendo los puntos de vistas sociales. Las personas que intervienen durante la captura de requisitos deberán ser desarrolladores.

A continuación se detallan los requisitos funcionales que se identificaron para una primera versión de la propuesta a construir:

RF1. Crear nuevo proyecto: El sistema debe permitir crear un proyecto nuevo con el objetivo de gestionar un proyecto que será compilado con CMake. El usuario debe especificar la ruta donde se guardará el proyecto, el nombre del proyecto y la versión mínima de CMake que requiere. Opcionalmente se puede especificar el tipo de proyecto.

RF2. Abrir proyecto: El sistema debe permitir abrir un proyecto guardado con anterioridad.

RF3. Abrir proyecto reciente: El sistema debe mostrar una lista de los proyectos con los cuales se ha trabajado recientemente. Una vez seleccionado un proyecto de la lista, se procederá a cargar el proyecto.

RF4. Guardar modificaciones: El sistema debe permitir guardar los cambios que se realicen en el proyecto. Los cambios se guardarán en la ruta donde se ha creado el proyecto. Al guardar el proyecto por primera vez, el sistema debe actualizar la lista de proyectos recientes de forma tal que el proyecto guardado encabece dicha lista.

RF5. Explorar la estructura del proyecto: El sistema debe permitir visualizar de forma jerárquica la estructura de un proyecto cargado. La estructura estará compuesta principalmente por los directorios del proyecto y los archivos CMakeLists.txt contenidos en los directorios.

RF6. Cerrar proyecto: El sistema debe permitir cerrar el proyecto actual. Si el proyecto ha sufrido modificaciones se debe alertar al usuario y preguntarle si desea guardar los cambios.

RF7. Editar archivo CMakeLists.txt del proyecto: El sistema debe permitir modificar la estructura de un archivo CMakeLists.txt que forma parte del proyecto.

RF8. Eliminar archivo CMakeLists.txt: El sistema debe permitir eliminar un archivo CMakeLists.txt de la estructura del proyecto.

RF9. Crear directorio: El sistema debe permitir crear un directorio en la estructura del proyecto. Opcionalmente se creará un archivo CMakeLists.txt en el directorio creado para gestionar su contenido.

RF10. Eliminar directorio: El sistema debe permitir eliminar un directorio de la estructura del proyecto.

RF11. Definir variable: El sistema debe permitir declarar una variable en el archivo CMakeLists.txt que se edita. El nombre y valor de la variable debe ser especificado por el usuario.

RF12. Editar variable: El sistema debe permitir cambiar el valor de una variable definida en el archivo CMakeLists.txt que se edita.

RF13. Eliminar variable: El sistema debe permitir eliminar una variable definida en el archivo CMakeLists.txt que se edita.

RF14. Añadir archivos para su compilación: El sistema debe permitir añadir un conjunto de archivos de código fuente (.cpp, .cxx, .c, .cc) y de cabecera (.h) para que sean compilados en un archivo CMakeLists.txt como resultado del proyecto.

RF15. Especificar tipo de proyecto a compilar: El sistema debe permitir al usuario especificar qué tipo de proyecto se está construyendo, ya sea una biblioteca estática, biblioteca compartida o código ejecutable. Adicionalmente se debe permitir especificar la versión mínima de CMake requerida para el proyecto.

RF16. Dependencias de paquetes: El sistema debe permitir incluir en el proyecto determinadas restricciones de dependencias de paquetes ya definidos en CMake.

RF17. Eliminar dependencia de paquete: El sistema debe permitir eliminar una dependencia de paquete anteriormente definida.

RF18. Imprimir mensaje: El sistema debe permitir insertar mensajes personalizados por el usuario con el fin de depurar el flujo de compilación. Los mensajes pueden ser informativos o de error.

RF19. Añadir comando: El sistema debe permitir añadir un comando o función CMake, así como especificar los parámetros correspondientes.

RF20. Eliminar comando: El sistema debe permitir eliminar un comando definido con anterioridad.

RF21. Definir función de usuario: El sistema debe permitir al usuario declarar una función o comando personalizado que no existe en el conjunto de comandos definidos. De esta forma es posible modificar el flujo de compilación del archivo CMake.

RF22. Eliminar función de usuario: El sistema debe permitir eliminar una función de usuario definida con anterioridad.

RF23. Definir condición: El sistema debe permitir regular el flujo de compilación del proyecto haciendo uso de condiciones.

RF24. Eliminar condición: El sistema debe permitir eliminar una condición definida anteriormente.

RF25. Crear proyecto a partir de un proyecto en Qt: El sistema debe permitir al usuario crear un nuevo proyecto a partir de la definición de un proyecto en Qt existente.

RF26. Crear proyecto a partir de un proyecto en CMake: El sistema debe permitir al usuario crear un nuevo proyecto a partir de un proyecto en CMake existente.

RF27. Convertir proyecto a un proyecto de Qt: El sistema debe permitir exportar las declaraciones y estructura del proyecto actual a la definición de un proyecto de Qt. Como resultado final se obtiene un archivo (.pro) que representa un proyecto de QtCreator.

3.3 Requisitos no funcionales

Los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener. Normalmente están vinculados a requisitos funcionales, es decir una vez que se conozca lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser (Pressman, 2002).

3.3.1 Requisitos no funcionales de usabilidad

1. El sistema estará destinado para desarrolladores de proyectos con C++ y compilación en CMake, con conocimientos mínimos de CMake.
2. El sistema debe poseer una interfaz entendible, amigable, con semejanza a un Entorno de Desarrollo Integrado (IDE) o algún editor conocido, de fácil acceso, manipulación y que facilite la localización de las diferentes funcionalidades presentes en ella.
3. El sistema debe guiar de forma entendible al usuario para alcanzar determinado objetivo.

3.3.2 Requisitos no funcionales de apariencia o interfaz externa

1. El sistema debe tener una apariencia profesional con un diseño gráfico sencillo de tonalidades de colores claros que facilite la localización de las funciones del sistema.
2. La terminología empleada en el sistema debe ser común a la empleada en un IDE y de conocimiento para los desarrolladores.

3.3.3 Requisitos no funcionales de portabilidad y operatividad

1. La aplicación debe ser compatible con los sistemas operativos de Microsoft Windows a partir de su versión 2000/NT y en plataformas GNU/Linux.

3.3.4 Requisitos no funcionales de software del sistema

1. Las computadoras que utilizarán el software deben tener instalado:
    a. La plataforma Mono para las distribuciones de GNU/Linux.
    b. La plataforma Microsoft .NET Framework en su versión 2.0 o superior para el sistema operativo Microsoft Windows.

3.3.5 Requisitos no funcionales de hardware del sistema

1. Las computadoras que utilizarán la aplicación deberán contar con un microprocesador con velocidad de procesamiento superior a un 1 GHz[4] y memoria RAM de 512 MB[5] o superior.

3.4 Descripción del sistema propuesto

Un caso de uso constituye una técnica utilizada para describir el comportamiento del sistema, a través de un documento narrativo que define la secuencia de acciones que obtienen resultados de valor para un actor que utiliza un sistema para completar un proceso, sin importar los detalles de la implementación.

Los actores se definen como los roles que puede tener un usuario; pueden ser humanos, otros sistemas, máquinas, hardware, etc. que interactúan con un sistema para de esta forma intercambiar datos.

