UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 6

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

FRAMEWORK PARA EL DESARROLLO BASADO EN COMPONENTES SOBRE QT

Autores:

Yunet Gasca Suárez

Adrian Martínez Borges

Tutor:

Ing. Reynier Pupo Gómez

La Habana, junio 2013

"Año 55 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos que _____ y _____ somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____del 2013.

_____ _____

Firma del Autor
Yunet Gasca Suárez

Firma del Autor
Adrian Martínez Borges

_____

Firma del Tutor
Ing. Reynier Pupo Gómez

DATOS DE CONTACTO

Tutor: Ing. Reynier Pupo Gómez (rgomez@uci.cu)

Graduado de Ingeniero en Ciencias Informáticas en la UCI en el año 2010. Trabajador del centro GEySED en el proyecto Video Vigilancia en el cual se desempeña como Líder de Proyecto.

Autor: Yunet Gasca Suárez (ygasca@estudiantes.uci.cu)

Estudiante perteneciente al departamento de Señales Digitales del centro GEySED, específicamente al proyecto Video Vigilancia. Desempeña el rol de Analista.

Autor: Adrian Martínez Borges (amborges@estudiantes.uci.cu)

Estudiante perteneciente al departamento de Señales Digitales del centro GEySED, específicamente al proyecto Video Vigilancia. Desempeña el rol de Programador.

AGRADECIMIENTOS

Adrian:

Agradecer a mi madre, por su preocupación y dedicación a pesar de la distancia, por haber contribuido a mi formación no solo como profesional sino también como persona.

Agradecer a mi padre, por ser mi consejero, por tener confianza en mí aún en los momentos más difíciles, si hoy soy la persona que soy es por ti, por tus enseñanzas, porque cada día a tu lado fue como estar en una escuela, en este caso la escuela de la vida, gracias papá, te debo mucho y en ocasiones las palabras no pueden expresar lo que hoy quiero decir, solo espero que estés orgulloso de mí.

A mi hermano, manito gracias por haberte quedado en la escuela trabajando por mí, por haberme cuidado, por hacerme entender la programación, por tu apoyo incondicional sin importar si tenía la razón o no, siempre estuviste a mi lado para lo que me hiciera falta, gracias mano.

A mi compañera de tesis Yunet, que más que una compañera ha sido una amiga a lo largo de estos cinco años y estoy en de deuda con ella. Gracias por estar a mi lado en las buenas y en las malas, gracias por ser mi amiga incondicional, por siempre decirme la verdad y no lo que quisiese oír, la verdad es que eres especial para mí y quiero que quede así para siempre.

A mis amistades por haber sido pilares claves en el transcurso de estos cinco años, no dejaré plasmado en este fragmento sus nombres ya que sería injusto si alguno no fuese mencionado, por lo que gracias a todos por haberme soportado, por aguantar mis malos humores, por en ocasiones ser más que amigos.

Para la realización de esta tesis no puedo dejar de mencionar a las personas que me apoyaron en la la implementación, a mi amigo Aramis por haberme dado el primer empuje, a Rayner y Reynier Pupo, ya que aparte de ser mis tutores, fueron mis amigos.

Yunet:

Madre a ti te agradezco en primer lugar este logro, por haberme dado la vida y el lugar que me corresponde. Por ser mi tesoro más preciado y estar siempre para mí en las buenas y en las malas. Por no darme el placer de la duda creyendo que no podía llegar hasta aquí. Gracias por existir y hacerme ver la vida de un modo donde tenga total protagonismo.

A mi abuela Nereida por ser más que mi abuela mi madre, por quererme y hacerme sentir importante, por apoyarme a pesar de sus prejuicios, por perdonarme a pesar de mis errores. Por desenmascarar en ocasiones el alma de niña que aun mí se esconde.

A mi abuelo Mario por confiar siempre en mí y considerarme la estrella que más brilla en el cielo. Por no escatimar ni pestañar ante mis necesidades.

A mi hermana Yuneisy por quererme y pelearme aun cuando no lo merezco. Por ser incondicional y estar ahí cuando más la necesito. Por saber ser cariñosa incluso después de sus enfados. Por saber mostrar las virtudes que en ocasiones esconde.

A mi padrasto Tony que ha sabido ocupar el lugar de aquel padre que de vez en vez obra por la vida como fantasma. Por considerarme su hija y velar por mi salud y bienestar aún más allá de sus posibilidades y prioridades.

A mis tíos: Freddy y Osvaldo que son mis tesoros más preciados, tal vez por eso quiero aún más a mi abuela, por dar a luz a dos hombres que son incondicionales, que obran por la vida sin interés de recibir nada cambio, por ser familiares y velar por el bienestar de todos.

DEDICATORIA

A nuestros padres, hermanos, abuelos y a todos aquellos amigos que de una forma u otra han contribuido a que hoy hayamos alcanzado la meta trazada.

A nuestro Comandante en Jefe Fidel Castro Ruz, por habernos guiado hacia un futuro de hombres de ciencia, por haber forjado una Revolución donde todos tienen acceso a la educación, desde el más pobre hasta el que vive en el lugar más intrincado del país.

A todos aquellos profesores que con su trabajo arduo han aportado su granito de arena para nuestra formación como profesional y educación, porque como decía José de la Luz y Caballero:

"Enseñar puede cualquiera, educar solo un evangelio vivo"

RESUMEN

Con el avance de las Tecnologías de la Información y las Comunicaciones (TICs) y el incremento de las exigencias por parte de los clientes en cuanto al desarrollo de aplicaciones complejas en períodos cortos de tiempo, es que el desarrollo de software basado en componentes se ha convertido en uno de los paradigmas de programación más efectivos e imprescindibles. Es por ello que el departamento de Señales Digitales de la Universidad de las Ciencias Informáticas (UCI) tiene como prioridad potenciar la producción y reutilización en el desarrollo de los proyectos que se le han asignado.

La presente investigación tuvo como propósito desarrollar un framework basado en componentes que permitiera la integración y comunicación de componentes. Dicho sistema facilita el desarrollo de aplicaciones a través de componentes prefabricados, garantizando que el desarrollo de los mismos se rija por una estructura común; lo cual posibilita una mayor organización y entendimiento por parte de los programadores. Permite aprovechar las funciones de los componentes para crear sistemas de software de gran alcance.

El documento refleja todo el proceso de desarrollo del software, para el que fue necesario realizar un estudio de algunos sistemas desarrollados mediante la reutilización de componentes. Se recogen las principales características de las herramientas utilizadas para la modelación e implementación de la solución. Se tratan los elementos fundamentales asociados a la metodología empleada y los principales artefactos generados por los flujos de trabajo que propone, además de la correcta implementación de la solución.

PALABRAS CLAVE: componentes, framework, paradigmas.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La programación constituye una de las actividades fundamentales en el desarrollo de software y está vinculada a un paradigma de programación; reconocido por los métodos y herramientas que selecciona el programador durante el desarrollo de cualquier sistema. Un paradigma de programación surge con la necesidad de desarrollar aplicaciones cada vez más complejas y constituye una alternativa a las desventajas de anteriores paradigmas.

En un principio la programación estructurada hacía fácil la escritura y verificación de un programa, pero cuando este estaba compuesto por grandes listas de instrucciones se hacía problemático su manejo. Para resolver este problema los programas se descomponían en unidades más pequeñas que adoptan el nombre de funciones. Cada una de ellas tenía un propósito bien definido y resolvían una tarea concreta, diseñándose una interfaz claramente definida para su comunicación con otras funciones. Con el paso de los años la idea de particionar un programa en funciones fue evolucionando y se llegó a su agrupamiento en unidades más grandes llamadas módulos; sin embargo el principio seguía siendo el mismo: agrupar componentes que ejecutaban listas de instrucciones.

La Programación Orientada a Objetos (POO) surge para dar solución a diversas limitaciones que se encontraban en anteriores enfoques de programación. En lugar de intentar ajustar un problema al enfoque procedimental de un lenguaje, la POO intenta ajustar el lenguaje al problema. La programación estructurada pretende resolver un problema de principio a fin en una sola estructura de código, mientras que la POO, lo resuelve identificando los actores que tienen participación en el problema y sus acciones. Con esta información se crean los objetos, compuestos por clases donde se detallan las acciones que realizan y las propiedades de estos. La POO intenta modelar la realidad del problema a través de objetos independientes que interactúen entre sí, creando un rumbo diferente en el proceso de desarrollo de software mediante la reutilización a través de la producción de componentes genéricos, fáciles de integrar, distribuidos e independientes de las plataformas de desarrollo. Este nuevo enfoque da respuesta a la creciente necesidad de desarrollar sistemas complejos en periodos cortos de tiempo, con los menores esfuerzos humanos y económicos posibles, siendo el punto de partida para el desarrollo de software basado en componentes.

El paradigma de programación basado en componentes se apoya en la idea de utilizar componentes de software ya desarrollados que se combinen adecuadamente para satisfacer los requisitos del sistema. La coordinación e interacción entre componentes exige un modelo de componentes que establezca la

infraestructura de software requerida o framework y las convenciones y restricciones de diseño de los mismos. Esto implica la adaptación o desarrollo de componentes con el propósito expreso de ser reutilizados en futuras aplicaciones, lo que significa que la aplicación apropiada de la reutilización en un proyecto de software conduce indiscutiblemente a una reducción significativa de los valores de las variables más importantes de la ingeniería de software: costo, tiempo y esfuerzo.

El centro de desarrollo Geoinformática y Señales Digitales (GEySED) de la Facultad 6 perteneciente a la Universidad de las Ciencias Informáticas (UCI) está compuesto por dos departamentos dedicados a la producción de software. El departamento de Señales Digitales cuenta con diversos proyectos productivos en los que se evidencia la deficiente interoperabilidad entre componentes; debido a que los mismos se encuentran aislados entre sí. No existe una comunicación entre ellos que permita que intercambien información para aprovechar las funciones de los mismos en aras de crear sistemas de software de gran alcance. Esto trae consigo que componentes desarrollados en determinados proyectos para ser utilizados en otros deban ser migrados o el código de los mismos sea copiado y adaptado. Existen casos donde los proyectos se ven afectados a la hora de realizar cambios a un software que ya ha sido liberado, esto se debe a las acciones de agregar o modificar funcionalidades a petición de los clientes o a necesidades internas de los mismos proyectos.

Teniendo en cuenta la problemática antes descrita, se define el siguiente **problema a resolver:** ¿Cómo integrar componentes provenientes de diferentes proyectos del departamento de Señales Digitales?

Con el fin de dar solución al problema propuesto se define como **objeto de estudio:** el desarrollo de software basado en componentes, delimitando como **campo de acción:** el desarrollo de software basado en componentes para el departamento de Señales Digitales.

El **objetivo general** de esta investigación es desarrollar un framework basado en componentes que permita la integración y comunicación de componentes para el departamento de Señales Digitales.

Como **idea a defender** se plantea: el desarrollo del framework basado en componentes permitirá la integración y comunicación de componentes desarrollados por diferentes proyectos del departamento de Señales Digitales.

Con el propósito de lograr un desarrollo exitoso se han trazado las siguientes tareas:

- Caracterización de los procesos necesarios para la gestión de componentes.

- Análisis de soluciones existentes similares a la que se desea desarrollar tanto a nivel nacional como internacional.
- Fundamentación de las tecnologías y herramientas a utilizar para el desarrollo de la solución.
- Análisis del framework de desarrollo para la gestión de componentes.
- Diseño del framework de desarrollo para la gestión de componentes.
- Implementación del framework de desarrollo para la gestión de componentes.
- Validación de la propuesta de solución.

Para definir el objeto de estudio, analizar el estado del arte y garantizar el correcto desarrollo de la investigación se utilizaron los siguientes métodos científicos:

Métodos Teóricos:

- Histórico – Lógico: se utilizó para analizar la evolución histórica de soluciones existentes similares al sistema a desarrollar en cuanto a su basamento sobre el paradigma de programación basado en componentes. Se realizó un estudio de su estado actual en función de que las características que se ajustaran a las necesidades del departamento pudieran ser utilizadas para el desarrollo en curso.
- Analítico – Sintético: se utilizó para realizar un análisis exhaustivo de documentos y bibliografías de diferentes autores para extraer los elementos más importantes relacionados al desarrollo de software basado en componentes.
- Modelación: se utilizó para la compresión de objetos y sus relaciones, al representar jerárquicamente el sistema a desarrollar.

Métodos Empíricos:

- Observación: se utilizó para obtener información acerca de soluciones similares al sistema que se desea desarrollar, lo que permite poseer una visión de cómo sería en su forma externa.
- Entrevista: se utilizó con el propósito de establecer conversaciones con diferentes líderes de proyectos del departamento de Señales Digitales, para obtener información acerca de los proyectos que están desarrollando componentes. Se efectuó una entrevista de tipo estructurada puesto que se elaboró un cuestionario que contiene todas las preguntas, las cuales fueron realizadas en el mismo orden a cada líder interrogado. **(Ver Anexo 1)**

Estructura de la investigación:

Resumen, Introducción, 4 Capítulos, Conclusiones, Recomendaciones, Bibliografía Referenciada, Bibliografía Consultada y Anexos. La presente investigación tiene su mayor peso en la estructura capitular, la cual está organizada de la siguiente manera:

Capítulo # 1_Fundamentación Teórica asociada a al desarrollo de software basado en componentes: en este capítulo se explican los conceptos asociados al dominio del problema y se realiza un estudio del estado del arte referente al desarrollo de software basado en componentes.

Capítulo # 2_Tendencias y tecnologías actuales a desarrollar: en este capítulo se analizan y caracterizan las tecnologías y herramientas a utilizar para el desarrollo de sistemas de software similares al que se pretende desarrollar, lo que posibilita que se realice la selección que se considere adecuada para el desarrollo de solución.

Capítulo # 3_Análisis y Diseño: en este capítulo se muestra una concepción general al problema planteado a través de un modelo conceptual en aras de lograr una mayor comprensión de los procesos involucrados. Se describen los elementos funcionales con los que debe cumplir el sistema a desarrollar mediante las historias de usuario, así como las iteraciones a las que son asignadas. Se desarrollan las tarjetas CRC para delimitar las clases y responsabilidades del sistema a desarrollar y se establece la arquitectura base que debe poseer el framework basado en componentes.

Capítulo # 4_Implementación y pruebas: en este capítulo se aborda todo el proceso de construcción y las pruebas para validar la propuesta de solución.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA ASOCIADA AL DESARROLLO DE SOFTWARE BASADO EN COMPONENTES

Introducción

Como parte del proceso de investigación se hace necesario conocer los aspectos más significativos enmarcados en el objeto de estudio; facilitando la comprensión tanto del equipo de desarrollo como de los usuarios finales. Por tanto en el presente capítulo se realiza un análisis de los principales elementos asociados al desarrollo de software basado en componentes, que permiten definir sus conceptos y terminologías más importantes. Se realiza un estudio de la situación actual del dominio del problema, así como de los procesos necesarios para la gestión de componentes. Seguidamente se lleva a cabo un estudio del estado del arte que sirva como punto de partida para el desarrollo de la solución.

1.1. Conceptos asociados al dominio del problema

Antes de comenzar a definir qué es el desarrollo de software basado en componentes, es necesario conocer determinados conceptos que faciliten la comprensión de la temática. A continuación se abordan los que se consideran más importantes.

Componente

Según Szyperski un componente: “Es un paquete coherente de artefactos de software que puede ser desarrollado independientemente y entregado como una unidad y este puede ser compuesto, intercambiado con otro componente para construir algo mucho más grande” (Szyperski, 2002).

Como resultado a los estudios realizados por Szyperski según otra de sus definiciones plantea que un componente: “Es una unidad de composición de aplicaciones software que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio” (Szyperski, 2002).

Kruchten propone otra definición para componente, la cual plantea que: “Es una parte no trivial, casi independiente y reemplazable de un sistema que llena claramente una funcionalidad dentro de un contexto en una arquitectura bien definida. Un componente se conforma y provee la realización física por medio de un conjunto de interfaces” (Krutchen, 2000).

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA ASOCIADA AL DESARROLLO DE SOFTWARE BASADO EN COMPONENTES

Específicamente en el departamento de Señales Digitales el sistema Suria cuenta con cinco módulos, entre ellos el Visor, el cual permite la visualización y gestión de cámaras IP[1]. Para ello cuenta con componentes visuales que tienen la tarea de mostrar el flujo de video proveniente de las cámaras. Los componentes que integran el módulo fueron estructurados de acuerdo a las preferencias del proyecto puesto que nos es posible su utilización en otros.

Framework

Un framework es un diseño reutilizable de todo o parte de un sistema software descrito por varias jerarquías de herencia de clases, generalmente algunas abstractas y por las colaboraciones que se establecen entre las instancias de estas clases (García, y otros).

Marco de trabajo, del inglés "*frame*" que significa marco y "*work*" que significa trabajo. Son herramientas implementadas tanto con objetivos específicos como generales que reúnen un conjunto de características y funcionalidades para facilitar la implementación de las aplicaciones a las cuales va dirigida. Es un patrón o esqueleto de una aplicación con un conjunto de estándares y organización, el cual permite la construcción de una aplicación organizada.

Modelos de Componentes

Pueden describir cómo interactúan los componentes entre sí y cómo interactúan dichos componentes con un framework de componentes. Los modelos de componentes imponen estándares y convenciones sobre:

- **Tipos de Componentes:** un tipo de componente puede ser definido en términos de las interfaces que implementa. Los tipos diferentes de componentes pueden desempeñar diferentes roles en el sistema y participar en distintos tipos de esquemas de interacción.
- **Esquemas de Interacción:** especifican cómo los componentes son localizados, cuáles protocolos de comunicación son utilizados y cómo se satisfacen las calidades de servicio, ya sean de seguridad, transacciones y alta disponibilidad (Montilva C., y otros, 2003).

Mantenibilidad

Es una característica primordial para los componentes de software, puesto que los mismos deben adaptarse a diferentes entornos independientemente del hardware. Un componente es mantenible cuando puede ser modificado frente a cambios en los requisitos o especificaciones. En el departamento se hace difícil que componentes desarrollados por diferentes proyectos puedan ser mantenibles. Esto se debe a

[1] *Internet Protocol* (Protocolo de Internet)

que los mismos son implementados sobre arquitecturas diferentes y a la hora de adaptarlos a las necesidades de otros proyectos dejan de cumplir con las reglas establecidas para su creación.

Interoperabilidad

Una aproximación para unir requisitos de sistemas futuros a través de la integración de los mismos es la formación de un "sistema de sistemas" al interconectar componentes aislados. La interoperabilidad no incluye solamente la habilidad de los sistemas para intercambiar información, sino también la capacidad de interacción y la ejecución de tareas conjuntas. Por tanto el objetivo es crear un "sistema de sistemas" que no provea solamente interconectividad entre los mismos sino que logre una unión de sistemas interoperables. Una primera dificultad para lograr la interoperabilidad entre componentes heterogéneos de una unión de sistemas es que estos se suelen desarrollar independientemente, sin ningún requisito para interoperar (Zapata, y otros, 2009).

La interoperabilidad entre componentes en el departamento es ineficiente puesto que los mismos son creados en los proyectos rigiéndose por una estructura que se define en el mismo, lo que imposibilita que componentes desarrollados en diferentes proyectos puedan comunicarse entre sí.

Reutilización

Constituye la acción de emplear elementos de software ya creados en desarrollos anteriores para reducir la complejidad y el tiempo durante el proceso de desarrollo de software. Implica no construir un sistema a partir de cero, sino utilizar componentes ya programados y probados para posteriores modificaciones que cumplan con los requisitos que se deseen. En el departamento de Señales Digitales la reutilización de componentes resulta ineficiente, puesto que los mismos no pueden ser utilizados ni adaptados en un proyecto diferente al que lo desarrolló debido a que la estructura que posee cada uno es distinta.

Interfaces

Se encargan de definir las operaciones que puede realizar un componente y facilitan un mecanismo para lograr una interconexión entre componentes y controlar las dependencias que los mismos generen. En fin es el medio que permite que un componente determinado exponga sus funcionalidades.

Entorno de desarrollo para el trabajo con componentes

Es el conjunto de recursos y componentes que rodean a un objeto o componente dado y que definen las acciones que sobre él se solicitan, así como su comportamiento. Se pueden definir al menos dos clases de entornos para los componentes: el entorno de ejecución y el de diseño. El primero de ellos es el

ambiente para el que se ha construido el componente y en donde se ejecuta normalmente. El entorno de diseño es un ambiente restringido que se utiliza para localizar, configurar, especializar y probar los componentes que van a formar parte de una aplicación, y en donde los mismos han de poder mostrar un comportamiento distinto a su comportamiento normal durante su ejecución (Fuentes, y otros).

Se determina entonces que un entorno es el intermediario entre el usuario y una aplicación determinada, que responde de forma activa a las solicitudes realizadas a los componentes que la integran, además de definir los patrones de comportamiento de los componentes que se deseen integrar a la aplicación.

Eventos

Un evento no es más que una acción emitida por un componente para indicar a los componentes de su entorno de cambios de comportamiento entre su estado actual y el deseado. Es el mecanismo a través del cual se propagan situaciones que ocurren en un sistema determinado.

1.2. Objeto de estudio

1.2.1. Descripción general del objeto de estudio

El desarrollo de software basado en componentes (DSBC) pertenece al paradigma de programación de sistemas abiertos y parte de la evolución del paradigma de programación orientado a objetos, planteando que el software debe ser desarrollado sobre la base de componentes prefabricados. Busca reducir el tiempo de trabajo, el esfuerzo que requiere implementar una aplicación y los costos del proyecto, y de esta forma incrementar el nivel de productividad de los grupos de desarrolladores y minimizar los riesgos globales sin incurrir en gastos exorbitantes. El hecho de poder integrar componentes de software para obtener aplicaciones funcionales ofrece otra gran ventaja concerniente en poder personalizar los productos a la medida de las necesidades de los clientes. Esto permite a las empresas y a los desarrolladores adquirir según sus necesidades las tecnologías que más se adapten a las mismas y de esta manera no incurrir en gastos por cuenta de licenciamiento o soporte y actualización. Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización la calidad de las aplicaciones basadas en componentes mejoran con el paso del tiempo. Esta característica es la clave principal para garantizar la evolución exitosa de los productos.

Beneficios del Desarrollo de Software basado en Componentes

La adopción del paradigma de programación basado en componentes brinda una serie de beneficios relacionados con las reutilización, reducción del esfuerzo y el aumento de la productividad. A continuación

se muestran las principales características que hacen al DBSC una de las ideas más prometedoras de estos tiempos.

Reutilización del software: la posibilidad de utilizar constantemente componentes de software que se encuentren previamente desarrollados por una entidad o por terceros aporta la ventaja de reducir notablemente los tiempos de desarrollo y el esfuerzo del equipo de desarrollo.

Simplifica las pruebas: el uso de componentes de software permite que los mismos puedan ser probados de manera unitaria para garantizar que cumplen con las funcionalidades para las que fue diseñado. Esta característica influye directamente en la reducción de los márgenes de error; así como en la ubicación de manera más acelerada de posibles fallas.

Simplifica el mantenimiento del sistema: en un sistema desarrollado a partir de componentes débilmente acoplados entre ellos, los desarrolladores se encuentran en la libertad de agregar o separar componentes o cambiar los mismos para lograr cumplir con éxito con las funcionalidades del sistema.

Mayor calidad: debido a que los componentes pueden ser constantemente mejorados por los desarrolladores de los mismos, los sistemas ensamblados a partir de estos incrementarán su calidad con el paso del tiempo debido a que nuevas versiones de los componentes podrán ser adheridas o sustituirán viejos componentes agregándole nuevas funcionalidades, mejor concebidas e implementadas.

Ciclos de desarrollo más cortos: en el DSBC la adición de un nuevo componente de software tomará días como mucho. En caso de que el equipo de desarrollo deba asumir la implementación de las funcionalidades dadas, podría tardarse meses o aun más tiempo en llevar la actividad a término e integrarla al sistema.

1.3. Situación actual del dominio del problema

A continuación se presenta la situación actual por la cual atraviesan los proyectos del departamento de Señales Digitales.

La forma de desarrollar software en el departamento en su mayoría está orientada a resolver las necesidades de clientes específicos con requisitos cambiantes. En determinados casos el departamento se ve afectado por el atraso en las fechas de entrega de los productos y el incumplimiento de los tiempos acordados con los clientes, así como la escasez de personal lo suficientemente capacitado como para garantizar la calidad de los productos. Por otra parte existe una escasa comunicación entre los proyectos

del departamento en cuanto al desarrollo de aplicaciones y/o componentes de software de manera individual que comparten características similares. El departamento no está orientado a la reutilización de componentes, lo que conlleva a desarrollar aplicaciones desde cero y no sobre componentes ya construidos, debido a esto se plantea que no existe un aprovechamiento de las funcionalidades que brindan los componentes ya desarrollados. Esto trae consigo que dichos componentes para ser reutilizados por otros proyectos deban ser migrados o el código de los mismos sea copiado y adaptado.

La implantación del DSBC en los proyectos garantizaría que se construyan sistemas donde los desarrolladores tengan la libertad de agregar, separar o cambiar componentes en función de cumplir con los requisitos de los clientes. La utilización de componentes previamente desarrollados y probados permite reducir los tiempos de desarrollo, el esfuerzo del equipo y aumentar la calidad de los productos. Provocaría una mayor comunicación entre los proyectos debido a que los mismos podrían resolver propósitos comunes realizando tareas dependientes, además de brindar la posibilidad de ser utilizados sin comprender su funcionamiento interno, solo habría que conocerlos y adaptarlos, pues existen casos en que los equipos de desarrollo de los proyectos poseen escasos conocimientos y experiencia.

1.4. Análisis de soluciones existentes

A continuación se realiza un estudio de diferentes frameworks de desarrollo que operan sobre el paradigma de programación basado en componentes con el propósito de brindarle al departamento de Señales Digitales una solución que permita la interacción entre componentes desarrollados sobre una misma arquitectura.

1.4.1. GCF (*Generic Component Framework*)

Es una biblioteca de marco de trabajo para Qt que permite la creación de aplicaciones de software altamente extensibles y mantenibles. En el ámbito de GCF los plugins[2] son llamados componentes, representados como un archivo de biblioteca dinámica. El elemento principal de una aplicación de este tipo es que ofrece una aplicación contenedora de *widget*[3] dentro del cual se pueden mezclar los elementos de los componentes, ya sean elementos de menú o de barra de herramientas. Permite a los desarrolladores comunicarse con otras aplicaciones GCF por medio de señales y ranuras. Una aplicación GCF debe ir acompañada de un archivo XML, el cual contiene información acerca de los componentes

[2] Es un complemento que brinda a un software determinado una función específica.
[3] Componente gráfico para el desarrollo de interfaces de usuario

que deben ser cargados por la aplicación. También contiene los principales parámetros de configuración para el software y la IPC[4].

La columna vertebral de dicho framework la componen tres elementos fundamentales:

- **La clase AbstractComponent**: de esta clase heredan los componentes que se van a desarrollar por el usuario, la misma garantiza que dichos componentes tengan una misma estructura, proporcionado una serie de datos básicos como el nombre de un componente y la ubicación del mismo. GCF ofrece un conjunto de servicios como por ejemplo: combinar la interfaz gráfica de usuario: acciones, menús, barras de herramientas, *widgets* y objetos del componente en la ventana principal de la aplicación. Decidir cómo los *widgets* de otros componentes se pueden colocar, quitar, estar oculto o mostrar en los *widgets* de este componente. GCF es capaz de informar al componente cuando ningún otro componente se crea o destruye en el sistema. Proporciona medios a través de los cuales el componente puede explorar objetos proporcionados por otros componentes en el sistema.
- **Archivo XML:** es un archivo fundamental para GCF. En él se describen los objetos, aparatos, acciones, menús y barras de herramientas que expone un componente. El mismo recoge la forma en que se relacionarán los componentes entre ellos y la forma en que serán cargados. Todo componente GCF debe tener un archivo XML.
- **La clase Application:** de esta clase heredan las aplicaciones creadas por el usuario. Para una correcta utilización de la misma se debe crear un objeto en la clase principal del componente que se esté desarrollando, ya que al ser ejecutado el objeto de la aplicación busca el archivo Application XML y analiza su contenido (VCreate Logic, 2012).

Estudios realizados sobre el framework GCF indican que posee funcionalidades bastante atractivas para ser utilizadas en el departamento. Esto se debe a que cuenta con un modelo de componentes que permite que los componentes cargados por el framework puedan interactuar entre sí. Del mismo se tomó la idea de dividir el sistema a desarrollar en dos partes fundamentales: una destinada a la creación de componentes y la otra a posibilitar que el usuario pueda acceder a un entorno de trabajo en el cual pueda interactuar con los componentes. Para la creación de componentes GCF posee una clase virtual pura que establece una misma estructura para cada componente creado por el usuario. La idea de que cada

[4] *Inter Process Comunication*(Comunicación entre procesos)

componente creado por el usuario esté acompañado por un archivo XML es otro de los elementos que hicieron de la estructura este framework la solución adecuada a los problemas existentes. En dicho archivo se especifica el nombre del componente, su ubicación en el área de trabajo en caso de poseer una interfaz gráfica de usuario, así como la conexión con otros componentes. También la idea de que las aplicaciones creadas por el usuario contaran con un archivo XML donde se especificaran los componentes que se deseen cargar en el entorno de trabajo brindó grandes facilidades para el desarrollo de la solución. Por tanto se concluye que el estudio del framework GCF ha sido aprovechado por el equipo de desarrollo, ya que las ideas bases para desarrollar el sistema fueron aportadas por el mismo. Sin embargo no puede ser adaptados a las necesidades del departamento puesto que se rige por la licencia GNU/GPL y el departamento tiene como política no producir software bajo este tipo de licencia, ya que para la comercialización de productos se debe estregar su código fuente.

1.4.2. Catalyst

Es un framework para el desarrollo de aplicaciones web de forma rápida y eficiente escrito en Perl, que responde al paradigma basado en componentes y al patrón Modelo Vista Controlador (MVC). Soporta Ajax y es multiplataforma, además de ser inspirado por Ruby on Rails y considerarse un framework de mucha aceptación en todo el mundo. Utiliza componentes o *plugins* para la gestión de sesiones, autenticación de usuarios y almacenamiento en la caché. Tiene incluido un servidor de desarrollo integrado que es capaz de reiniciarse automáticamente cuando los aspectos de configuración que posee son modificados. Su distribución se realiza mediante la plataforma CPAN[5], la cual brinda una idea de la calidad del código desarrollado y permite utilizar la mayoría de los módulos de los que dispone. Aunque es posible utilizar cualquier módulo disponible en CPAN Catalyst incluye una lista de *plugins* que facilitan el desarrollo web (Saavedra).