3.4.1 Descripción de los actores del sistema

Actor	Descripción
Desarrollador	Se refiere al principal beneficiario de trabajar con el sistema. Representa a la persona que requiere crear y configurar un proyecto para ser compilado utilizando CMake y ganar en portabilidad. Es el responsable de interactuar con el sistema y explotar las prestaciones del mismo. Participa en la inicialización de los casos de usos del sistema.

Tabla 1Descripción de los actores del sistema

3.4.2 Diagrama de casos de uso del sistema

A continuación se muestra el diagrama de casos de uso del sistema donde se representa el actor del sistema encargado de iniciar los casos de usos; posteriormente se describe textualmente el caso de uso del sistema Abrir proyecto y se muestran los prototipos de interfaz de usuario correspondiente. Las descripciones de los restantes casos de usos del sistema pueden ser consultados en el Anexo I.

[4]Gigahercio: es un múltiplo de la unidad de medida de frecuencia hercio (Hz) y equivale a 109 Hz.
[5]Memoria de Acceso Aleatorio.

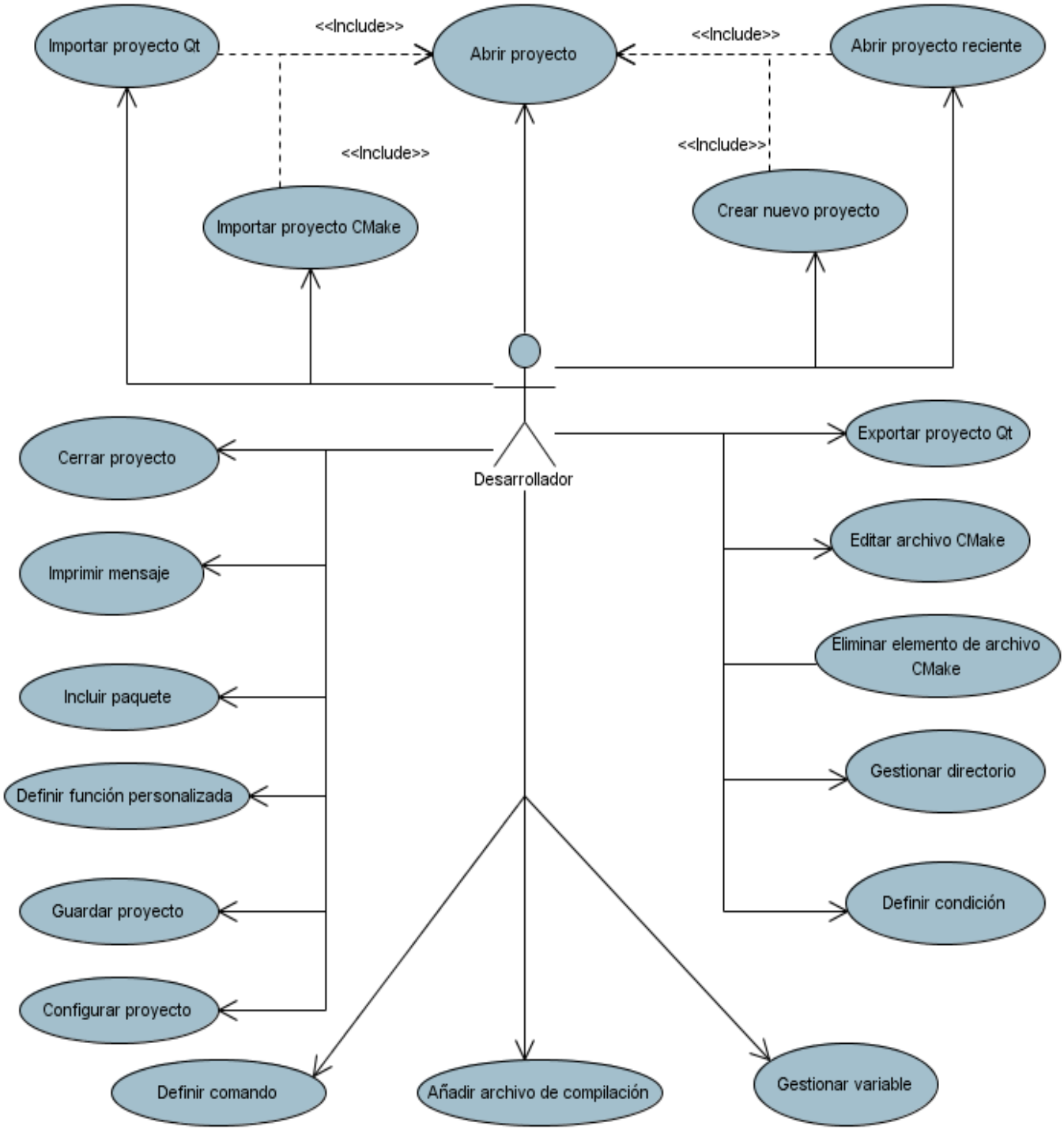


Imagen4 Diagrama de casos de uso del sistema

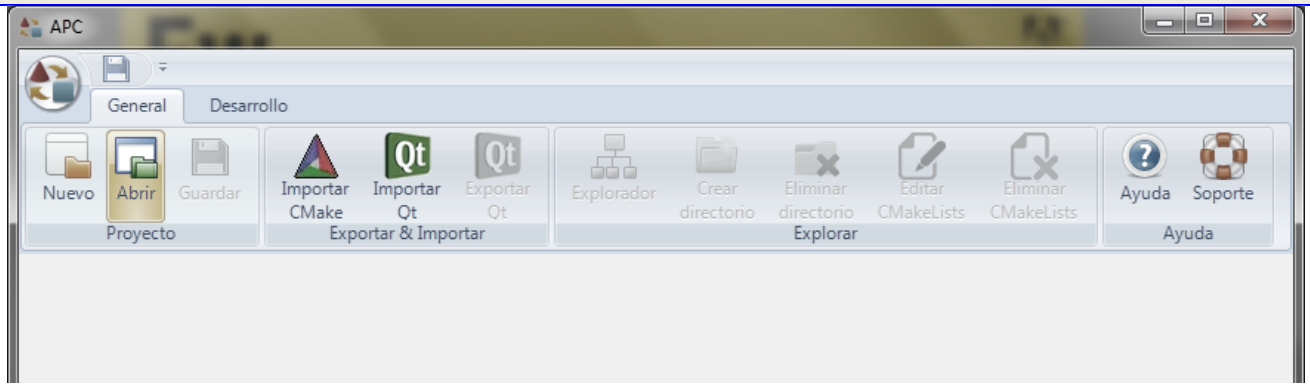
3.4.3 Descripción textual de los casos de usos del sistema