Una vez realizado un estudio del framework Catalyst se determina que cuenta con un conjunto de características que son necesarias para resolver los problemas existentes en el departamento. Es un framework multiplataforma que se apoya en componentes para la creación de sistemas, sin embargo se necesita que la solución esté destinada a la creación de aplicaciones de escritorio y Catalyst está orientado al desarrollo de aplicaciones web. No cuenta con una clase que establezca una arquitectura

[5] *Comprehensive Perl Archive Network*(Archivo de Red Exhaustivo para Perl)

común para el desarrollo de componentes por lo que imposibilita que los programadores del departamento puedan seguir un estándar común en cuanto al desarrollo de componentes.

1.4.3. LibCoral

Este framework fue implementado usando el lenguaje de programación C++ para el trabajo con componentes. La primera versión fue Coral 0.6 liberada el 22 de junio de 2011 por Thiago Bastos y la última la Coral 0.7.1 el 14 de diciembre de 2011. Luego de esta última versión el proyecto fue abandonado (Coral, 2011).

Dicho framework aunque se diseñó para desarrollar sistemas sobre componentes fabricados y probados en procesos de desarrollo anteriores y está orientado al desarrollo de aplicaciones de escritorio utilizando el lenguaje de programación C++, no cumple con los requisitos para el desarrollo de la solución. Esto se debe a que no está destinado al desarrollo de aplicaciones sobre el framework Qt y no puede ser utilizado por el departamento puesto que no recibió soporte por parte del proyecto que lo creó y fue abandonado quedando sus versiones obsoletas.

1.4.4. JSF (*Java Server Face*)

Es un framework muy usado para el desarrollo de aplicaciones web basadas en componentes de interfaz de usuario utilizando java como lenguaje de programación. Posee una gran facilidad para desarrollar aplicaciones web a partir de un conjunto de componentes de GUI[6] reusables. Posibilita el desarrollo de aplicaciones web complejas de forma sencilla y entendible consumiendo el menor tiempo posible en la realización de las mismas. Unas de las ventajas principales de dicho framework es que provee un conjunto de componentes de interfaz de usuario predefinidos y un modelo de componentes de interfaz de usuarios. Además cuenta con una técnica para procesar en el servidor los eventos que se generan en el cliente a partir de la interacción del usuario con la aplicación. Permite utilizar tecnologías como XML/XSLT, HTML, XHTML. Soporta el patrón MVC separando el código de la presentación del de la lógica de negocio. Cuando un cliente solicita una nueva página el estado del componente es guardado en el servidor y se recupera una vez que el mismo retorne una respuesta a la petición realizada (Díaz, y otros).

Una vez realizado un estudio del framework JSF se determina que posee características que pueden servir de guía para el desarrollo de la solución. Cuenta con un modelo de componentes a través del cual

[6]GraficUser Interface(Interfaz de Usuario Gráfica)

se define como van a interactuar los componentes que son reconocidos por el framework y posee un conjunto de componentes que permiten su reutilización de acuerdo a las necesidades existentes. Aunque JSF opera sobre el paradigma de programación basado en componentes y cuenta con un conjunto de características que reducen el tiempo y el esfuerzo de los desarrolladores está orientado al desarrollo de aplicaciones web sobre el lenguaje de programación java y se necesita una solución destinada a la creación de aplicaciones de escritorio que trate a los componentes de acuerdo a su funcionalidad y estructura.

1.5. Conclusiones Parciales.

Una vez abordados los principales elementos asociados al desarrollo de software basado en componentes y conocidas las necesidades existentes en el departamento de Señales Digitales se concluye que para el desarrollo de la solución es necesario operar sobre el paradigma de programación basado en componentes, con el propósito de establecer una estructura estándar para cada componente creado por el programador y que los mismos puedan ser reutilizados para el desarrollo de aplicaciones futuras. Los frameworks estudiados aportaron ideas que son imprescindibles para la solución; tales como contar con un modelo de componentes el cual establezca las reglas que deben seguir los mismos para lograr una interacción entre ellos. El framework GCF sirvió como punto de partida para el desarrollo de la solución; por la forma en que el mismo trata a los componentes y las reglas que establece para que interactúen entre sí. Además, posee una estructuración organizada y entendible de las clases con las que cuenta para llevar a cabo los procesos referentes a los componentes y a las aplicaciones donde serán visualizados.

CAPÍTULO 2. TENDENCIAS Y TECNOLOGÍAS ACTUALES A DESARROLLAR

Introducción

Antes de implementar la solución que se propone, una de las primeras tareas a ejecutar es llevar a cabo un estudio de las tendencias actuales, tecnologías existentes y selección de las herramientas que giran alrededor del desarrollo de aplicaciones de escritorio, puesto que las mismas permiten guiar, visualizar, entender y documentar el proceso de desarrollo de software. Por tanto en el presente capítulo se describen los aspectos relacionados a las tendencias y tecnologías actuales a desarrollar en la producción de software, centrando la atención en las que se utilizan para el desarrollo del framework basado en componentes. Los aspectos mencionados están determinados por: la metodología para guiar el desarrollo del sistema de software, el lenguaje de modelado, las herramientas, el lenguaje de programación y el framework y entorno de desarrollo.

2.1. Metodología de desarrollo de software

Las metodologías ágiles dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque muestra su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo manteniendo una alta calidad. Entre ellas se puede encontrar la metodología Programación Extrema (XP, en inglés *Extreme Programming*).

2.1.1 Metodologías Ágiles. Programación Extrema (XP)

La Programación Extrema es una metodología que se centra en el desarrollo de software y en un conjunto de reglas que giran alrededor de las necesidades del cliente, con el objetivo de lograr un producto de buena calidad con una reducción considerable de tiempo. Básicamente se encarga de potenciar las relaciones interpersonales como un factor clave para el éxito, por lo que opera directamente con el cliente. Uno de los elementos a tener en cuenta en esta metodología es definir un estándar en el tipo de codificación, debido a la necesidad de poner en práctica la programación en pares, lo que conlleva a que los programadores tengan bien definido un estilo común de programación. Las pruebas constituyen una acción más que necesaria en cada iteración, ya que permite que se prevean errores a medida que se programe. El fin de XP es generar versiones de un sistema tan pequeñas como sea posible, pero que proporcionen un valor adicional claro desde la perspectiva del negocio. Utiliza historias de usuarios para cubrir la falta de casos de uso (Letelier, y otros, 2006).

Dicha metodología se caracteriza por la comunicación que establece entre los clientes y desarrolladores, la simplicidad que posee al desarrollar y codificar los módulos de sistemas y por la retroalimentación concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales. Enfatiza en el trabajo en equipo, pues tanto los clientes como los desarrolladores son parte del mismo equipo; dedicado a entregar software de calidad. Propone que se desarrolle en pares de programadores, ambos trabajando juntos en un mismo ordenador. Aunque parezca que esta práctica duplica el tiempo asignado al proyecto y con ello los costos en recursos humanos, la misma minimiza los errores y se logran mejores diseños. La refactorización durante todo el proceso de desarrollo es otro aspecto importante que plantea esta metodología; puesto que permite que el equipo mejore constantemente el diseño de sistemas, haciéndolos más fáciles de entender, sencillos y cohesivos. Las integraciones continuas y de corta duración permiten descubrir errores tempranamente e incorporar sus correcciones como tareas en las iteraciones siguientes, además de obtener software funcional desde los inicios del desarrollo del mismo.

2.2. Fundamentación de la metodología seleccionada

Teniendo en cuenta el análisis realizado sobre la metodología XP se decidió utilizarla para guiar el proceso, debido a que el equipo de desarrollo sólo está compuesto por dos personas; de las cuales una de ellas es programador. El sistema propuesto debido al alto riesgo en desarrollo que presenta y a los requisitos que engloba va a estar inmerso en un proceso constante de cambios por lo que se hace imprescindible la presencia del cliente durante todo el proceso de desarrollo. No se tiene un proceso de negocio bien definido por lo que es necesario obtener software funcional desde los inicios del desarrollo, con esto y la retroalimentación continua del cliente y el equipo se logra una mayor comprensión del sistema a desarrollar. Además contar con un desarrollo guiado por pruebas garantiza que se corrijan errores a medida que se programe; garantizando que la calidad del producto sea la esperada.

2.3. Lenguaje de Modelado

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimientos sobre los sistemas que se deben construir.

2.3.1. UML 2.0

Entre los lenguajes de modelado visual más conocidos se encuentra el Lenguaje de Modelado Unificado (UML, en inglés *Unified Model Language*), el cual se usa para entender, diseñar, hojear, configurar, mantener y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos

de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. UML incluye conceptos semánticos, notación y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas (Rumbaugh, y otros).

Con la utilización de UML es posible llevar a cabo las siguientes funciones:

- **Visualizar:** permite expresar de forma gráfica un sistema de forma que otro lo pueda entender.
- **Especificar:** permite especificar cuáles son las características de un sistema antes de su construcción.
- **Construir:** a partir de los modelos especificados se pueden construir los sistemas diseñados.
- **Documentar:** los propios elementos gráficos sirven como documentación del sistema desarrollado y pueden servir para su futura revisión.

Conocidas las funciones que trae consigo el empleo de un lenguaje de modelado se decidió utilizar UML 2.0 para facilitar al programador la implementación del sistema una vez que sea diseñado.

2.4. Herramientas CASE (*Computer Aided Software Engineering*)

"Ingeniería de Software Asistida por Computadora (del inglés *Computer Aided Software Engineering*-CASE) es un tipo de ingeniería de software en la que se intenta aumentar la eficacia de sus procesos, al soportar la realización de las tareas con el uso de tecnologías (Pérez, 2002).

Las herramientas *CASE* son usadas en algunas de las fases de desarrollo de sistemas de información, incluyendo análisis, diseño e implementación. Tienen como objetivo fundamental proveer un lenguaje para describir el sistema general, que sea lo más explícito posible a la hora de generar determinados programas.

2.4.1. Visual Paradigm for UML 8.0

Visual Paradigm for UML es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Este software ayuda a una más rápida construcción de aplicaciones de calidad y a un menor coste (Headquarters, Company, 2010).

Visual Paradigm for UML es una herramienta CASE que utiliza UML como lenguaje de modelado. Entre sus principales características se encuentran la disponibilidad en múltiples plataformas y que soporta el ciclo de vida completo del desarrollo de software. Brinda a los usuarios un entorno que posibilita la creación de diagramas para UML 2.0. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar

código desde diagramas y generar documentación. Permite generar toda la documentación posible a partir de lo que se hace, bajo determinados estándares previamente establecidos. Está concebido bajo el paradigma orientado a objetos e incorpora el soporte para trabajo en equipo, lo que permite que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios hechos por sus compañeros.

2.5. Justificación de la herramienta CASE seleccionada

Se determina que para llevar a cabo todo el proceso referente a la ingeniería de software del sistema en desarrollo, la herramienta CASE indicada es *Visual Paradigm for UML* en su versión 8.0. Se selecciona en primer lugar porque es la herramienta CASE estándar que utiliza la universidad en proyectos de gran alcance y al ser multiplataforma puede ser utilizada en diferentes sistemas operativos. Brinda la posibilidad de modelar un sin número de diagramas de clases, facilitando la codificación desde diagramas, la organización y el entendimiento por parte de los desarrolladores. Además, porque contribuye a lograr mayor rapidez en la construcción de aplicaciones informáticas.

2.6. Lenguaje de Programación

Los lenguajes de programación surgen con la necesidad de la comunicación hombre-máquina. En la informática no son más que un conjunto de instrucciones, reglas, operadores para crear programas que representen procedimientos, cálculos y decisiones. Además constituyen una vía para que el programador pueda lograr una comunicación entre el hardware y el software. Los lenguajes de programación pueden ser de dos tipos:

Bajo Nivel: en este tipo de lenguaje las instrucciones de los programas no poseen gran complejidad, debido a que se acercan al funcionamiento de la máquina. El código máquina y el ensamblador constituyen un ejemplo de lo antes expuesto.

Alto Nivel: consta de instrucciones independientes de la máquina; ha de ser compilado o interpretado para traducir su código en otro de bajo nivel, en lenguaje máquina. Por lo que constituye el lenguaje de programación más próximo a los usuarios en el proceso de desarrollo de software.

Resaltar que cuanto más se acerque un lenguaje de programación al lenguaje natural del usuario se considerará de más alto nivel.

2.6.1. C++

Es una extensión del lenguaje de programación C, se considera un lenguaje de alto nivel que al mismo tiempo se basa en instrucciones cercanas a la máquina. Esto se debe a que aunque C++ es independiente del tipo de máquina existen sistemas operativos o partes de ellos implementados bajo dicho lenguaje. Permite programar además compiladores, procesadores de texto, entre otras aplicaciones. Otra característica importante del lenguaje en cuestión es su basamento sobre los paradigmas de programación: estructurado, genérico y orientado a objetos. Posee una biblioteca estándar que es rica en cuanto a clases y funciones: la STL (*Standard Template Library*). Proporciona facilidades con respecto a la creación de estructuras de datos integradas fuertemente al lenguaje. Esto facilita el trabajo de los programadores en cuanto a la creación de tipos de datos con operaciones asociadas. Además dichas estructuras constituyen una extensión natural de los tipos de datos primitivos, lo que contribuye a elevar el grado de claridad y entendimiento (Guérin).

Es un lenguaje versátil, potente a la hora de realizar sistemas complejos y muy empleados debido a la documentación que posee para su entendimiento y utilización. Existen muchos algoritmos y librerías implementadas en C++, por lo que se puede adaptar fácilmente.

2.7. Justificación del lenguaje de programación seleccionado

Luego de valorar algunos de los lenguajes de programación vinculados total o parcialmente con los objetivos trazados por la presente investigación se decide utilizar C++.Esto se debe en primer lugar a que es el lenguaje nativo del framework Qt. Se espera un sistema de software complejo que sea adaptable, por lo que no se debe prescindir de la versatilidad y potencia del lenguaje en cuestión. Además C++ tiene a su favor que es utilizado por un conjunto de herramientas libres de desarrollo, es orientado a objetos y en cuanto a ejecución es uno de los más rápidos y eficientes.

2.8. Framework de desarrollo

2.8.1. Framework Qt 4.8

Qt es un framework multiplataforma y orientado a objetos. La función más conocida de Qt es la creación de interfaces de usuario, sin embargo no se limita a esto, ya que también provee varias clases para facilitar ciertas tareas de programación como: el manejo de *sockets*[7], soporte para programación multihilo,

[7] Es un método para la comunicación entre un programa cliente y uno servidor en una red, por lo que se define como el punto final en una conexión.

comunicación con bases de datos, manejo de cadenas de caracteres y también para el desarrollo de programas sin interfaz gráfica; como herramientas de la consola y servidores (Martínez, 2011).

Está programado en C++ y brinda soporte para un número elevado de lenguajes de programación de forma no oficial. Qt ofrece una suite de aplicaciones para facilitar y agilizar las tareas de desarrollo, las cuales se mencionan a continuación:

***QtAssistant*:** herramienta para visualizar la documentación oficial de Qt.
***QtDesigner*:** herramienta para crear interfaces de usuario.
***QtLinguist*:** herramienta para la traducción de aplicaciones.
***QtCreator*:** es un *IDE* para el lenguaje C++, especialmente diseñado para Qt, integra las primeras dos herramientas mencionadas.