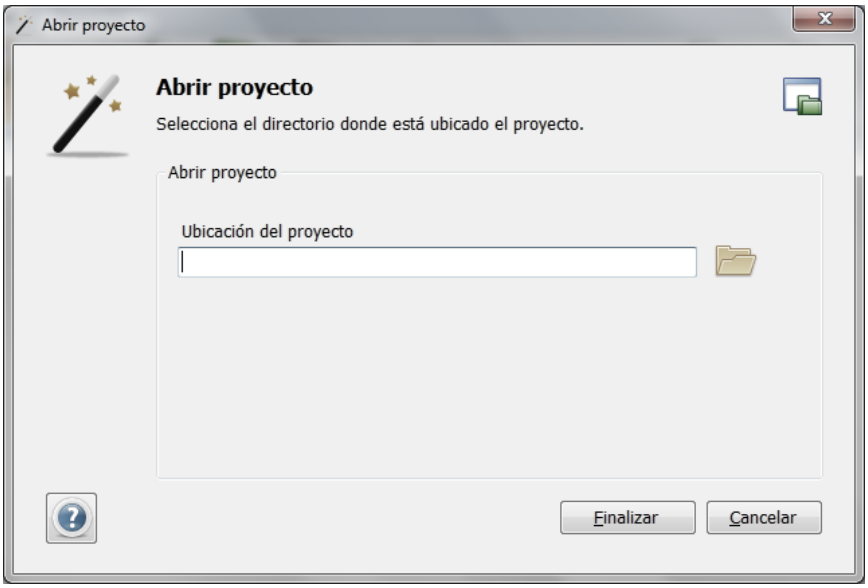
Caso de Uso	**Abrir proyecto**
Actores	Desarrollador
Resumen	El caso de uso se inicia cuando el desarrollador desea abrir un proyecto existente. El sistema debe permitir buscar la ubicación del proyecto y cargar la información asociada al mismo. El caso de uso finaliza cuando el sistema carga la información del proyecto y muestra su estructura.
Precondiciones	No tiene.
Referencias	RF2
Prioridad	Crítico

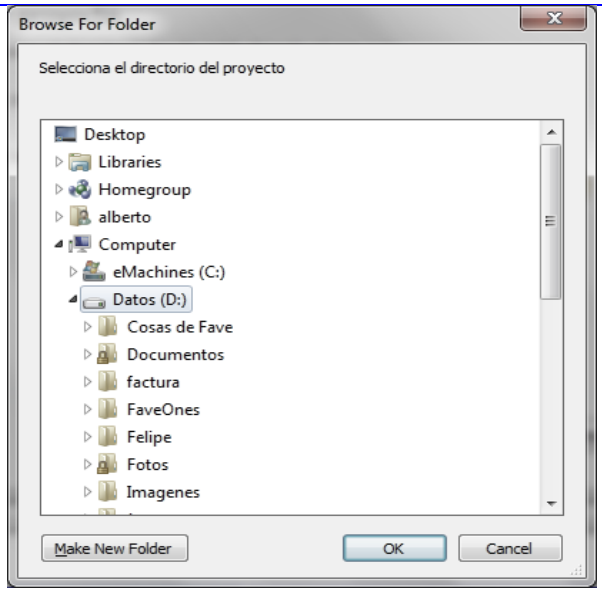
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1-El caso de uso se inicia cuando el desarrollador selecciona la opción “Abrir proyecto” en el panel “Proyecto” (*prototipo 1*).	2- El sistema muestra una ventana en forma de asistente que guía al desarrollador para seleccionar el proyecto que desea abrir (*prototipo 2*).
3-El desarrollador hace clic en la opción de explorar para seleccionar el directorio donde se encuentra el proyecto que desea abrir.	4-El sistema muestra una ventana en forma de explorador que contiene la lista de carpetas navegables para buscar el directorio donde se encuentra el proyecto (*prototipo 3*).
5-El desarrollador selecciona el directorio del proyecto y hace clic en “Aceptar”	6-El sistema cierra la ventana de explorador de proyecto y muestra la ruta al directorio seleccionado.
7-El desarrollador hace clic en el botón “Finalizar” para abrir el proyecto seleccionado con anterioridad.	8-El sistema carga el proyecto mostrando su estructura en el “Explorador de proyecto” y habilitando las opciones de edición. El caso de uso finaliza (*prototipo 4*).

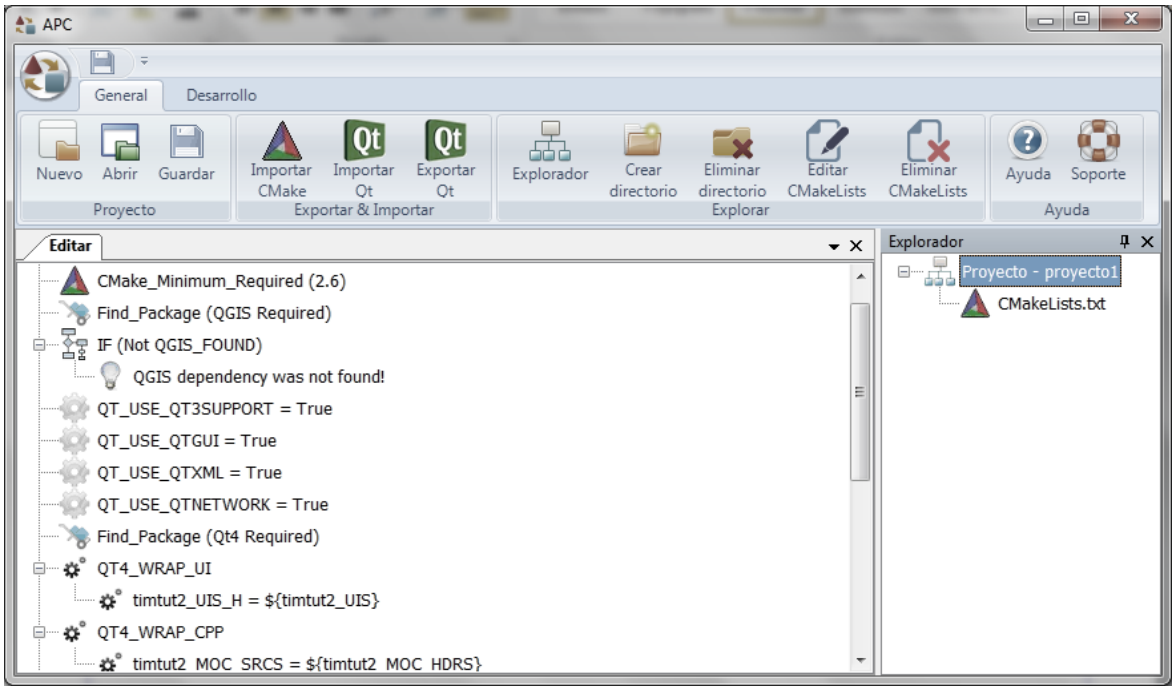
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
5.1-El desarrollador no encuentra el proyecto	5.2-El sistema cierra la ventana.

buscado y hace clic en el botón “Cancelar”.	
7.1-El desarrollador hace clic en el botón “Cancelar”.	7.2-El sistema cierra el asistente para abrir proyecto y finaliza el caso de uso.
Prototipo de Interfaz	









Poscondiciones	Se abre el proyecto CMake.

Tabla 2 Descripción textual del caso de uso del sistema "Abrir proyecto"

3.5 Principios de diseño

Definir principios de diseño en el desarrollo de una aplicación permite que la misma se convierta en una herramienta útil y atractiva para el usuario. Los principios de diseño no solo definen la apariencia estética de la interfaz, sino que posibilitan también que la misma brinde adecuadamente la información que le sea útil al usuario.

Como principios de diseño del sistema se tienen:

- Brindar una interfaz sencilla ambientada de forma similar a un IDE, de forma tal que las personas con conocimientos mínimos en cuanto a desarrollo de software puedan trabajar fácilmente con la aplicación.
- Garantizar la legibilidad de manera que exista contraste entre los colores de los textos y el fondo y que el tamaño de la fuente sea lo suficientemente adecuado a la vista del usuario.
- Mostrar al usuario, siempre que vaya a realizar una acción relevante sobre el sistema, un mensaje de confirmación que le permita asegurarse que es correcta la opción seleccionada.
- Los mensajes que son mostrados al usuario deben ser precisos y de fácil comprensión.
- Los Menús y las etiquetas de botones deben comenzar con la palabra más importante.

3.5.1 Estándares de la interfaz de la aplicación

Barra de Título

- El título debe ubicarse a la izquierda y los botones minimizar, maximizar y cerrar, ubicados del lado derecho.
- No deben usarse colores degradados u otros que interrumpan la visibilidad de los elementos anteriores.
- La barra de título debe mostrar un resumen de la acción que se ejecuta.

Barra de Menú

- Los menús se ordenan según la importancia de las funcionalidades que brindan.
- El menú Ayuda está compuesto por dos elementos: “Temas de ayuda” y un cuadro de diálogo "Acerca de" o “Soporte”.

- Para una mejor simplicidad de la barra de menú se agrupan las funcionalidades en bloques de acuerdo al ámbito.

3.5.2 Concepción general de la ayuda

La ayuda del sistema está concebida en dos formas distintas:

- Los temas de ayuda: donde el usuario puede consultar los procedimientos que dispone el sistema para alcanzar determinados objetivos.
- Las ayudas de contexto: donde se refiere a la ayuda relacionada con la parte del sistema en donde se trabaja.

3.6 Diagrama de clases del diseño

Para la transición de los requisitos al diseño normalmente se utiliza la fase de análisis; esta fase tiene dos propósitos fundamentales: refinar los casos de uso con más detalle y establecer la asignación inicial de funcionalidad del sistema a un conjunto de objetos que proporcionan el comportamiento. Es una especificación detallada de los requisitos y funciona como primera aproximación del modelo de diseño.

El propósito y objetivo del análisis debe alcanzarse de algún modo en todo proyecto. Pero la manera exacta de ver y de emplear el análisis puede diferir de un proyecto a otro y una de las variantes que se pueden emplear es no utilizar en absoluto el modelo de análisis para describir los resultados del análisis. En cambio el proyecto analiza los requisitos como parte integrada de la captura de requisitos o en el diseño (Jacobson, y otros, 2000).

La investigación actual, decidió prescindir de la realización del Modelo de Análisis en el desarrollo de esta investigación debido a que: se comprende en su totalidad el modelo de casos de uso del sistema, los requisitos son simples, bien conocidos y se cuenta con cierta comprensión de los mismos, además se logra evitar el costo en tiempo y recursos de mantener este flujo.

Los diagramas de clases del diseño son diagramas de estructura estática que muestran las clases del sistema y las relaciones entre ellas; muestran lo que el sistema puede hacer y cómo puede ser

construido (Jacobson, y otros, 2000). Cuando se crea un diagrama de clases, se está modelando una parte de los elementos y las relaciones que configuran la vista de diseño del sistema.

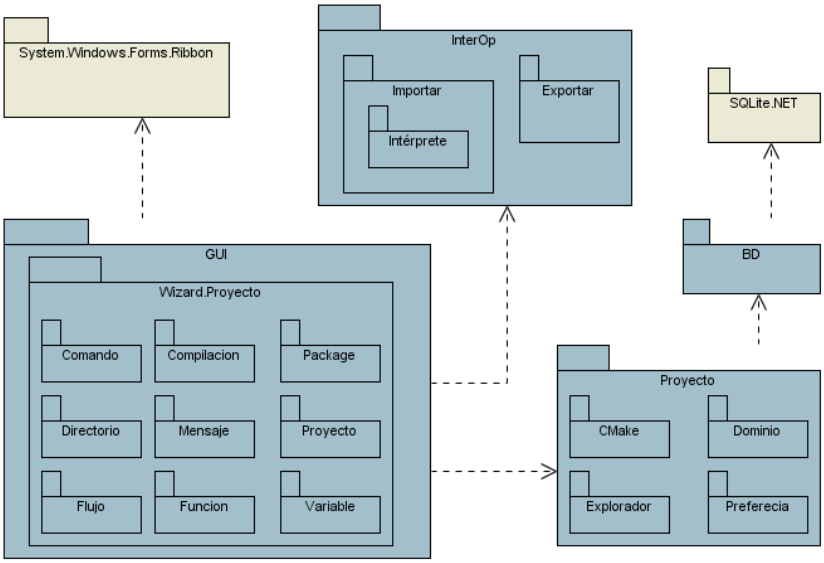


Imagen5 Estructura del sistema organizado en paquetes del diseño

A continuación se muestra el diagrama de clases del diseño para el caso de uso “Abrir proyecto”, los restantes diagramas de clases del diseño pueden consultarse en el Anexo II.

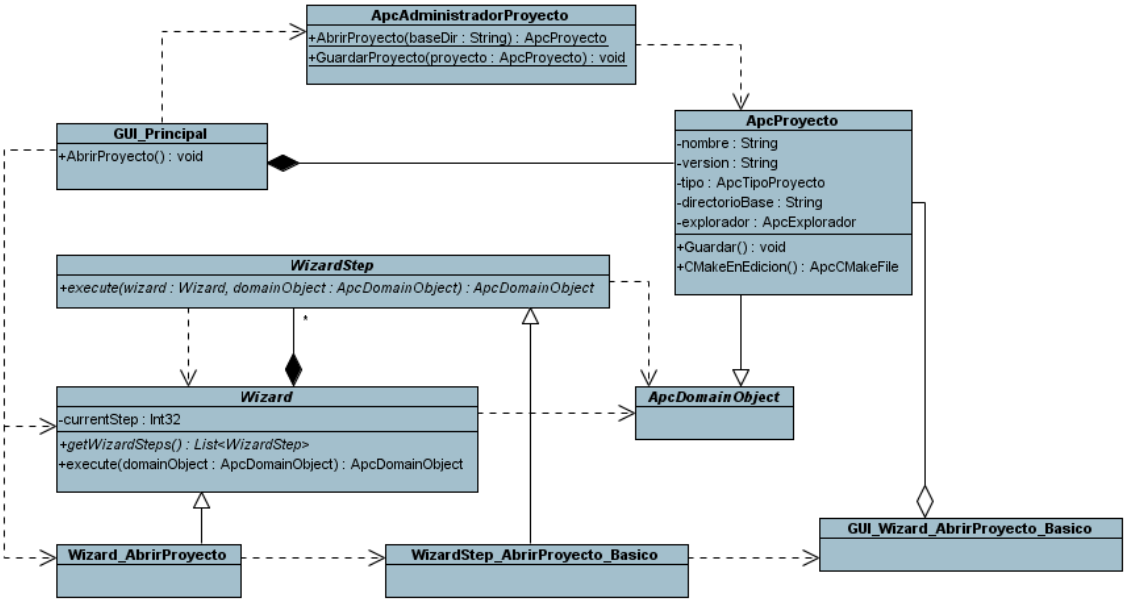


Imagen 6 Diagrama de clases del diseño del caso de uso Abrir proyecto

Para la construcción del sistema propuesto se organizaron sus clases de acuerdo al patrón arquitectónico seleccionado y se agruparon por paquetes del diseño como se muestra en la Imagen 5. En el diagrama mostrado en la Imagen 6 se puede observar los patrones de diseño Cadena de Responsabilidad Fábrica Abstracta, los cuales posibilitan la implementación de los requisitos funcionales identificados y el cumplimiento de las especificaciones de los requisitos no funcionales.

3.7 Patrones de diseño

Un patrón de diseño puede considerarse como el conjunto de reglas que describen cómo afrontar tareas y solucionar problemas que surgen durante el desarrollo de software. Otros autores definen un patrón de diseño como la idea que ha sido útil en la práctica y determinado contexto y que probablemente sea útil a otros (Flower, 2006).