2.9. Justificación del framework de desarrollo seleccionado

Se considera que para implementar la solución de la presente investigación lo ideal sería la utilización del framework Qt en su versión 4.8. Esto se debe en primer lugar a que es el framework que se utiliza en la mayoría de los proyectos del departamento para desarrollar software. Por otra parte brinda una serie de facilidades a la hora de programar, al ser orientado a objetos y contar con un conjunto de bibliotecas; con clases y herramientas incluidas, las mismas están bien documentadas y son muy fáciles de usar. Otra razón es la característica de ser multiplataforma. Esto facilita que la solución software pueda adaptarse tanto para el sistema operativo Linux, Windows o Mac, permitiendo que una aplicación pueda ser compilada y utilizada en cualquier plataforma sin necesidad de cambiar el código constantemente. Además Qt es completamente gratuito para aplicaciones de código abierto y una de sus licencias: la LGPL, permite que se utilice gratuitamente con fines comerciales.

2.10. Entorno de Desarrollo Integrado (IDE)

2.10.1. Qt-Creator 2.4

Un Entorno de Desarrollo Integrado (del inglés, *Integrated Development Environment*, IDE, por sus siglas en inglés) es un programa compuesto por una serie de herramientas que utilizan los desarrolladores para escribir el código. Puede soportar varios lenguajes de programación o puede estar diseñado para uno únicamente (Viñolo, y otros, 2012).

Qt-Creator. La principal herramienta del Nokia Qt SDK es el IDE Qt-Creator, pues trae consigo una serie de componentes integrados: un editor de código C++, un administrador de proyectos, documentación de la

API[8] y de todas las funcionalidades que provee Qt. Además trae una serie de ejemplos de distintos tipos de aplicaciones y la gran mayoría de ellas completamente documentadas (Briñes, y otros, 2011).

Qt-Creator es un IDE multiplataforma que se ejecuta tanto en Windows, Linux o Mac OS. Facilita la implementación a los desarrolladores al crear aplicaciones tanto para escritorio como para plataformas de dispositivos móviles. Entre las principales características de Qt-Creator se encuentran:

- **Editor de código sofisticado:** ofrece soporte para la edición de C ++ y QML, ayuda sensible al contexto, completado de código y navegación.
- **Control de versiones:** se integra con la mayoría de los sistemas de control de versiones populares, incluyendo Git, Subversion, Bazaar, Perforce, CVS [9] y Mercurial.
- **Construcción y gestión proyecto:** si se importa un proyecto existente o se crea uno desde cero, Qt-Creator genera todos los archivos necesarios.

2.11. Justificación del IDE seleccionado

Se empleará como IDE Qt-Creator en su versión 2.4, debido a que está orientado al desarrollo de aplicaciones a través del framework Qt. Además ofrece gran compatibilidad con varios sistemas operativos, posee riquezas en sus funciones y bibliotecas permitiendo el desarrollo de interfaces amigables y por su alto rendimiento con C++ (lenguaje de programación seleccionado para el desarrollo).

2.12. Conclusiones Parciales

Una vez realizado un estudio referente a la situación actual de las tecnologías, herramientas y metodologías se determinó que la selección de las mismas está sustentada por las necesidades del departamento de Señales Digitales y las políticas establecidas por la universidad. Por tanto se concluye que la propuesta para dar cumplimiento a las tareas trazadas debe consistir en utilizar XP como metodología de desarrollo, UML 2.0 como lenguaje de modelado; Visual Paradigm for UML 8.0 como herramienta CASE; C++ como lenguaje de programación y Qt 4.8 y Qt-Creator 2.4 como framework e IDE respectivamente. Esto garantiza la facilidad por parte de los programadores a la hora de codificar el sistema en desarrollo y la comprensión y el entendimiento por parte de los usuarios finales.

[8] *Application Programation Interface* (Interfaz de Programación de Aplicaciones)
[9] *Version Control Systems* (Sistemas de Control de Versiones)

CAPÍTULO 3. ANÁLISIS Y DISEÑO

Introducción

Durante el desarrollo de software se está evidenciando poca claridad con respecto a definir el ámbito de un proyecto de software; de ahí que se desconozca el tiempo estimado y el esfuerzo a emplear para el desarrollo del mismo. Una buena planificación constituye la respuesta a estos problemas, puesto que brinda al equipo de desarrollo una guía como factor clave para el éxito. De acuerdo a las metodologías más conocidas XP se comporta de una manera diferente en cuanto al diseño de un sistema y aunque menosprecie el uso de técnicas como UML, arquitectura y patrones es una metodología muy flexible y adaptable. La metodología XP es partidaria de los diseños simples donde se ponga de manifiesto la refactorización del código, eliminando la duplicidad y garantizando la simplicidad del mismo. Se considera que para la construcción de la propuesta de solución de acuerdo a la metodología seleccionada se deben estudiar las fases de exploración, planificación e iteraciones. En el presente capítulo se ponen de manifiesto los elementos asociados al análisis y diseño del sistema a desarrollar. Para llevar a cabo el análisis es necesario establecer un modelo conceptual que sirva como base para diseñar y construir objetos en el sistema. Por otra se hace imprescindible definir las características del sistema propuesto, obteniéndose de las mismas las historias de usuario, iteraciones a las cuales serán asignadas las historias de usuario, cronograma de liberación del producto y requisitos no funcionales que debe poseer dicho sistema. Una vez culminado el proceso de análisis se lleva a cabo el diseño donde se traza la línea base de la arquitectura y se definen los patrones para el diseño de cada clase. Se desarrolla el diagrama de clases del diseño y las tarjetas CRC en función de conocer el comportamiento de las clases que intervienen en la implementación y las colaboraciones que presentan entre ellas.

3.1. Modelo Conceptual

Este modelo incluirá los conceptos y sus relaciones y se describirá mediante un diagrama de clases UML, en el que los conceptos se representan mediante clases del dominio (Ortín, y otros). Dicho modelo es una representación estática del sistema debido a que no describe su comportamiento dependiente del tiempo, aunque sirve como base para diseñar y construir objetos en el contexto del mismo. Los objetos del dominio son elementos que existen en el entorno en que trabaja el sistema.

3.1.1. Diagrama de clases del Modelo Conceptual

Una vez realizado un análisis exhaustivo del problema en cuestión se llega a la conclusión de que el negocio del presente trabajo posee un bajo nivel de estructuración. No se definen concretamente los

procesos del mismo, de ahí que se decide dar un nuevo enfoque a todo el proceso. La metodología seleccionada en el capítulo anterior es caracterizada por la simplicidad y adaptabilidad; por lo que permite el empleo de diagramas UML y entre ellos el modelo conceptual. Se decidió realizar dicho modelo ya que permite de manera visual mostrar al equipo de desarrollo los principales conceptos que se manejan en el dominio del sistema en desarrollo. Además, contribuye a un mayor entendimiento del problema y sirve como base para futuras acciones. El mismo se realiza mediante un diagrama de clases de UML.

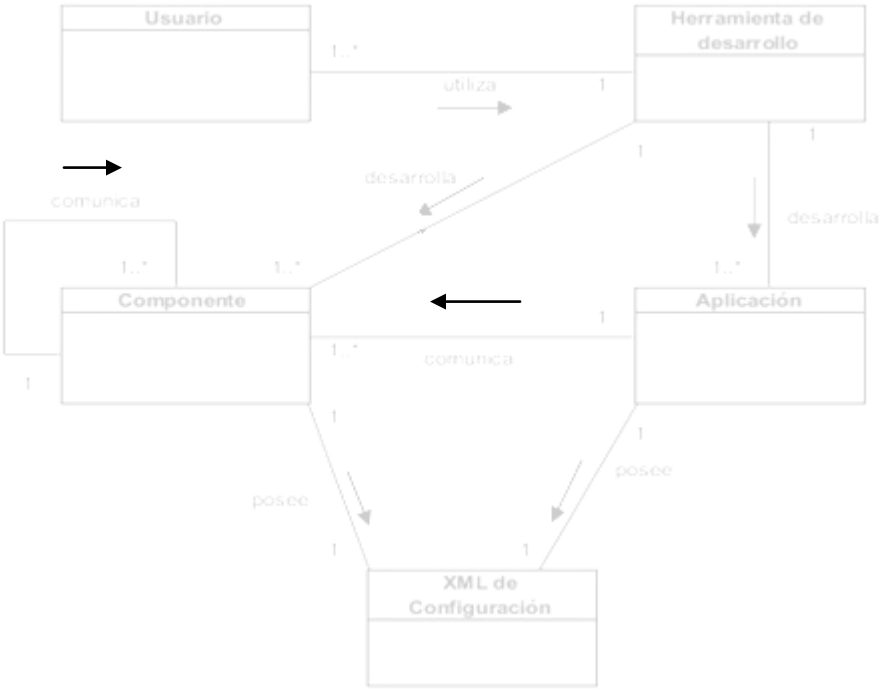

Figura 1 Modelo Conceptual

Principales conceptos del Modelo de Dominio

Usuario: es aquel individuo que hace uso de la herramienta de desarrollo en función de satisfacer las necesidades de los proyectos del departamento de Señales Digitales. Generalmente dicho usuario en un proyecto específico desempeña el rol de programador.

Herramienta de desarrollo: en este caso la herramienta es un framework compuesto por un conjunto de clases que realizan operaciones orientadas al trabajo con componentes.

Componente: engloba un conjunto de requisitos con interfaces definidas que cubren las necesidades de los proyectos del departamento de Señales Digitales.

Aplicación: programa que se desarrolla con el propósito de servir de entorno de trabajo para visualizar el trabajo con componentes.

XML de Configuración: es un archivo que especifica los parámetros de configuración de los componentes.

3.2. Descripción del sistema propuesto

Para llevar a cabo la realización del framework basado en componentes es necesario contar con dos partes fundamentales, la primera es la encargada de la creación de componentes y la segunda de crear un entorno de trabajo para la gestión de los mismos. Dichas partes deben estar relacionadas en función de visualizar las diferentes acciones que realizan los componentes.

Cada componente debe contar con un archivo XML donde se especifique el nombre del componente y la ubicación del mismo en caso de poseer una interfaz gráfica de usuario. Además, en caso de que reciba alguna señal emitida por otro componente se especifica el nombre del emisor, la señal que emite, el nombre del receptor y el evento que realiza.

Las aplicaciones creadas por el usuario deben ser capaces de cargar y comunicar componentes. Para ello deben contar con un archivo XML donde se especifique el nombre de los componentes que se deseen cargar. Además, deben contar con áreas definidas para la ubicación de componentes que posean una interfaz gráfica de usuario. En la raíz de dichas aplicaciones debe existir dos carpetas que permitan la copia del componente que se desee cargar y su archivo XML respectivamente.

3.3. Fase de Exploración

Es la fase en la que se define el alcance general del proyecto. En la misma el cliente define lo que necesita mediante la redacción de sencillas “historias de usuarios”. Los programadores estiman los tiempos de desarrollo en base a esta información. Debe quedar claro que las estimaciones realizadas en esta fase son primarias, puesto que estarán basadas en datos de muy alto nivel y podrían variar cuando se analicen más en detalle en cada iteración. Esta fase dura típicamente un par de semanas y el resultado es una visión general del sistema, y un plazo total estimado (Joskowicz, José, 2008).

Personal relacionado con el sistema

Uno de los aspectos fundamentales a tener en cuenta al comienzo del desarrollo de cualquier sistema de software es saber quién es el personal relacionado con el mismo; ya sea un miembro del equipo o un

usuario final. Las personas relacionadas con el sistema son tanto aquellas personas que desarrollan las funcionalidades del sistema como aquellas que hacen uso de las mismas con fines específicos.

Tabla 1 Personal relacionado con el sistema

Personal	**Justificación**
Usuario	Representa a la persona (programador) que desarrolle y haga uso de las funcionalidades del framework basado en componentes.

Historias de Usuario (HU)

Una historia de usuario es una simple descripción en lenguaje natural de una funcionalidad de software. Los detalles extra son agregados cuando la historia de usuario es implementada, siendo estos procesos de requisitos simples y sencillos. En cualquier momento estas historias pueden romperse, remplazarse, unirse o dividirse. Entre las principales características que debe tener una buena historia de usuario se encuentran: la historia debe ser entendida por el cliente (representa un concepto y no una especificación detallada), cada una debe devolver algún valor para el cliente, su tamaño debe ser tal que se puedan construir varias de ellas en una iteración (duración aproximada entre 1-3 semanas ideales), deben ser independientes unas de otras y cada una de ella debe ser testeable.

Para establecer la duración de las semanas en las estimaciones de las HU es necesario aclarar que una (1) semana equivale a los cinco (5) días laborales de la misma. Se considera válida esta aclaración; debido a que generalmente los cálculos erróneos de estimación de los tiempos de desarrollo se realizan en base a los siete (7) días de la semana. En caso de que la duración de las mismas sea mayor de 3 semanas es divida en pequeñas HU y en caso de ser menor es combinada con otras. Por tanto se definieron 4 HU atendiendo a las funcionalidades con las que debe cumplir el sistema a desarrollar.

Tabla 2 Historia de Usuario Crear componente

Historia de Usuario	
Número: 1	**Usuario:** Programador
Nombre historia: Crear componente	
Puntos estimados:1	**Puntos reales:** 1.4

Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Programador responsable: Adrián Martínez	
Descripción: El sistema crea un proyecto de tipo componente el cual hereda de la clase AbstractComponent con el objetivo de que los componentes desarrollados por el usuario sigan una misma estructura. Además es creado un archivo XML donde se especifica el nombre del componente, la ubicación donde el mismo será cargado en el entorno de trabajo en caso de poseer una interfaz gráfica de usuario y la conexión que presenta con otros componentes.	
Observaciones: -	

Tabla 3 Historia de Usuario Crear aplicación

Historia de Usuario	
Número: 2	**Usuario:** Programador
Nombre historia: Crear aplicación	
Puntos estimados:1	**Puntos reales:**1.4
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Programador responsable: Adrián Martínez	
Descripción: El sistema crea un proyecto de tipo aplicación donde se crea un objeto de la clase Application con el objetivo de acceder al entorno de trabajo. Por otra parte es creado un archivo XML donde se refleja el nombre de los componentes que el usuario desee cargar.	
Observaciones: -	

Tabla 4 Historia de Usuario Cargar componente

Historia de Usuario	
Número: 3	**Usuario:** Programador
Nombre historia: Cargar componente	
Puntos estimados: 2	**Puntos reales:** 3
Prioridad en negocio: Media	**Riesgo en desarrollo:** Alto
Programador responsable: Adrian Martínez – Yunet Gasca	
Descripción: se especifica en el archivo XML de la aplicación el nombre del componente que se desea cargar. El sistema busca el componente a cargar y su archivo XML en la carpetas Components y XML ubicadas en la raíz de la aplicación y procede a cargarlo.	
Observaciones: en caso de que el componente ya haya sido cargado anteriormente en la aplicación el sistema notifica que un componente solo se debe cargar una vez.	

Tabla 5 Historia de Usuario Comunicar componentes

Historia de Usuario	
Número: 4	**Usuario:** Programador
Nombre historia: Comunicar componentes	
Puntos estimados: 2.5	**Puntos reales:** 3
Prioridad en negocio: Media	**Riesgo en desarrollo:** Alto
Programador responsable: Adrián Martinez – Yunet Gasca	

Descripción: el sistema busca la sentencia de comunicación en el archivo XML del componente que va a ser receptor de la señal emitida. En esta sentencia se especifica que componente emite la señal y cuál es el encargado de recibirla. Una vez obtenida la sentencia de conexión el sistema procede a comunicar los dos componentes que se especifican en la misma.
Observaciones: el usuario debe especificar en el archivo XML del componente receptor de la señal su nombre y el del componente emisor de la señal de forma correcta. En caso de que la sentencia sea errónea ya sea porque los componentes no están cargados o porque la señal emitida o el slot receptor no existan, el sistema notifica el error.