Un patrón de diseño expresa el problema que resuelve, las clases que participan con sus responsabilidades y colaboraciones entre ellas; en ocasiones muestran fragmentos de códigos que especifican la correcta implementación del patrón (Pressman, 2002).

Durante el desarrollo de la investigación se utilizaron distintos patrones de diseño que ayudaron a la mejor estructuración del sistema y solventar problemas comunes durante el desarrollo. A continuación se describen los patrones más relevantes que se utilizaron:

- **Instancia única (Singleton)**: Es un patrón tipo creacional que expresa la necesidad de mantener una y sola una instancia de una clase; este patrón además define un único punto de acceso a la clase. El uso de este patrón en la investigación permite administrar una sola instancia de la clase que contiene la lista de proyectos recientes y también es utilizado para mantener una sola instancia de la clase que administra la información del proyecto que se gestiona.
- **Cadena de responsabilidad (Chain of Responsibility)**: Es un patrón del tipo comportamiento el cual especifica cómo repartir las responsabilidades respecto a una petición u objeto, de forma tal que al recibir un objeto éste pasa a través de un conjunto de “manejadores” que tienen la responsabilidad de atender la solicitud. El primer manejador recibe la petición y puede atenderla o bien reenviarla al próximo candidato el cual puede hacer lo mismo. El uso de este patrón en la investigación permitió la creación de asistentes visuales para guiar al usuario paso

a paso, donde cada paso representa un elemento o manejador de la cadena y la petición constituye el objetivo que se desea alcanzar.

- **Fábrica abstracta (Abstract Factory)**: Es un patrón tipo creacional que provee las interfaces necesarias para crear un conjunto de familias de objetos relacionadas o de objetos dependientes sin necesidad de especificar los tipos concretos. En la presente investigación este patrón permitió junto al patrón Cadena de responsabilidad la creación de asistentes visuales. En este caso se cuenta con dos familias: el asistente que puede ser para crear un proyecto, abrir proyecto, definir variable, etc.; y los pasos que tiene cada asistente para alcanzar el objetivo.

3.8 Clases persistentes

La persistencia de una clase está dada por la capacidad de la misma para mantener su valor en el espacio y en el tiempo; contrario a las clases temporales que son manejadas y almacenadas por el sistema en tiempo de ejecución, dejando de existir al finalizar el programa. Todas las clases identificadas durante el desarrollo del diseño no tienen que ser necesariamente persistentes y las clases persistentes no necesariamente deben persistir en una base de datos. La solución que se propone mantiene un registro de los proyectos que se abrieron para posteriormente acceder a ellos de forma rápida, la información relativa a esta operación es guardada en un archivo de base de datos tipo SQLite. En el siguiente diagrama se muestra la definición de la única tabla que tiene el archivo.

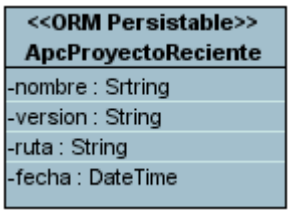


Imagen7 Diagrama de clases persistente en base de datos

El sistema al guardar un proyecto con los cambios efectuados, crea un archivo con extensión .apc que contiene la definición de dicho proyecto; para permitir su posterior edición y poder reconocer los elementos que lo componen. Por tal motivo se identificaron un conjunto de clases cuya definición es guardada en el archivo .apc.

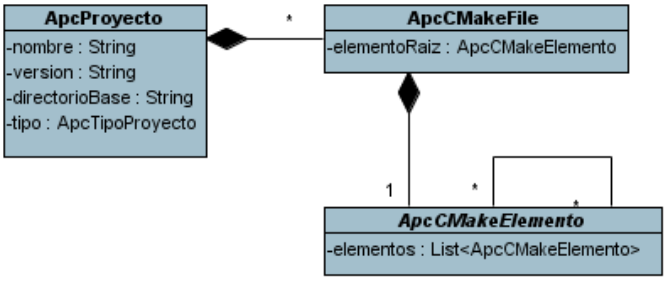


Imagen8 Diagrama de clases persistente en un archivo de proyecto (.apc)

3.9 Conclusiones parciales

En el presente capítulo se describieron detalladamente las funcionalidades que ofrecerá el sistema con motivo a dar respuesta al problema identificado. Se especificaron las necesidades en cuando a software y hardware para lograr una interfaz amigable y un producto usable y robusto.

Posteriormente se agruparon los requisitos funcionales para dar paso al diagrama de casos de uso del sistema y la identificación de los posibles actores del mismo. Se presentó una descripción textual de los casos de uso, así como los prototipos de interfaces. Se presentó la estructura del sistema en cuanto a organización de las clases y paquetes en un diagrama de clases del diseño y se describieron los patrones más relevantes que definieron las especificaciones del diseño. Se presentó además la persistencia de las clases, siendo de dos tipos distintos, base de datos y archivos.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

En el presente capítulo se realiza un análisis de la estructura del sistema en términos de componentes y se muestran las especificaciones de la implementación. Se presenta además un diseño del modelo de despliegue y finalmente se arriba a los casos de pruebas necesarios para asegurar el correcto funcionamiento del sistema.

4.1 Generalidades de la implementación

En la fase de construcción se implementa el sistema, es decir, se crea el código correspondiente al resultado de la fase de diseño, siguiendo los patrones y la arquitectura escogida. El modelo de implementación permite planificar las integraciones de sistemas necesarias en cada iteración, distribuye el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue, implementa las clases y subsistemas encontrados durante el diseño y posibilita probar los componentes individualmente para después integrarlos.

4.1.1 Modelo de componentes

Los componentes son la parte modular del sistema, desplegable y reemplazable que encapsula implementación y un conjunto de interfaces con su realización. Un componente típicamente contiene clases y puede ser implementado por uno o más artefactos, éstos pueden ser un fichero de código fuente, scripts, ficheros de código binario, ejecutables, etc. Los diagramas de componentes son utilizados para modelar la vista estática del sistema, mostrando la organización y las dependencias lógicas entre los componentes.

Durante el desarrollo de la investigación se arribaron a un conjunto de componentes relacionados principalmente con la implementación de la lógica del sistema. Producto de la misma estructura que ofrece la tecnología .NET y los estándares de implementación que sugiere el lenguaje C#, cada clase del diseño fue implementada en archivos separados, manteniendo de este modo una estructura y organización homogénea.

En la siguiente imagen se muestran las relaciones de dependencias entre los componentes físicos del sistema. Los componentes SQLite.dll, SQLite3.dll y SQLite.NET.dll son componentes correspondientes

a un subsistema importado que gestionan el trabajo con base de datos de SQLite. Los componentes WeifenLuo.WinFormsUI.Docking.dll y System.Windows.Forms.Ribbon.dll gestionan la interfaz de usuario.

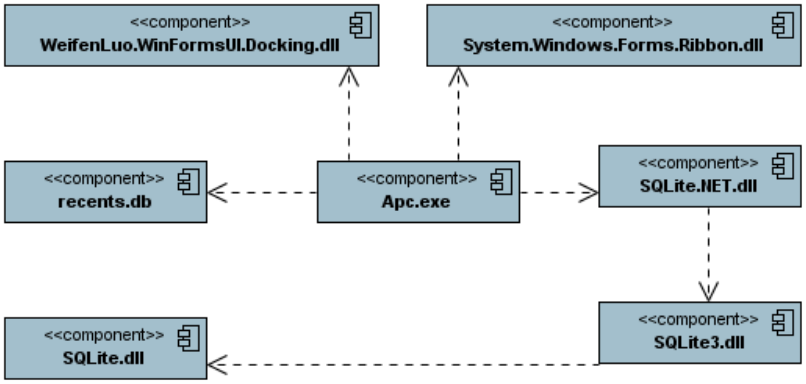


Imagen 9 Modelo de componentes físicos

4.1.2 Modelo de despliegue

El Modelo de Despliegue describe la distribución física del sistema en términos de cómo las funcionalidades se distribuyen entre los nodos de computación sobre los que se va a instalar el sistema; mostrando las relaciones entre el hardware y el software en el sistema final. Se representa como un grafo de nodos unidos por conexiones de comunicación (Jacobson, y otros, 2000).

En la siguiente figura se ilustra el modelo de despliegue para el sistema planteado.

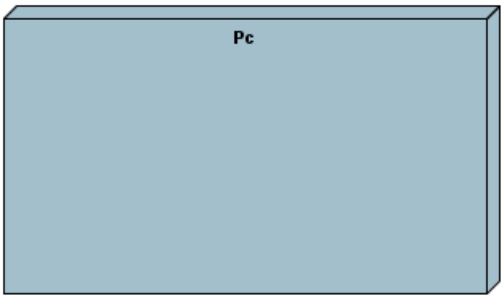


Imagen 10 Modelo de despliegue

Para el despliegue de la solución se cuenta únicamente con un nodo de procesamiento el cual tendrá los componentes físicos del sistema. Las especificaciones en cuanto a hardware y software del nodo Pc se describen en los requisitos no funcionales.

4.2 Pruebas del sistema

El desarrollo del software implica una serie de actividades en las que la posibilidad de que aparezcan errores humanos es muy común. Los errores pueden comenzar a manifestarse desde el primer momento del proceso en el que los objetivos pueden estar especificados de forma errónea e imperfecta, y así en los posteriores pasos del diseño y desarrollo (Pressman, 2002).

La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación, estas tienen diferentes clasificaciones, algunas de ellas pueden ser:

- **Pruebas de verificación**: Se comprueba el cumplimiento de las especificaciones del diseño.
- **Pruebas de validación**: Se encargan de velar por el cumplimiento de los requisitos del análisis.
- **Pruebas de caja blanca**: Se conoce el código y se trata de ejecutar cada uno de los elementos del mismo.
- **Pruebas de caja negra**: Solamente se conoce la interfaz y se trata de probar cada uno de los elementos que componen a la misma.

Para que las pruebas aplicadas a un software tengan éxito es necesario realizar casos de pruebas con probabilidad de descubrir los errores en el sistema, a través de técnicas que guíen el proceso de la prueba.

4.3 Pruebas de caja negra

Para probar el sistema propuesto se utiliza la prueba de caja negra, la cual se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, por lo que los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que la salida producida es correcta, ya que no es necesario conocer la lógica del programa, únicamente la funcionalidad que debe realizar. Muchos autores consideran que estas pruebas permiten encontrar (Pressman, 2002):

- Funciones incorrectas o ausentes.
- Errores de interfaz.

- Errores en estructuras de datos o en accesos a las bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Para preparar los casos de pruebas hacen falta un número de datos que ayuden a la ejecución de los casos y que permitan que el sistema se ejecute en todas sus variantes, pueden ser datos válidos o inválidos para el sistema según si lo que se desea es encontrar un error o probar una funcionalidad. Los datos se escogen atendiendo a las especificaciones del problema, sin importar los detalles internos del sistema, a fin de verificar que el mismo funcione correctamente. Para desarrollar la prueba de caja negra existen varias técnicas, entre ellas están (Pressman, 2002):

- **Técnica de la partición de equivalencia**: esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del sistema.
- **Técnica del análisis de valores límites**: esta técnica prueba la habilidad del sistema para manejar datos que se encuentran en los límites aceptables.
- **Técnica de grafos de causa-efecto**: es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

Dentro del método de caja negra la técnica de la partición de equivalencia es una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas existentes en el sistema, descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico.

La partición de equivalencia se dirige a la definición de casos de pruebas que descubran clases de errores, reduciendo así en número de clases de prueba que hay que desarrollar. El diseño de casos de prueba para la partición de equivalencia se basa en una evaluación de las clases de equivalencia para una condición de entrada. Una clase de equivalencia representa un conjunto de estados válidos o inválidos para condiciones de entrada. Regularmente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica (Pressman, 2002).

Las clases de equivalencia se pueden definir de acuerdo con las siguientes directrices:

- Si un parámetro de entrada debe estar comprendido en un cierto rango, aparecen tres clases de equivalencia: por debajo, en y por encima del rango.
- Si una entrada requiere un valor concreto, aparecen tres clases de equivalencia: por debajo, en y por encima del rango.
- Si una entrada requiere un valor presente en un conjunto de valores, aparecen dos clases de equivalencia: en el conjunto o fuera de él.
- Si una entrada es booleana (lógica), hay dos clases: si o no.

Aplicando estas directrices se ejecutan casos de pruebas para cada elemento de datos del campo de entrada a desarrollar, donde los casos de prueba se seleccionan de forma que ejerciten el mayor número de atributos de cada clase de equivalencia a la vez.

A continuación se muestra el diseño de caso de prueba para el caso de uso del sistema “Abrir proyecto” utilizando la técnica de partición de equivalencia. Los restantes casos de pruebas pueden ser consultados en el Anexo III.

4.3.1 Caso de uso “Abrir proyecto”

Descripción general

El caso de uso se inicia cuando el desarrollador desea abrir un proyecto. El sistema debe permitir buscar la ubicación del proyecto y cargar la información asociada al proyecto. El caso de uso finaliza cuando el sistema carga la información del proyecto y muestra su estructura.

Condiciones de ejecución

No tiene.

Secciones a probar en el caso de prueba

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo central
SC 1: Abrir proyecto	EC 1.1 Abrir proyecto	El caso de uso se inicia cuando el desarrollador selecciona la opción	❖ Panel Proyecto.

CMake	exitosamente.	"Abrir proyecto" en el panel "Proyecto", el sistema muestra una ventana en forma de asistente que guía al desarrollador para seleccionar el proyecto que desea abrir, el desarrollador hace clic en la opción de explorar para seleccionar el directorio donde se encuentra el proyecto que desea abrir, el sistema muestra una ventana en forma de explorador que contiene la lista de carpetas navegables para buscar el directorio donde se encuentra el proyecto, el desarrollador selecciona el directorio del proyecto y hace clic en "Aceptar", el sistema cierra la ventana de explorador de proyecto y muestra la ruta al directorio seleccionado, el desarrollador hace clic en el botón "Finalizar" para abrir el proyecto seleccionado con anterioridad, el sistema carga el proyecto mostrando su estructura en el "Explorador de proyecto" y habilitando las opciones de edición.	❖ Abrir.
	EC 1.2 Cancelar la apertura del proyecto.	El caso de uso se inicia cuando el desarrollador selecciona la opción "Abrir proyecto" en el panel "Proyecto", el sistema muestra una ventana en forma de asistente que guía al desarrollador para seleccionar el proyecto que desea abrir, el	❖ Panel Proyecto. ❖ Abrir. ❖ Cancelar.

		desarrollador hace clic en la opción de explorar para seleccionar el directorio donde se encuentra el proyecto que desea abrir, el sistema muestra una ventana en forma de explorador que contiene la lista de carpetas navegables para buscar el directorio donde se encuentra el proyecto, el desarrollador selecciona el directorio del proyecto y hace clic en “Aceptar”, el sistema cierra la ventana de explorador de proyecto y muestra la ruta al directorio seleccionado, el desarrollador hace clic en el botón “Cancelar”.	
	EC 1.3 No se encuentra el proyecto.	El caso de uso se inicia cuando el desarrollador selecciona la opción “Abrir proyecto” en el panel “Proyecto”, el sistema muestra una ventana en forma de asistente que guía al desarrollador para seleccionar el proyecto que desea abrir, el desarrollador hace clic en la opción de explorar para seleccionar el directorio donde se encuentra el proyecto que desea abrir, el sistema muestra una ventana en forma de explorador que contiene la lista de carpetas navegables para buscar el directorio donde se encuentra el proyecto, el proyecto no se encuentra en ese directorio, el desarrollador hace clic en el botón	❖ Panel Proyecto. ❖ Abrir. Cancelar.