3.4. Lista de reserva del producto

La lista de reserva del producto es una tabla que contiene los requisitos funcionales que debe cumplir el framework de componentes a desarrollar, ordenados según la prioridad en el negocio (Alta, Media). Por cada requisito se indica la estimación por semanas y el rol que lo estimó y por último contiene los requisitos no funcionales que requiere el sistema a desarrollar, teniendo en cuenta que los mismos presentan prioridad baja en el negocio.

Tabla 6 Lista de reserva del producto

Iteración	Descripción	Estimación	Estimado por
Prioridad: Alta			
1.	Crear componente	1.4 semanas	Programador
1.	Crear aplicación	1.4 semanas	Programador
Prioridad: Media			
2.	Cargar componente	3 semanas	Programador y Analista
3.	Comunicar componentes	3 semanas	Programador y Analista
Prioridad: Baja			
1. Requisito de usabilidad: es una aplicación de escritorio concebida con el fin de guiar al desarrollador en la implementación de componentes estructurados. Dicha			

aplicación será desplegada en las estaciones de trabajo destinadas al desarrollo de software basado en componentes en el departamento de Señales Digitales, donde se ajustará a las características del hardware de la estación de trabajo donde será utilizado.

2. Requisito de rendimiento: los tiempos de respuesta en lo que respecta a cargar los componentes deben ser rápidos, puesto que los componentes desarrollados por el usuario podrán adaptarse a las características del hardware de la aplicación del usuario donde serán integrados.

3. Requisito de portabilidad: se requiere que el framework de componentes sea multiplataforma.

4. Requisito de diseño e implementación: se utilizará el lenguaje de programación C++, el framework Qt y el entorno de desarrollo integrado Qt-Creator.

3.5. Fase de Planificación

Esta es una fase muy corta en la que el cliente establece la prioridad de cada historia de usuario con el propósito de que el programador conozca el orden en que serán implementadas las mismas. Una vez conocido en orden en que serán implementadas las HU el equipo de desarrollo define cuáles son las que estarán listas para la primera liberación. Por tanto la prioridad:

- **Alta:** se le otorga a las HU que resultan funcionalidades fundamentales en el desarrollo del sistema, a las que el cliente define como principales para el control integral del sistema.
- **Media:** se le otorga a las HU que resultan para el cliente como funcionalidades a tener en cuenta, sin que estas tengan una afectación sobre el sistema que se esté desarrollando.
- **Baja:** se le otorga a las HU que constituyen funcionalidades que sirven de ayuda al control de elementos asociados al equipo de desarrollo, a la estructura y no tienen nada que ver con el sistema en desarrollo.

Prioridad de las Historias de Usuario

La acción del cliente al establecer la prioridad para cada historia usuario es la que le permite conocer al programador en qué orden deberá implementar las mismas. Aclarar que las historias de usuario de menor prioridad son las más importantes y por tanto deben desarrollarse de primeras.

Tabla 7 Prioridad de las Historias de Usuario

Historia de Usuario	**Prioridad**
Crear componente	1
Crear aplicación	1
Cargar componentes	2
Comunicar componentes	2

El riesgo en su desarrollo

- **Alto:** cuando en la implementación de las HU se considera la posible existencia de errores que lleven a la inoperatividad del código.
- **Medio:** cuando pueden aparecer errores en la implementación de las HU que puedan retrasar la entrega de la versión.
- **Bajo:** cuando pueden aparecer errores que serán tratados con relativa facilidad sin que traigan prejuicios para el desarrollo del proyecto. El cliente y los desarrolladores trabajan en conjunto para definir como agrupar las HU para su lanzamiento.

Estimación de esfuerzo de las Historias de Usuario

Teniendo en cuenta la prioridad asignada a las historias de usuario es imprescindible estimar el esfuerzo necesario para el desarrollo de cada una de ellas. La estimación se basa en los conocimientos y velocidad que presente el equipo de desarrollo para llevar a cabo las tareas ingenieriles. Los puntos estimados son expresados en días ideales y se debe tener presente que los puntos estimados por historias de usuario no siempre se cumplen de acuerdo a como se planifica, puede que existan casos donde varíe. Por tanto la estimación del esfuerzo de las historias de usuario definidas en la presente investigación queda conformada de la siguiente forma:

Tabla 8 Estimación del esfuerzo necesario por Historia de Usuario

Historia de Usuario	**Esfuerzo necesario(Puntos estimados)**
Crear componente	7
Crear aplicación	7
Cargar componentes	15

Comunicar componentes	15

Cronograma de liberación

La metodología XP plantea que se debe llevar cabo una liberación cada vez que culmine una iteración; proponiendo una vez lanzada una un producto completamente funcional. La planificación de la liberación es realizada entre el cliente y el equipo de desarrolladores, el primero decide qué historias de usuario tienen la mayor prioridad y el segundo estima el tiempo que le llevará implementar las mismas. A continuación se presenta el cronograma de liberación de la investigación en curso:

Tabla 9 Cronograma de liberación

Iteración	**Fecha de liberación**
Primera	3ra semana de febrero de 2013
Segunda	3ra semana de marzo de 2013
Tercera	1era semana de abril de 2013

3.6. Fase de Iteración

Una vez identificadas y descritas por el cliente las historias de usuario y con ello la estimación del esfuerzo de cada una de ellas por el equipo de desarrollo se procede a realizar la planificación de las etapas de implementación del sistema. Por tanto se establece un plan de iteraciones donde se especifican las historias de usuario y el orden en que serán implementadas en cada iteración de acuerdo a su duración. La cantidad de iteraciones (IN) a realizar para el desarrollo de las historias de usuario está determinada por la suma de los puntos de esfuerzo (PE) para cada una de ellas dividida por la velocidad de iteración del equipo (VIE).

IN=PE / VIE

PE= 44 días (8.8 semanas)

IN=8.8 / 3.75

IN =2.3

La velocidad de iteración del equipo (VIE) se obtiene dividiendo la cantidad de desarrolladores (CD) entre el factor de dedicación (FD) al proyecto (en el caso de la presente investigación es de 4 [100%]) y

multiplicado por el tiempo de duración máximo de una iteración (DMI) (en el caso de la presente investigación es de 15 días máximo).

VIE= (CD / FD)*DMI

CD=1, FD =100% (4), DMI =15

VIE= (1 / 4) * 15

VIE=3.75

Teniendo en cuenta los cálculos realizados anteriormente se obtiene por valor de la velocidad de equipo aproximadamente 8 y la cantidad de iteraciones necesarias para desarrollar las historias de usuario 2. El plan de iteraciones queda conformado de la siguiente forma:

Tabla 10 Primera iteración

Iteración		
Numero: 1	**H.U (por orden):** 1,2	**Duración total:** 14
Descripción: esta iteración tiene como objetivo la implementación de las HU con prioridad alta. La misma se encargará de la creación de componentes y aplicaciones que sirvan para garantizar y probar el correcto funcionamiento del proceso referente a la gestión de componentes. Al final de esta iteración se contará con una primera versión de prueba, la cual será mostrada al cliente con el objetivo de obtener una retroalimentación para el grupo de trabajo.		

Tabla 11 Segunda iteración

Iteración		
Numero: 2	**H.U (por orden):** 3	**Duración total:** 15
Descripción: el objetivo de esta iteración es la implementación de la HU Cargar componente, al finalizar la misma se contará con una primera versión de prueba permitiendo esto que los componentes sean cargados por el entorno de trabajo Esta será mostrada al cliente con el objetivo de realizar cambios necesarios en base a la opinión del mismo.		

Tabla 12 Tercera iteración

Iteración		
Numero: 2	**H.U (por orden):** 4	**Duración total:** 15
Descripción: el objetivo de esta iteración es la implementación de la HU Comunicar componentes, al finalizar la misma se contará con una primera versión de prueba permitiendo que los componentes puedan interactuar entre sí a través del entorno de trabajo. Esta será mostrada al cliente con el objetivo de realizar cambios necesarios en base a la opinión del mismo.		

3.7. Diseño del Sistema

XP enfoca sus esfuerzos para conseguir diseños simples y sencillos. Es necesario procurarlo todo lo menos complicado posible para conseguir un diseño fácilmente entendible y que sea posible implementarlo. Esto garantizará menos tiempo y esfuerzo durante el proceso de desarrollo.

3.7.1. Propuesta de Arquitectura del Sistema

La arquitectura de software define la estructura del sistema. Esta estructura se constituye de componentes o piezas de código que nacen de la noción de abstracción, cumpliendo funciones específicas e interactuando entre sí con un comportamiento definido. Puede considerarse entonces como el “puente” entre los requisitos del sistema y la implementación (Eckel, 2012).

La metodología XP plantea que la arquitectura del sistema debe establecerse en la primera iteración. Por tanto para la construcción del framework basado en componentes es necesario establecer los elementos de software que conforman a dicha arquitectura, los mismos son ilustrados a continuación.

Componente Abstracto: De este componente heredaran todos los componentes creados por el usuario, será el responsable de proveer a los mismos de una única estructura definida por el sistema.

Componente entorno de trabajo: Este será el componente responsable de gestionar un entorno de trabajo en el cual los componentes puedan ser cargados y se comuniquen entre sí.

Componente Controlador componente: Este será el componente responsable de proveer al sistema de las funcionalidades básicas para operar sobre los componentes.

Componente Fábrica de Conexión: Este componente será el responsable de gestionar las conexiones entre los componentes.

Componente Responsable GUI: Este será el responsable de proveer al componente Entorno de Trabajo de todos los elementos gráficos que necesite el mismo para brindar a los usuarios un entorno de trabajo.

Componente Controlador de XML: Este es el responsable de gestionar los datos de los archivos XML de los componentes que se encuentren cargados en el entorno de trabajo.

Figura 2 Arquitectura del framework basado en componentes

3.7.2. Estilos Arquitectónicos

Definen las reglas generales de organización en términos de un patrón y las restricciones en la forma y la estructura de un grupo numeroso y variado de sistemas de software. Un estilo determina el vocabulario de componentes y relaciones que pueden ser utilizados en instancias de este estilo, con un conjunto de restricciones en las descripciones arquitectónicas (Reynoso, y otros, 2004).

El Estilo de Llamada y Retorno permite al diseñador de software construir una estructura de programa relativamente fácil de modificar y ajustar a escala. Se basa en la abstracción de procedimientos, funciones y métodos utilizados en grandes sistemas de software. Persigue la escalabilidad y modificabilidad.

La arquitectura basada en componentes consiste en una rama de la Ingeniería de Software en la cual se trata con énfasis la descomposición del software en componentes funcionales. Esta descomposición permite convertir componentes pre-existentes en piezas más grandes de software. Dicha arquitectura está encaminada a la reutilización de código mediante un diseño centrado en interfaces y componentes. En el desarrollo de los sistemas de software que siguen este estilo, una vez definidas las interfaces de comunicación de los componentes, se pueden realizar las actualizaciones e incorporación de funcionalidades de manera fácil, pues los componentes funcionan como cajas negras de las que solo se necesita saber la funcionalidad que realizan. Para efectuar cualquier cambio en su modo de funcionamiento solo es necesario reescribir el código interno de la implementación (Gil, José A., 2011).

La Arquitectura Orientada a Objetos describe el uso de objetos que contienen los datos y el comportamiento para trabajar con esos datos; y además tiene un rol o responsabilidad distinta. Hace hincapié en la reutilización a través de la encapsulación, la modularidad, el polimorfismo y la herencia (de la Torre, y otros, 2010).

Se aplica como subestilo del Estilo de Llamada y Retorno la arquitectura basada en componentes, pues es necesario establecer una estructura que represente al sistema a desarrollar desde todas sus perspectivas. Específicamente se utiliza estaya que se pretende desarrollar un sistema que permita la creación de componentes con una estructura definida, así como un entorno de trabajo que posibilite que dichos componentes sean cargados e interactúen entre sí.

Como otro subestilo del estilo en cuestión se aplica la arquitectura Orientada a Objetos debido a que es necesario definir objetos que permitan la interacción con otros en función de realizar múltiples tareas. Para la realización del marco de trabajo es necesario reducir operaciones complejas mediante generalizaciones que mantengan las características de las operaciones. Es necesario contar con objetos que hereden de otros para utilizar las funcionalidades de estos objetos bases o redefinir dicha funcionalidad en función de implementar un nuevo comportamiento. Todo esto con el objetivo de hacer el trabajo de los programadores más fácil y entendible.

3.7.3. Patrones

Un patrón es una pareja de problema/solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas (Larman).

Patrones de Diseño.

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Poseen una serie de elementos como: un nombre, el problema que indica cuando aplicar el patrón y la solución que es la descripción abstracta del problema.

Con el propósito de desarrollar el framework basado en componentes se tuvieron en cuenta los patrones GRASP (Patrones Generales de Software para Asignación de Responsabilidades), debido a que describen los principios fundamentales de la asignación de responsabilidades a objetos expresados en forma de patrones. Se utilizó el patrón Experto debido a que plantea que se debe asignar la responsabilidad al experto en información, que en este contexto sería la clase que cuenta con la información necesaria para cumplir la responsabilidad (Larman).

El framework basado en componentes debe contar con las clases: AbstractComponent que es la encargada de definir una estructura por la cual se debe regir los componentes creados por el usuario y Application que es la encargada de establecer áreas de trabajo para llevar a cabo la gestión de componentes.

Se empleó el patrón Creador, el cual se encarga de asignar a una clase la responsabilidad de crear una instancia de otra. Es de gran importancia saber asignar la responsabilidad de crear instancias de una clase sólo a aquella que la requiera. En el caso del framework basado en componentes la clase Application es la responsable de crear instancias de las clases: DataController, ComponentsView, ComponentManage, ConnectionFactory y contiene una lista de los componentes cargados.

Otro patrón utilizado es el Alta Cohesión ya que consiste en asignar una responsabilidad de modo que la cohesión siga siendo alta. La cohesión es la medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Grady Booch señala que "se da una alta cohesión funcional cuando los elementos de un componente (clase, por ejemplo) colaboran para producir algún comportamiento bien definido" (Larman).

Se emplea debido a que las clases que componen al framework de componentes poseen funcionalidades moderadas y colaboran con otras clases para llevar a cabo las tareas. Se pone de manifiesto en la clase Application; la cual cumple solamente con sus responsabilidades y cuenta con la colaboración de las clases ComponentManage, ConnectionFactory, ComponentsView, Connect, DataController, WorkArea y la AbstractComponent para realizar las tareas relacionadas con los componentes.

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras; "muchas otras" depende del contexto (Larman).

El patrón Bajo Acoplamiento tiene como propósito evitar la dependencia excesiva entre las clases del sistema, garantizando que un cambio en una clase no provoque grandes cambios en otras. En el caso del framework basado en componentes se evidencia un bajo acoplamiento con respecto a las aplicaciones y componentes creados por el usuario, ya que solo dependerán de la clase Application y AbstractComponent respectivamente.

Otro patrón que se aplica es el Controlador ya que durante el diseño es necesario contar con una clase controlador; la cual tiene la responsabilidad de manejar los eventos del sistema. Generalmente los eventos del sistema son generados por actores externos. En este caso la clase que juega el papel de Controlador es Application, pues posee operaciones centralizadas que manejan las acciones que se generan en el sistema.