		“Cancelar”.	

Tabla 3 Secciones a probar en el caso de uso “Abrir proyecto”

Descripción de variable

No	Nombre de campo	Clasificación	Valor nulo	Descripción
1	Ubicación del proyecto.	Selección	No	Permite seleccionar la ubicación del proyecto.

Tabla 4 Descripción de las variables para el caso de uso Abrir proyecto

Matriz de datos

SC 1: “Abrir proyecto”

ID del escenario	Escenario	Ubicación del proyecto	Respuesta del sistema
1.1	Abrir proyecto Cmake exitosamente.	V	Se abre el proyecto con éxito.
1.2	Cancelar la apertura del proyecto Cmake.	NA	Se cancela la operación.

Tabla 5 Matriz de Datos. SC1. Caso de uso Abrir proyecto

Resultados de las pruebas

Como se puede observar en la siguiente tabla, se realizaron pruebas al sistema en las cuales se encontraron varios errores durante las dos primeras iteraciones, estos fueron erradicados y se realizaron dos iteraciones más en las que no se encontró ningún error y se pudo comprobar que el software está libre de errores y listo para ser usado por el cliente.

Numero de iteraciones	Errores
1ra	8 errores
2da	2 errores
3ra	0 errores
4ta	0 errores

Tabla 6 Resultados de las pruebas

4.4 Conclusiones parciales

En el capítulo que concluye se presentó la especificación del sistema en términos de componentes y sus relaciones de dependencia. Se presentó el modelo de despliegue ilustrando el único nodo de procesamiento que requiere el sistema y finalmente se concluyó con los diseños de casos de pruebas para validar la implementación de los requisitos funcionales capturados en las primeras etapas.

CONCLUSIONES

Finalizada la investigación y luego de obtenidos los resultados, es posible resaltar una serie de conclusiones que se enumeran a continuación:

1. La puesta en explotación de la solución alcanzada garantizará la reducción de tiempo para la creación y configuración de proyectos realizados con CMake.
2. La selección de la plataforma .NET permitió centrarse en el desarrollo de las especificaciones del sistema, brindando una amplia biblioteca de clases con las funciones más comunes durante el desarrollo de un software.
3. El uso de clases únicamente hasta la versión 2.0 del framework .NET permitió crear una solución multiplataforma, compatible con la plataforma Mono para entornos GNU/Linux.
4. El sistema obtenido tiene doble valor debido a que soluciona un problema real en el proyecto SIG-Desktop, es aplicable a otros proyectos y constituye la primera solución de su tipo en la Universidad de las Ciencias Informáticas.
5. La correcta implementación de la arquitectura seleccionada (tres capas) posibilitó el rápido desarrollo del sistema y la creación de los mecanismos necesarios para incrementar el sistema en funcionamiento sin afectar las prestaciones actuales.
6. El sistema cuenta con una ayuda general y contextual que posibilita mantener informado al usuario en todo momento, haciendo del sistema un producto robusto y fácil de utilizar.
7. La metodología de desarrollo seleccionada (RUP) posibilitó documentar ampliamente el sistema y obtener artefactos claves para su entendimiento y el posterior desarrollo de nuevas funcionalidades.
8. La implementación de las pruebas arrojaron un conjunto de errores que fueron corregidos y contribuyeron a obtener un mejor producto.

RECOMENDACIONES

Finalizada la investigación, el autor de la misma recomienda lo siguiente:

- Continuar la identificación de requisitos funcionales en aras de ampliar las posibilidades de trabajo con la herramienta desarrollada.
- Desarrollar funcionalidades con el objetivo de dar tratamiento diferenciado a elementos del lenguaje CMake que se usan con mayor frecuencia.
- Incorporar nuevas funcionalidades de Importación/Exportación que permitan la interoperabilidad con otros sistemas de compilación y configuración como autoconf, automake, etc.
- Diseñar e implementar nuevas pruebas para identificar y documentar los defectos en función de lograr un producto más robusto y libre de errores.
- Utilizar el sistema en la gestión de proyectos en otros departamentos de la Universidad con vista a identificar errores y nuevas prestaciones que garanticen una mejor solución.
- Traducir las interfaces de la aplicación, así como la ayuda y el manual de usuario a otros idiomas para incrementar la posibilidad de uso de la herramienta.

TRABAJOS CITADOS

Acevedo Martínez, Liesner Ramírez y Osorio, Karel. 2011. *Técnicas de Compilación: Manual Práctico para estudiantes de Informática.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2011.

Alegsa. 2012. Diccionario de Informática. [En línea] 2012. http://www.alegsa.com.ar/Dic/multiplataforma.php.

Álvares Rojas, Segio y Mora Mata, Miguel Ángel. 2005. *Traductores y Compiladores con Lex/Yacc, JFlex/Cup y JavaCC.* s.l. : Universidad de Málaga, 2005. ISBN 84-689-1037-6.

CMake-Project. 2012. CMake Editor. [En línea] 2012. [Citado el: 15 de Abril de 2012.] http://www.cthing.com/CMakeEd.asp.

Cornejo, J.E.G. 2001. *Arquitectura en Capas. Un camino hacia los procesos distribuidos.* 2001.

Flower, Martin. 2006. *Analysis Pattern - Reusable Object Models.* s.l. : Prentice-Hall, 2006.

Fowler, Martin, y otros. 2002. *Patterns of Enterprise Application Architecture.* s.l. : Addison Wesley, 2002. ISBN 0-321-12742-0.

Garrido, Roberto. 2012. Configurando proyectos multiplataforma fácilmente con CMake. [En línea] 12 de 5 de 2012. http://www.genbetadev.com/herramientas/configurando-proyectos-multiplataforma-facilmente-con-cmake.

Herández León, Rolando Alfredo y Coello González, Sayda . 2011. *El proceso de Investigación Científica.* Ciudad de la Habana : Editorial Universitaria del Ministerio de Educación Superior, 2011. ISBN 978-959-16-1307-3.

Jacobson, I y Rumbauhg, J. 2000. *El Lenguaje Unificado de Modelado. Manual de referencia.* Madrid : Pearson Educación, 2000.

Jacobson, I, Booch, G y Rumbaugh, J. 2000. *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison-Wesley, 2000. ISBN 84-7829-036-2.

KRUCHTEN, P. 2001. *The Rational Unified Process: An Introduction (2nd Edition).* Boston : s.n., 2001.

Larman, Craig. 1999. *UML y Patrones Introducción al análisis y diseño orientado a objetos.* México : s.n., 1999.

Mamone, Mark. 2006. *Practical Mono.* Berkeley,California : Apress, 2006. ISBN 1-59059-548-3.

Martin, Ken y Hoffman, Bill. 2008. *Mastering CMake.* s.l. : Kitware, Inc, 2008. ISBN 1-930934-16-5.

Menendez Romero, Alberto. 2010. *Herramienta para la captura de datos de campo del Inventario de Petróleo y Gas perteneciente al Sistema de Gestión de Datos Geológicos.* Ciudad de La Habana : Universidad de las Ciencias Informáticas, 2010.

Microsoft. 2011. Características del lenguaje C#. [En línea] 2011. [Citado el: 4 de Noviembre de 2011.] http://msdn.microsoft.com/es-es/library/aa287997%28v=vs.71%29.aspx.

Micorosft. 2005. Lo nuevo en Visual Studio 2005. [En línea] 2005. [Citado el: 21 de Noviembre de 2011.] http://msdn.microsoft.com/es-es/library/88fx1xy0%28v=vs.80%29.aspx.

Molina, Daniel. 2009. s.l. : Universidad de Cadiz. Departamento de Lenguaje y Sistemas Informáticos, OSLUCA, 2009.

Nourie, Dana. 2005. Getting Started with an Integrated Development Environment (IDE). [En línea] 24 de Marzo de 2005. [Citado el: 2 de Diciembre de 2011.] http://java.sun.com/developer/technicalArticles/tools/intro.html.

Pressman, Roger S. 2002. *Ingeniería del software. Un enfoque práctico. Quinta edición.* Madrid : Mc.Graw Hill / Interamericana de España, 2002.

RAE. 2008. Diccionario de la Lengua Española – Vigésima Segunda Edición. [En línea] 2008. http://rae.es/lenguaje_programacion.

RAE. 2001. Multiplataforma. [En línea] 2001. http://rae.es/multiplataforma.

Rammer, Ingo y Szpuszta, Mario. 2005. *Advanced .NET Remoting, Second Edition.* Berkeley, California : apress, 2005. ISBN 1-59059-417-7.

Schulte, R. 1995. *Three-Tier Computing Architectures and Beyond.* s.l. : Gartner Group, 1995.

STAIR, Ralph M. 2003. *Principles of Information Systems, Sixth Edition.* s.l. : Thomson Learning, 2003. ISBN 0-619-06489-7.

UNE-ISO-15489-1. 2005. *Información y Documentación. Gestión de documentos.* Madrid : s.n., 2005.

vissemee. 2012. A VisualStudio CMake Editing Extension. [En línea] 2012. http://code.google.com/p/vissemee/.

Zapata, Luis Giraldo y Yuliana. 2005. Herramientas de desarrollo de ingeniería de SW para Linux. [En línea] 24 de Septiembre de 2005. [Citado el: 5 de Noviembre de 2011.] http://hugolopez.phi.com.co/docs/download/file=Giraldo-Zapata-Herramientas%20de%20ISW.pdf,_id=17.

BIBLIOGRAFÍA

Acevedo Martínez, Liesner Ramírez y Osorio, Karel. 2011. *Técnicas de Compilación: Manual Práctico para estudiantes de Informática.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2011.

Albin, Stephen. 2003. *The Art of Software Architecture - Design Methods and Techniques.* Indianapolis, Indiana : Wiley Publishing, Inc, 2003. ISBN 0-471-22886-9.

Ambler, Scott W. 2002. *The Elements of UML Style.* Cambridge, United Kingdom : Cambridge University Press, 2002. ISBN 0-521-52547-0.

Barker, F. Scott. 2005. *Visual C# 2005 Express Edition Starter Kit.* Indianapolis, Indiana : Wiley Publishing, Inc, 2005. ISBN-13: 978-0-7645-8955-3.

Bartlett, Roscoe A. , y otros. 2008. *Trilinos CMake Evaluation.* Albuquerque, New Mexico : Sandia National Laboratories, 2008.

Bolognese, Luca, Pizzo, Mike y Short, Keith. 2002. *Designing Data Tier Components and Passing Data Through Tiers.* s.l. : Microsoft Corporation, 2002.

Brena, Ramón. 2003. *AUTOMATAS Y LENGUAJES.* Monterrey : Tec de Monterrey, 2003.

Codeproject. 2002. Object Serialization using C#. [En línea] 31 de Enero de 2002. [Citado el: 20 de 03 de 2012.] http://www.codeproject.com/Articles/1789/Object-Serialization-using-C.

Cooper, James W. 2002. *Introduction to Design Patterns in C#.* s.l. : Addison Wesley, 2002.

De la Torre, César, y otros. 2010. *Guía de arquitectura N-Capas orientada al Dominio con .NET.* España : Krasis Press, 2010. ISBN: 978-84-936696-3-8.

Engels, Jan. 2007. *CMake Tutorial.* 2007. Kitware, Inc..

Fowler, Martin, y otros. 2002. *Patterns of Enterprise Application Architecture.* s.l. : Addison Wesley, 2002. ISBN 0-321-12742-0.

Hamilton, Kim y Miles, Russell. 2006. *Learning UML 2.0 .* s.l. : O' Reilly, 2006. ISBN 0-596-00982-8.

Jacobson, I y Rumbauhg, J. 2000. *El Lenguaje Unificado de Modelado. Manual de referencia.* Madrid : Pearson Educación, 2000.

Jacobson, I, Booch, G y Rumbaugh, J. 2000. *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison-Wesley, 2000. ISBN 84-7829-036-2.

KRUCHTEN, P. 2001. *The Rational Unified Process: An Introduction (2nd Edition).* Boston : s.n., 2001.

Larman, Craig. 1999. *UML y Patrones Introducción al análisis y diseño orientado a objetos.* México : s.n., 1999.

MacKenzie, David, Elliston, Ben y Demaille, Akim. 2008. *Autoconf. Creating Automatic Configuration Scripts.* s.l. : Free Software Fundation, 2008.

MacKenzie, David, Tromey, Tom y Duret-Lutz, Alexandre. 2008. *GNU Automake.* s.l. : Free Software Foundation, Inc, 2008.

Mamone, Mark. 2006. *Practical Mono.* Berkeley,California : Apress, 2006. ISBN 1-59059-548-3.

Martin, Ken y Hoffman, Bill. 2008. *Mastering CMake.* s.l. : Kitware, Inc, 2008. ISBN 1-930934-16-5.

Moreno, Ana M. y Sánchez Segura, Maribel. 2007. *Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el momento de Arquitectónico.* España : Universidad Carlos III de Madrid, 2007.

Övergaard, Gunnar y Palmkvist, Karin. 2004. *Use Cases Patterns and Blueprints.* s.l. : Addison Wesley Professional, 2004. ISBN 0-13-145134-0.

Pressman, Roger S. 2002. *Ingeniería del software. Un enfoque práctico. Quinta edición.* Madrid : Mc.Graw Hill / Interamericana de España, 2002.

Rammer, Ingo y Szpuszta, Mario. 2005. *Advanced .NET Remoting, Second Edition.* Berkeley, California : apress, 2005. ISBN 1-59059-417-7.

Randy Davis, Stephen. 2009. *C++ fro Dummies. 5th Edition.* Indianapolis, Indiana : Wiley Publishing, Inc, 2009. ISBN: 0-7645-6852-3.