Además, se utilizan los patrones GoF (Gang of Four). Los patrones de diseño creacionales se centran en resolver problemas acerca de cómo crear instancias de las clases de una determinada aplicación. Dentro de los mismos se utiliza el Instancia Única debido a que restringe la creación de objetos pertenecientes a una clase a un único objeto. Esto garantiza que una clase solo contenga una instancia y un punto de acceso global a la misma. En el caso del framework de componentes se evidencia en la clase Application, pues garantiza que las aplicaciones creadas por el usuario contengan una única instancia de tipo Application.

Los Patrones de diseño estructurales son los que plantean las relaciones entre clases, las combinan y forman estructuras mayores. Tratan de conseguir que los cambios en los requisitos del sistema no ocasionen cambios en las relaciones entre los objetos. Dentro de los mismos se utiliza el Fachada debido a que simplifica el acceso a un conjunto de clases proporcionando una única clase que los programadores

utilicen para comunicarse con dicho conjunto de clases. En el contexto del marco de trabajo la clase Application funciona como fachada, ya que a través de ella el usuario puede acceder a las funcionalidades de las demás clases con las que esta se relaciona sin necesidad de que el usuario acceda directamente a las mismas.

Los patrones de comportamiento caracterizan las formas en las que interactúan y reparten responsabilidades las distintas clases u objetos. Un comando es un objeto función en su estado más puro: una función que tiene un objeto. AL envolver una función en un objeto puede pasarla a otras funciones u objetos como parámetro, para decirles que realicen esta operación concreta mientras llevan a cabo su petición. Algunas veces es necesario enviar mensaje a un objeto sin conocer el selector del mensaje ni el objeto receptor (García, 2003).

En el ámbito del diseño del framework de componentes se utiliza el patrón de diseño Comando ya que los componentes funcionan de forma autónoma; puesto que un componente puede estar emitiendo una señal sin importar quien la vaya a recibir. Por otra parte el componente que recibe la señal solamente recibe la misma sin preocuparse cuál es el componente emisor de la misma.

Interfaz es otro patrón utilizado para el diseño del framework y se pone de manifiesto en la clase AbstractComponent, ya que la misma es una clase virtual pura debido a su implementación en el lenguaje de programación C++, que solamente se encarga de definir los atributos y operaciones que deben poseer los componentes. Todas las clases que quieran utilizar este comportamiento deberán implementar dicha interfaz.

Diagrama de Clases del Diseño

Una clase de diseño es una abstracción de una clase o construcción similar en la implementación del sistema y tienen operaciones, parámetros, atributos y tipos que son necesarios para su implementación en el lenguaje de programación elegido. El Diagrama de Clases de Diseño representa un nivel de detalle alto, pues se relaciona con el lenguaje de programación del cual se hará uso en la implementación del sistema (Carvajal, 2010).

Realizar el diseño de las aplicaciones utilizando la metodología XP no requiere de la representación de diagramas de clases utilizando notación UML, en su lugar se usan las Tarjetas CRC. Aunque no se utilicen dichos diagramas se realizará el Diagrama de Clases del Diseño con el propósito de servir de guía en función de lograr un nivel más elevado de detalle del sistema a desarrollar.

Figura 3 Diagrama de Clases del Diseño asociado al framework basado en componentes

3.8. Tarjetas CRC.

Aunque en general el diseño es realizado por los propios desarrolladores en ocasiones se reúnen aquellos con más experiencia o incluso se involucra al cliente para diseñar las partes más complejas. En estas reuniones se emplean un tipo de tarjetas denominadas CRC (*Class, Responsabilities and Collaborator* - Clase, Responsabilidad y Colaborador) cuyo objetivo es facilitar la comunicación y documentar los resultados. Para cada clase identificada se rellenará una tarjeta de este tipo, se especificará su finalidad y las clases con las que interactúa. Las tarjetas CRC son una buena forma de cambiar de la programación estructurada a una filosofía orientada a objetos.

A continuación se presentan las correspondientes a las clases AbstractComponent y Application, las restantes se pueden encontrar en los anexos de la presente investigación **(Ver Anexo 3)**.

Tabla 13 Tarjeta CRC ‘‘Application’’

Tarjeta CRC	
Clase: Application	
Responsabilidades: es la clase controladora que se encarga de gestionar un área de trabajo donde los componentes se puedan cargar, funcionar de forma autónoma y comunicar entre sí.	**Colaboraciones:** • AbstractComponent • Connect • ComponentManage • DataController • ComponentsView • WorkArea • ConnectionFactory

Tabla 14 Tarjeta CRC ‘‘AbstractComponent’’

Tarjeta CRC	
Clase: AbstractComponent	
Responsabilidades: define la estructura de los componentes que serán creados utilizando el framework de componentes, la misma referencia a un objeto de la clase ComponentData para que el usuario pueda acceder a los atributos de dicha clase.	**Colaboraciones:** • ComponentData

3.9. Conclusiones Parciales

Una vez identificadas las necesidades existentes en el departamento y con ello las características que debe poseer el framework basado en componentes se concluye que se debe realizar una buena planificación para obtener un producto con calidad que responda a las necesidades del cliente en el menor tiempo posible. Por tanto para automatizar los procesos que intervienen en la realización del sistema se

identificaron 4 historias de usuario. Luego de la estimación de cada historia de usuario se obtuvo la cantidad de iteraciones necesarias para llevar a cabo la implementación de las mismas y el tiempo de duración para cada iteración. Finalmente se obtuvieron 3 iteraciones; por lo que a la primera fueron asignadas las historias de usuario de prioridad alta, a la segunda la correspondiente a Cargar componente y a la tercera la historia de usuario Comunicar componentes. Las iteraciones fueron organizadas para que existieran tres momentos de liberación en los que se obtuvieran las características del sistema completamente funcionales y listas para ser probadas en el entorno de trabajo del cliente. La construcción de cualquier sistema debe estar respaldada por un buen diseño arquitectónico, por lo que se concluye que la aplicación del estilo de Llamada y Retorno garantizó que el framework basado en componentes sea escalable y modificable, posibilitando que pueda ser mejorado en futuras versiones. La simplicidad que garantiza la realización de las tarjetas CRC permitió que puedan ser modificadas con facilidad frente a posibles cambios o actualizaciones con respecto al diseño. Por tanto fueron identificadas 11 tarjetas CRC en las cuales se describieron las clases que intervienen en el proceso y las colaboraciones entre ellas. Aunque la metodología XP no exige la utilización de diagramas UML su empleo brindó un nivel más alto de detalle con relación a las piezas de software que componen el sistema a codificar. Por tanto se concluye que la realización de un buen análisis y diseño da paso una implementación más organizada, sencilla y entendible del sistema.

CAPÍTULO 4. IMPLEMENTACIÓN Y PRUEBAS

Introducción

Una vez realizado el proceso de análisis y diseño del sistema se procede a su implementación, la cual está guiada por las tareas que componen a cada historia de usuario descrita en la fase de exploración. El presente capítulo está dividido en dos secciones: Implementación y Pruebas. En la primera sección se realiza la implementación de las clases y objetos correspondientes a la solución propuesta en capítulos anteriores, contando para ello con la realización de las tareas de ingeniería y el establecimiento de una serie de estándares de codificación **(Ver Anexo 5)**. En la segunda sección se describen las pruebas a las que fue sometido el framework de componentes en cada iteración; en función de identificar y corregir fallos cometidos durante el desarrollo de las HU.

4.1. Tareas de ingeniería por HU

Una vez definidas las HU el equipo de desarrollo divide cada una de ellas en una serie de tareas que contribuyan al desarrollo de las mismas. Las tareas pueden ser descritas en un lenguaje técnico que no necesariamente garantice el entendimiento del cliente. A continuación se muestran las correspondientes a las HU de prioridad alta, las restantes se muestran en los anexos de la presente investigación **(Ver Anexo 2)**.

Tabla 15 Tarea 1 de la HU Crear componente

Tarea	
Número tarea: 1	**Número historia:** 1
Nombre tarea: Definir la estructura de un componente.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 7 semanas
Fecha inicio: 29/01/2013	**Fecha fin:** 6/02/2013
Programador responsable: Adrián Martínez	
Descripción: todos los componentes creados por el usuario deben heredar de la clase AbstractComponent, por lo que la misma posee una serie de métodos virtuales puros que el usuario deberá implementar una vez creado el componente, permitiendo dar una estructura al mismo.	

Tabla 16 Tarea 1 de la HU Crear aplicación

Tarea	
Número tarea: 1	**Número historia:** 2
Nombre tarea: Crear menú.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.4 semanas
Fecha inicio: 8/02/2013	**Fecha fin:** 10/02/2013
Programador responsable: Adrián Martínez	
Descripción: esta tarea se encarga proveer al sistema de un menú a través del cual sea posible interactuar con los componentes. En el mismo se muestran el nombre de los componentes cargados que están activos y sus correspondientes acciones, también posibilita que el usuario tenga acceso a un área de trabajo para activar y desactivar los componentes. Este menú puede ser modificado por el usuario. La clase ComponentsView es la que controla todo este proceso.	

Tabla 17 Tarea 2 de la HU Crear aplicación

Tarea	
Número tarea: 2	**Número historia:** 2
Nombre tarea: Crear área para componentes.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.6 semanas
Fecha inicio: 12/02/2013	**Fecha fin:** 15/02/2013
Programador responsable: Adrián Martínez	
Descripción: provee al sistema un área donde serán ubicados aquellos componentes que sean cargados por la aplicación y que a su vez posean una interfaz gráfica de usuario. La clase DockWidget es la clase responsable de controlar el proceso relacionado con la creación de las áreas.	

Tabla 18 Tarea 3 de la HU Crear aplicación

Tarea	
Número tarea: 3	**Número historia:** 2
Nombre tarea: Crear área para activar y desactivar componentes y sus acciones.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.6 semanas
Fecha inicio: 17/02/2013	**Fecha fin:** 21/02/2013
Programador responsable: Adrián Martínez	
Descripción: el sistema provee un área a la cual se accede a través del menú, en la que se muestran los componentes con interfaces gráficas cargados. Además permite que el usuario active o desactive un componente determinado y sus correspondientes acciones.	

4.2. Pruebas de software

Las pruebas de software son un elemento crítico para la garantía de la calidad de software y representan una revisión final de las especificaciones del diseño y de la codificación (Pressman, 2012). Constituyen una actividad en la cual un sistema o componente es ejecutado bajo condiciones o requisitos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente (Booch, y otros, 2004).

Una vez finalizado el desarrollo de un sistema se procede a realizarle diferentes tipos de pruebas en función de garantizar que cumpla con los requisitos planteados por los clientes y que los posibles errores sean corregidos antes de ser desplegado. La metodología XP propone un modelo inverso, en el que lo primero que se escribe son las pruebas que el sistema debe pasar. Luego, el desarrollo debe ser el mínimo necesario para pasar las pruebas previamente definidas. Las pruebas a las que se hace referencia son las pruebas unitarias, realizadas por los desarrolladores. Las mismas se realizan al comienzo para condicionar o dirigir el proceso de desarrollo.

4.2.1. Pruebas Unitarias

Las pruebas unitarias son una de las piedras angulares de XP. Todos los módulos deben pasar las pruebas unitarias antes de ser liberados o publicados. Por otra parte, como se mencionó anteriormente,

las pruebas deben ser definidas antes de realizar el código. Que todo código liberado pase correctamente las pruebas unitarias es lo que habilita que funcione la propiedad colectiva del código. En este sentido, el sistema y el conjunto de pruebas deben ser guardados junto con el código, para que pueda ser utilizado por otros desarrolladores, en caso de tener que corregir, cambiar o recodificar parte del mismo.

Pruebas de Caja Blanca

La prueba de caja blanca, denominada a veces prueba de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, el ingeniero del software puede obtener casos de prueba que garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo, ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa; ejecuten todos los ciclos en sus límites y con sus límites operacionales, y ejerciten las estructuras internas de datos para asegurar su validez (Wiley, 2001).

La prueba del camino básico es una técnica de prueba de Caja Blanca que consiste en derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el grafo de flujo asociado al código y se calcula su complejidad ciclomática. Se identifican los nodos y según las instrucciones serán las aristas que conecten a esos nodos que son los que permiten dibujar el grafo de flujo. Se selecciona el método de camino básico como prueba de caja blanca ya que este permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizarán que durante la prueba se ejecute por lo menos una vez cada sentencia del programa.

El procedimiento para llevar a cabo la técnica de prueba del camino básico es el siguiente:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo de flujo.
3. Se determina el conjunto básico de caminos independientes.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Cálculo de la Complejidad Ciclomática (V (G))

La complejidad ciclomática de un grafo de flujo se puede calcular de 3 formas diferentes:

V (G)= número de regiones.

V (G)=E-N+2; donde E es el número de aristas del y N el número de nodos del grafo de flujo.

V (G)=P+1; donde P es el número de nodos predicados incluidos en el grafo de flujo. Un nodo predicado es aquel que representa una condición, por lo que de él salen dos o más caminos.

Casos de Pruebas

Los Casos de pruebas definen un conjunto específico de entradas de pruebas, ejecución de condiciones y resultados esperados. Luego de la elaboración de los Grafos de Flujos y los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se escogen los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino.

Funcionalidad loadComponent

Figura 4 Codificación de la funcionalidad loadComponent

Figura 5 Grafo de Flujo asociado a la funcionalidad loadComponent

Tabla 19 Caminos básicos de funcionalidad loadComponent

No.	Camino
1	1-2-3-4-9
2	1-2-3-5-6-9
3	1-2-7-8-9

Una vez obtenidos los caminos básicos asociados al Grafo de Flujo de la funcionalidad en cuestión se procede a la creación de casos de pruebas por cada camino. A continuación se muestra el caso de prueba asociado al camino 1-2-3-4-9, los restantes se pueden encontrar en los anexos de este trabajo **(Ver Anexo 4)**.

Tabla 20 Camino básico No.1 de la funcionalidad loadComponent

Caso de prueba para el camino básico No.1	
Camino	1-2-3-4-9
Descripción: una vez que obtiene el componente verifica que haya sido cargado correctamente y que no haya sido cargado con anterioridad. En caso de que no haya sido cargado antes activa las acciones del componente y lo añade a una lista de componentes cargados.	

Entrada	Nombre de un componente.
Resultado esperado	Se cargó el componente correctamente.
Resultado obtenido	Se cargó el componente correctamente.

Funcionalidad connectComponents

Figura 6 Codificación de la funcionalidad connectComponents

Figura 7 Grafo de Flujo asociado a la funcionalidad connectComponents

Tabla 21 Caminos básicos de la funcionalidad connectComponents

No.	Camino
1	1-2-3-4-5-6-7-8-11
2	1-2-3-4-11
3	1-2-3-4-5-11
4	1-2-3-4-5-6-7-9-10-11
5	1-2-11

A continuación se muestra el caso de prueba asociado al camino 1-2-3-4-5-6-7-8-11, los restantes se pueden encontrar en los anexos de este trabajo **(Ver Anexo 5)**.

Tabla 22 Camino básico No.1 de la funcionalidad connectComponents

Caso de prueba para el camino básico No.1	
Camino	1-2-3-4-5-6-7-8-11
Descripción: una vez declaradas las variables necesarias para el desarrollo de la funcionalidad en cuestión se recorre la lista de componentes con conexión y se procede a la construcción de las conexiones del componente. Se verifica que exista conexión, en caso de existir se procede a recorrer la lista de conexiones con el propósito de asignar valores a los componentes emisor y receptor. Se verifica que el componente emisor haya sido cargado y en caso de que esto se cumpla se realiza la conexión.	
Entrada	Lista de componentes con conexión y lista de componentes cargados.
Resultado esperado	Se logró la comunicación entre componentes.
Resultado obtenido	Se logró la comunicación entre componentes.

4.2.2. Pruebas de Aceptación

Son una serie de pruebas que escribe y realiza el cliente, son llevadas a cabo después que el usuario ha definido sus necesidades respecto a lo que desea que realice la aplicación. Son básicamente pruebas

funcionales sobre el sistema completo que se deben realizar en intervalos regulares, y busca validar los resultados obtenidos (Joyanes, 2006).

Las pruebas de aceptación fueron creadas en base a las historias de usuario en cada ciclo de la iteración del desarrollo. El cliente fue el encargado de especificar los diversos casos de prueba para comprobar que las historias de usuario fueron implementadas correctamente, por tanto no se pueden considerar terminadas hasta tanto pasen correctamente todas las pruebas de aceptación. Dichas pruebas no se realizan durante el desarrollo pues serían impresentables al cliente, se realizan sobre el producto terminado, una versión del producto o una iteración pactada previamente con el cliente.

Conociendo esto al finalizar la primera iteración del proceso de desarrollo, se realizaron las pruebas de aceptación correspondientes. En la primera iteración de pruebas se detectaron dos no conformidades asociadas a la HU Crear componente, por lo que fue necesario realizar otra iteración para corregir las no conformidades, lográndose esto en la segunda iteración.

Una vez culminada la segunda iteración se realizaron las pruebas de aceptación asociadas. En la primera iteración de prueba se detectó una no conformidad, por lo que fue necesario hacer otra iteración para corregir dicha no conformidad, lográndose que un componente fuera cargado de forma satisfactoria.

Terminada la tercera iteración del proceso de desarrollo se realizaron las pruebas de aceptación correspondientes, en la primera iteración de la prueba se detectó una no conformidad, lo que conllevó a realizar otra iteración para corregir la misma, lográndose en la segunda iteración que los componentes pudieran comunicarse correctamente. A continuación se muestran los casos de pruebas asociados a las HU identificadas en la fase de Exploración.

Tabla 23 Primera iteración de prueba-HU Crear aplicación

Caso de Prueba de Aceptación
Código Caso de Prueba: FC-1-1-1 **HU:** Crear aplicación.
Nombre de la persona que realiza la prueba: Ing. Reynier Pupo Gómez
Descripción de la Prueba: Dirigida a probar que la aplicación sea creada correctamente.
Condiciones de Ejecución: Incluir las librerías de cabecera del framework así como las bibliotecas del mismo.
Entrada / Pasos de ejecución: Crear un Proyecto de tipo Aplicación.

El proyecto creado debe contar con un puntero a un objeto de la clase Application de la siguiente forma: Application a(argc, argv); Una vez creado el puntero se accede a la funcionalidad del mismo GetMainWindow, haciendo uso de la siguiente sentencia: a.GetMainWindow()->show();
Resultado Esperado: Correcta creación de la aplicación
Evaluación de la Prueba: Satisfactoria.

Tabla 24 Primera iteración de prueba- HU Crear componente

Caso de Prueba de Aceptación	
Código Caso de Prueba: FC-1-1-2	**HU:** Crear componente.
Nombre de la persona que realiza la prueba: Ing. Reynier Pupo Gómez	
Descripción de la Prueba: Dirigida a probar que los componentes sean creados de forma correcta.	
Condiciones de Ejecución: Incluir las librerías de cabecera del framework así como las bibliotecas del mismo.	

Entrada / Pasos de ejecución:
El proyecto creado hereda de la clase AbstractComponent con la siguiente sentencia:
class Componente_1: public AbstractComponent
Se define una estructura para el Componente, implementando los métodos virtuales puros de la clase padre con las siguientes sentencias:
public:
QWidget* GetWidget();
void SetNombreComponente(QString nombre);
QString GetNombreComponente();
QList<QAction*> GetListaAccion();
void SetListaAccion(QList<QAction*> list);
DatosComponente1* GetDatosComponete();
 	
private:
QWidget *WidPrincipalUser;	

Exportar el Componente. Q_EXPORT_PLUGIN2(EchoPlugin, EchoPlugin)
Resultado Esperado: Correcta creación del componente
Evaluación de la Prueba: Insatisfactoria.

Tabla 25 Segunda iteración de prueba-HU Crear componente

Caso de Prueba de Aceptación	
Código Caso de Prueba: FC-1-2-1	**HU:** Crear componente.
Nombre de la persona que realiza la prueba: Ing. Reynier Pupo Gómez	
Descripción de la Prueba: Dirigida a probar que los componentes sean creados de forma correcta.	
Condiciones de Ejecución: Incluir las librerías de cabecera del framework así como las bibliotecas del mismo.	

Entrada / Pasos de ejecución:
Crear un Proyecto de tipo Componente.
El proyecto creado hereda de la clase AbstractComponent con la siguiente sentencia:
class Componente_1: public AbstractComponent
Se define una estructura para el Componente, implementando los métodos virtuales puros de la clase padre con las siguientes sentencias:
public:
QWidget* getWidget();
void setNameOfComponent(QString name);
QString getNameOfComponent();
QList<QAction*> getListOfActions();
ComponentData* getComponentData();
void setListActions(QList<QAction*> list);
private:
ComponentData *objComponentData;
QWidget *WidPrincipalUser;
 	
Exportar el Componente.	

Q_EXPORT_PLUGIN2(EchoPlugin, EchoPlugin)
Resultado Esperado: Correcta creación del componente
Evaluación de la Prueba: Satisfactoria.

Tabla 26 Primera iteración de prueba-HU Cargar componente

Caso de Prueba de Aceptación	
Código Caso de Prueba: FC-2-1-1	**HU:** Cargar componente.
Nombre de la persona que realiza la prueba: Ing. Reynier Pupo Gómez	
Descripción de la Prueba: Dirigida a que el componente sea cargado de forma correcta.	
Condiciones de Ejecución: Incluir las librerías de cabecera del framework así como las bibliotecas del mismo. Es necesario contar con una aplicación desarrollada sobre el framework de componentes, con su respectivo archivo XML.	
Entrada / Pasos de ejecución: Definir en el archivo XML correspondiente a la aplicación el nombre del componente que se desea cargar. El sistema procede a cargar en memoria el componente especificado en el XML de la aplicación con la siguiente funcionalidad: loadComponent(QString name) En caso de que no se haya cargado y sea un componente con una interfaz gráfica de usuario el mismo es ubicado en el área de trabajo que se especifica en su archivo XML, con la siguiente sentencia: locateComponent(component,NameOfComponent);	
Resultado Esperado: El componente fue cargado correctamente	
Evaluación de la Prueba: Insatisfactoria.	

Tabla 27 Segunda iteración de prueba-HU Cargar componente

Caso de Prueba de Aceptación	
Código Caso de Prueba: FC-2-2-1	**HU:** Cargar componente.
Nombre de la persona que realiza la prueba: Ing. Reynier Pupo Gómez	
Descripción de la Prueba: Dirigida a que el componente sea cargado de forma correcta.	
Condiciones de Ejecución: Incluir las librerías de cabecera del framework así como las bibliotecas del mismo. Es necesario contar con una aplicación desarrollada sobre el framework de componentes, con su	

respectivo archivo XML.
Entrada / Pasos de ejecución: Definir en el archivo XML correspondiente a la aplicación el nombre del componente que se desea cargar. El sistema procede a cargar en memoria el componente especificado en el XML de la aplicación, con la siguiente funcionalidad: loadComponent(QString name) El sistema verifica que el componente no se haya cargado con anterioridad, con la siguiente sentencia: componentExists(componentLoadList,name) En caso de que no se haya cargado y sea un componente con una interfaz gráfica de usuario el mismo es ubicado en el área de trabajo que se especifica en su archivo XML, con la siguiente sentencia: locateComponent(component,NameOfComponent);
Resultado Esperado: El componente fue cargado correctamente
Evaluación de la Prueba: Satisfactoria.

Tabla 28 Primera iteración de prueba-HU Comunicar componentes

Caso de Prueba de Aceptación	
Código Caso de Prueba: FC-3-1-1	**HU:** Comunicar componentes.
Nombre de la persona que realiza la prueba: Ing. Reynier Pupo Gómez	
Descripción de la Prueba: Dirigida a probar que los componentes sean correctamente conectados.	
Condiciones de Ejecución: Incluir las librerías de cabecera del framework así como las bibliotecas del mismo. Es necesario contar con una aplicación desarrollada sobre el framework de componentes, con su respectivo archivo XML. Se debe contar con al menos dos componentes cargados.	
Entrada / Pasos de ejecución: Definir en el Archivo XML del componente receptor la sentencia de conexión. El sistema lista todos los componentes con conexión, con la siguiente sentencia: listOfComponentsConnection(componentLoadList); Una vez obtenido el listado de sentencias, el sistema construye la conexión para cada componente listado y la devuelve en una lista de conexiones, para esto utiliza la funcionalidad siguiente: buildConnection(QString compoName). Obtenida la lista de conexiones el sistema procede a realizar la conexión de cada uno de los elementos	

de la lista, utilizando la funcionalidad: connectComponents(AbstractComponent *compoT, AbstractComponent *compoR)
Resultado Esperado: Correcta comunicación entre componentes.
Evaluación de la Prueba: Insatisfactoria.

Tabla 29 Segunda iteración de prueba-HU Comunicar componentes

Caso de Prueba de Aceptación	
Código Caso de Prueba: FC-3-2-1	**HU:** Comunicar componentes.
Nombre de la persona que realiza la prueba: Ing. Reynier Pupo Gómez	
Descripción de la Prueba: Dirigida a probar que los componentes sean correctamente conectados.	
Condiciones de Ejecución: Incluir las librerías de cabecera del framework así como las bibliotecas del mismo. Es necesario contar con una aplicación desarrollada sobre el framework de componentes, con su respectivo archivo XML. Se debe contar con al menos dos componentes cargados.	
Entrada / Pasos de ejecución: Definir en el Archivo XML del componente receptor la sentencia de conexión. El sistema lista todos los componentes con conexión, con la siguiente sentencia: listOfComponentsConnection(componentLoadList); El sistema busca las sentencias de conexión de los componentes y las devuelve en una lista, valiéndose para esto de la funcionalidad: searchConnectionSentence(QString compoName) Una vez obtenido el listado de sentencias, el sistema construye la conexión para cada componente listado y la devuelve en una lista de conexiones, para ello utiliza la funcionalidad siguiente: buildConnection(QString compoName). Obtenida la lista de conexiones el sistema procede a realizar la conexión de cada uno de los elementos de la lista, utilizando la funcionalidad: connectComponents(AbstractComponent *compoT, AbstractComponent *compoR)	
Resultado Esperado: Correcta comunicación entre componentes.	
Evaluación de la Prueba: Satisfactoria.	

4.3. Conclusiones Parciales.

Con la realización de este capítulo se concluye que la descomposición de las historias de usuario en tareas garantizó que la codificación de las mismas fuera más precisa y entendible por parte de los programadores responsables. La codificación guiada por estándares; específicamente definidos para el lenguaje de programación C++ sobre el IDE Qt-Creator conllevó a que el equipo de desarrollo tuviera bien definido un estilo común de programación y permitió que se llevara a cabo la propiedad colectiva del código, siendo uno de los aspectos claves de la metodología XP. Una vez implementada la propuesta de solución se procedió a aplicarle las pruebas de software permitiendo verificar la calidad del producto desarrollado y comprobar si verdaderamente cumple con las características esperadas por el cliente. Aplicar la técnica de prueba de Caja Blanca para las pruebas unitarias requirió del conocimiento total de la estructura interna del programa. Por tanto una vez recorridos los caminos independientes del Grafo de Flujo asociado a las principales funcionalidades del sistema se concluye que los resultados obtenidos se corresponden con los esperados por el equipo de desarrollo. Con la realización de las pruebas de aceptación a las historias de usuario se obtuvieron cuatro no conformidades que contribuyeron a la mejora continua del producto, las mismas fueron corregidas en iteraciones de pruebas siguientes arrojando resultados satisfactorios para el cliente.

CONCLUSIONES GENERALES

Con la culminación del presente trabajo de diploma se logró dar cumplimiento a las tareas de la investigación propuestas al inicio de la investigación, obteniéndose un framework basado en componentes para el departamento de Señales Digitales, lo que arribó a las siguientes conclusiones:

- El análisis de aplicaciones de escritorio que operaran sobre el paradigma de programación basado en componentes permitió identificar clases y funcionalidades que posteriormente fueron implementadas en el framework.
- Para el cumplimiento de los objetivos y en correspondencia con las necesidades expuestas por el cliente, se requirió hacer un estudio de la situación actual del desarrollo de software en el departamento de Señales Digitales con el propósito de elevar la eficiencia del proceso en cuestión.
- La utilización de la metodología XP garantizó un ahorro considerable de tiempo y esfuerzo en cuanto a la realización de los procesos referentes a la ingeniería y codificación del sistema.
- El uso de estilos arquitectónicos y patrones de diseño brindó como resultado una estructura organizacional que respondiera a un diseño solido y flexible para la codificación del sistema.
- Los resultados arrojados por las pruebas realizadas cumplieron con las expectativas del equipo de desarrollo, demostrando la calidad de las funcionalidades implementadas.
- Se implementó un sistema factible que responde a las necesidades existentes en el departamento de Señales Digitales.

RECOMENDACIONES

- Extender la solución al centro GEySED con el objetivo de que sea utilizado por los proyectos que lo integran y de esta forma someterla a pruebas más rigurosas.
- Agregar una funcionalidad que permita la interacción con los componentes a través de la web; facilitando que los mismos puedas ser manejados tanto por aplicaciones de web como de escritorio.
- Establecer reglas de comunicación a través de la red que permitan que los componentes provenientes de sistemas y estaciones de trabajo ubicadas en los diferentes proyectos del Centro puedan comunicarse entre sí.

BIBLIOGRAFÍA REFERENCIADA

Booch, Grady, Rumbaugh, James and Jacobson, Ivar. 2004.*El Proceso Unificado de Desarrollo de Software.* La Habana,Cuba : s.n., 2004.

Briñes, Rengifo, Fernanda, Johanna and Betancourt, Carlos Alberto. 2011.*FRAMEWORKS Y HERRAMIENTAS PARA EL DESARROLLO DE APLICACIONES ORIENTADAS A DISPOSITIVOS MÓVILES.* 2011.

Carvajal, Erlys. 2010.*Análisis y diseño del subsistema de Análisis de Resultados de un Simulador de Procesos Químicos.* La Habana : s.n., 2010.

Coral. 2011. Coral - Lightweight C++ component framework. *Coral - Lightweight C++ component framework.* [Online] 2011. http://libcoral.org/.

de la Torre, César, Zorilla Castro, Unai and Ramos Barros, Miguel Angel. 2010.*Guía de Arquitectura en N-Capas orientada al Dominio con .NET 4.0.* 2010.

Díaz, Javier F., Queiruga, Claudia A. and Iuliano, Pablo J.*Incorporando seguridad a las componentes de interfaz de usuario del framework JSF (JAVA Server Faces).*

Eckel, Bruce. 2012. Pensar en C++ (Volumen 2). 2012.

Fuentes, Lidia, Troya, José M. and Vallecillo, Antonio.*Desarrollo de Software Basado en Componentes.*

García, Francisco José and Laguna, Miguel Angel.*Construcción de frameworks basada en análisis de conceptos formales y soportada por Mecanos.* Valladolid : s.n.

García, Jesús. 2003.*Análisis y Diseño de Software.Patrones de Diseño.* 2003.

Gil, José A. 2011. Arquitectura Basada en Componentes. [Online] diciembre 4, 2011. http://www.scribd.com/doc/14704374/.

Guérin, Brice Arnaud.*C++. Programación Unix & Windows Standar Template Library . Creación de un programa archivado.*

Headquarters, Company. 2010. Sitio Web de Visual Paradigm. *Sitio Web de Visual Paradigm.* [Online] Febrero 19, 2010. http://www.visual-paradigm.com/product/vpuml/.

Joskowicz, José;. 2008.*Reglas y Prácticas en Extreme Programming.* 2008.

Krutchen, Philippe. 2000. Rational Unified Process. 2000.

Larman, Craig.*UML y PATRONES. Introducción al análisis y diseño orientado a objetos.*

Letelier, Patricio and Penadés, Carmen. 2006.*Métodologías ágiles para el desarrollo de software: Extreme Programming (XP).* Valencia : s.n., 2006.

Martínez, León Alberto. 2011.*Implementación de un editor gráfico de circuitos eléctricos con Qt.* Cartagena : s.n., 2011.

Montilva C., Juan A, Arapé, Nelson and Colmenares, Juan Andrés. 2003.*Desarrollo de Software basado en Componentes.* Mérida : s.n., 2003.

Ortín, María José, et al.*El Modelo del Negocio como base el Modelo de Requisitos 1.* Universidad de Murcia : s.n.

Pérez, M. 1999.*Arquitectura para Ambientes CASE Integrados.* 1999.

Pressman, Roger S. 2012. Software Engineering. A partitioner´s Approach. 2012.

Reynoso, Carlos and Kiccillof, Nicolás. 2004.*Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004.

Rumbaugh, James, Jacobson, Ivar and Booch, Grady.*El Lenguaje Unificado de Modelado.*

Saavedra, Esteban.*Catalyst: Framework para el desarrollo de aplicaciones Web.*

Szyperski, Clemens. 2002. Component Software: Beyond Object Oriented Programming. 2002.

VCreate Logic. 2012.*Ayuda de GCF 2.6.0.* 2012.

Viñolo, Raydel Raúl and Roquero Figueroa, Alexander. 2012.*Sistema Gestor de Procesos de Media v2.* 2012.

Wiley, John. 2001.*Software Engineering Standards.* 2001.

Zapata, Carlos Mario and González, Guillermo. 2009. Revisión de la literatura en interoperabilidad entre sistemas heterogéneos de software. [Online] Mayo 2009. http://www.scielo.org.co/scielo.php.

BIBLIOGRAFÍA CONSULTADA

Arizas, Marible and Molina, Juan Carlos. 2004. *INTRODUCCIÓN Y PRINCIPIOS BÁSICOS DEL DESARROLLO DE DE SOFTWARE BASADO EN COMPONENTES.* 2004.

Booch, Grady, Rumbaugh, James and Jacobson, Ivar. 2004. *El Proceso Unificado de Desarrollo de Software.* La Habana,Cuba : s.n., 2004.

Briñes, Rengifo, Fernanda, Johanna and Betancourt, Carlos Alberto. 2011. *FRAMEWORKS Y HERRAMIENTAS PARA EL DESARROLLO DE APLICACIONES ORIENTADAS A DISPOSITIVOS MÓVILES.* 2011.

Carvajal, Erlys. 2010. *Análisis y diseño del subsistema de Análisis de Resultados de un Simulador de Procesos Químicos.* La Habana : s.n., 2010.

Coral. 2011. Coral - Lightweight C++ component framework. *Coral - Lightweight C++ component framework.* [Online] 2011. http://libcoral.org/.

Córcoles, Jose Eduardo and Lozano, María Dolores. *MODELADO DE APLICACIONES CON UML.* Castilla : s.n.

de la Torre, César, Zorilla Castro, Unai and Ramos Barros, Miguel Angel. 2010. *Guía de Arquitectura en N-Capas orientada al Dominio con .NET 4.0.* 2010.

Díaz, Javier F., Queiruga, Claudia A. and Iuliano, Pablo J. *Incorporando seguridad a las componentes de interfaz de usuario del framework JSF (JAVA Server Faces).*

Eckel, Bruce. 2012. Pensar en C++ (Volumen 2). 2012.

Fuentes, Lidia, Troya, José M. and Vallecillo, Antonio. *Desarrollo de Software Basado en Componentes.*

García, Francisco José and Laguna, Miguel Angel. *Construcción de frameworks basada en análisis de conceptos formales y soportada por Mecanos.* Valladolid : s.n.

García, Jesús. 2003. *Análisis y Diseño de Software.Patrones de Diseño.* 2003.

Gil, José A. 2011. Arquitectura Basada en Componentes. [Online] diciembre 4, 2011. http://www.scribd.com/doc/14704374/.

Guérin, Brice Arnaud. *C++. Programación Unix & Windows Standar Template Library . Creación de un programa archivado.*

Headquarters, Company. 2010. Sitio Web de Visual Paradigm. *Sitio Web de Visual Paradigm.* [Online] Febrero 19, 2010. http://www.visual-paradigm.com/product/vpuml/.

Hernández, Rolando Alfredo and Coello, Sayda. 2011. *EL PROCESO DE INVESTIGACIÓN CIENTÍFICA.* Cuba : Universiataria Cubana, 2011.

Joskowicz, José. 2008. *Reglas y Prácticas en Extreme Programming.* 2008.

KENDALL, KENNETH. E. and KENDALL, JULIE E. 2005. *ANÁLISIS Y DISEÑO DE SISTEMAS.* México : s.n., 2005.

Krutchen, Philippe. 2000. Rational Unified Process. 2000.

—. 2000. *The Rational Unified Process: An Introduction.* 2000.

Larman, Craig. *UML y PATRONES. Introducción al análisis y diseño orientado a objetos.*

Letelier, Patricio and Penadés, Carmen. 2006. *Métodologías ágiles para el desarrollo de software: Extreme Programming (XP).* Valencia : s.n., 2006.

Martínez, León Alberto. 2011. *Implementación de un editor gráfico de circuitos eléctricos con Qt.* Cartagena : s.n., 2011.

Montilva C., Juan A, Arapé, Nelson and Colmenares, Juan Andrés. 2003. *Desarrollo de Software basado en Componentes.* Mérida : s.n., 2003.

Ortín, María José, et al. *El Modelo del Negocio como base el Modelo de Requisitos 1.* Universidad de Murcia : s.n.

Pérez, M. 2002. *Arquitectura para Ambientes CASE Integrados.* 2002.

Pressman, Roger S. 2012. Software Engineering. A partitioner´s Approach. 2012.

Reynoso, Carlos and Kiccillof, Nicolás. 2004. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004.

Rumbaugh, James, Jacobson, Ivar and Booch, Grady. *El Lenguaje Unificado de Modelado.*

Saavedra, Esteban. *Catalyst: Framework para el desarrollo de aplicaciones Web.*

Sommerville, Ian ;. 2005. *Ingeniería del software.* Madrid : s.n., 2005.

Szyperski, Clemens. 2002. Component Software: Beyond Object Oriented Programming. 2002.

VCreate Logic. 2012. *Ayuda de GCF 2.6.0.* 2012.

Vignaga, Andrés and Perovich, Daniel. *Enfoque Metodológico para el Desarrollo Basado en Componentes.*

Viñolo, Raydel Raúl and Roquero Figueroa, Alexander. 2012. *Sistema Gestor de Procesos de Media v2.* 2012.

Weitzenfeld, Alfredo. *Ingeniería de Software orientada a objetos con UML.JAVA e Internet.*

Wiley, John. 2001. *Software Engineering Standards.* 2001.

Zapata, Carlos Mario and González, Guillermo. 2009. Revisión de la literatura en interoperabilidad entre sistemas heterogéneos de software. [Online] Mayo 2009. http://www.scielo.org.co/scielo.php.

ANEXOS

Anexo 1. Tareas de ingeniería de las HU que presentan prioridad media

Tabla 30 Tarea 1 de la HU Cargar componente

Tarea	
Número tarea: 1	**Número historia:** 3
Nombre tarea: Leer del archivo XML de la aplicación.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.6 semanas
Fecha inicio: 23/02/2013	**Fecha fin:** 26/02/2013
Programador responsable: Adrián Martínez	
Descripción: el sistema recorre el archivo XML de la aplicación en busca del nombre del componente.	

Tabla 31 Tarea 2 de la HU Cargar componente

Tarea	
Número tarea: 2	**Número historia:** 3
Nombre tarea: Buscar componente.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.8 semanas
Fecha inicio: 27/02/2013	**Fecha fin:** 3/03/2013
Programador responsable: Adrián Martínez	
Descripción: el sistema busca el componente especificado en el archivo XML de la aplicación y en caso de existir crea una instancia del mismo.	

Tabla 32 Tarea 3 de la HU Cargar componente

Tarea	
Número tarea: 3	**Número historia:** 3
Nombre tarea: Verificar existencia del componente.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.8 semanas
Fecha inicio: 4/03/2013	**Fecha fin:** 8/03/2013
Programador responsable: Adrián Martínez	
Descripción: el sistema verifica si el componente ha sido cargado por la aplicación con anterioridad. En caso de que ya haya sido cargado anteriormente el sistema notifica que un componente solo puede ser cargado por la aplicación una sola vez y en caso contrario procede a cargarlo.	

Tabla 33 Tarea 4 de la HU Cargar componente

Tarea	
Número tarea: 4	**Número historia:** 3
Nombre tarea: Ubicar componente	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.8 semana
Fecha inicio: 9/03/2013	**Fecha fin:** 13/03/2013
Programador responsable: Adrián Martínez	
Descripción: destacar que solo son ubicados aquellos componentes que posean una interfaz gráfica de usuario. El sistema lee del archivo XML del componente con el propósito de indicar en qué área de la aplicación debe ser ubicado el componente. En caso de que el área especificada esté ocupada por otro componente en la aplicación es creada una nueva área y ubicado en la misma el componente.	

Tabla 34 Tarea 1 de la HU Comunicar componentes

Tarea	
Número tarea: 1	**Número historia:** 4
Nombre tarea: Listar componentes con conexión.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.8 semanas
Fecha inicio: 15/03/2013	**Fecha fin:** 20/03/2013
Programador responsable: Adrián Martínez – Yunet Gasca	
Descripción: recorre la lista de componentes cargados en busca de componentes que establezcan conexión y devuelve una lista de los mismos. Para conocer si un componente posee conexión es necesario acceder a su archivo XML y verificar que la misma exista.	

Tabla 35 Tarea 2 de la HU Comunicar componentes

Tarea	
Número tarea: 2	**Número historia:** 4
Nombre tarea: Crear conexiones.	
Tipo de tarea : Desarrollo	**Puntos estimados:** 0.6 semanas
Fecha inicio: 21/03/2013	**Fecha fin:** 24/03/2013
Programador responsable: Adrián Martínez – Yunet Gasca	
Descripción: dado un componente con conexión el sistema busca en su archivo XML su sentencia de conexión, crea la conexión y devuelve las conexiones correspondientes al componente en cuestión.	

Anexo 2. Tarjetas CRC.

Tabla 36 Tarjeta CRC ‘‘ComponentsView’’

Tarjeta CRC	
Clase: ComponentsView	
Responsabilidades: se encarga de crear los elementos con interfaces gráficas de usuario de la aplicación. Ya sean el menú, las áreas donde serán ubicados los componentes con interfaz gráfica de usuario que han sido cargados y un área para la activación y desactivación de los componentes cargados con sus correspondientes acciones.	**Colaboraciones:** • AbstractComponent • WorkArea • ComponentManage

Tabla 37 Tarjeta CRC ‘‘ComponentManage’’

Tarjeta CRC	
Clase: ComponentManage	
Responsabilidades: provee funcionalidades básicas como: buscar un componente; eliminar un componente en caso de que exista. Es la clase responsable de cargar los componentes.	**Colaboraciones:** • AbstractComponent

Anexo 3. Casos de pruebas para los caminos básicos asociados a la funcionalidad loadComponent

Tabla 38 Camino básico No.2 de la funcionalidad loadComponent

Caso de prueba para el camino básico No.2	
Camino	1-2-3-5-6-9
Descripción: una vez que obtiene el componente verifica que haya sido cargado correctamente y que no haya sido cargado con anterioridad. En caso de que haya sido cargado antes el sistema muestra un mensaje de error.	
Entrada	Nombre de un componente.

Resultado esperado	El sistema muestra un mensaje de error.
Resultado obtenido	El sistema muestra un mensaje de error.

Anexo 5. Casos de pruebas para los caminos básicos asociados a la funcionalidad connectComponents

Tabla 39 Camino básico No.2 de la funcionalidad connectComponents

Caso de prueba para el camino básico No.2	
Camino	1-2-3-4-11
Descripción: una vez declaradas las variables necesarias para el desarrollo de la funcionalidad en cuestión se recorre la lista de componentes con conexión y se procede a la construcción de las conexiones del componente. En caso de no existir ninguna conexión entre componentes no se realiza la comunicación entre componentes.	
Entrada	Lista de componentes con conexión y lista de componentes cargados.
Resultado esperado	No se logró la comunicación entre componentes.
Resultado obtenido	No se logró la comunicación entre componentes.

Tabla 40 Camino básico No.3 de la funcionalidad connectComponents

Caso de prueba para el camino básico No.3	
Camino	1-2-3-4-5-11
Descripción: una vez declaradas las variables necesarias para el desarrollo de la funcionalidad en cuestión se recorre la lista de componentes con conexión y se procede a la construcción de las conexiones del componente. Se verifica que existan conexiones entre componentes, en caso de no existir no se establece la comunicación entre componentes.	
Entrada	Lista de componentes con conexión y lista de componentes cargados.
Resultado esperado	No se logró la comunicación entre componentes.
Resultado obtenido	No se logró la comunicación entre componentes.

Tabla 41 Camino básico No.4 de la funcionalidad connectComponents

Caso de prueba para el camino básico No.4	
Camino	1-2-11
Descripción: una vez declaradas las variables necesarias para el desarrollo de la funcionalidad en cuestión se recorre la lista de componentes con conexión, en caso de que no existan componentes con conexión no se establece la comunicación entre componentes.	
Entrada	Lista de componentes con conexión y lista de componentes cargados.
Resultado esperado	No se logró la comunicación entre componentes.
Resultado obtenido	No se logró la comunicación entre componentes.

Tabla 42 Camino básico No.5 de la funcionalidad connectComponents

Caso de prueba para el camino básico No.5	
Camino	1-2-3-4-5-6-7-9-10-11
Descripción: una vez declaradas las variables necesarias para el desarrollo de la funcionalidad en cuestión se recorre la lista de componentes con conexión y se procede a la construcción de las conexiones del componente. Se verifica que exista conexión, en caso de existir se procede a recorrer la lista de conexiones con el propósito de asignar valores a los componentes emisor y receptor. Se verifica que el componente emisor haya sido cargado, en caso contrario el sistema muestra un mensaje de error y no se establece la comunicación entre componentes.	
Entrada	Lista de componentes con conexión y lista de componentes cargados.
Resultado esperado	No se logró la comunicación entre componentes.
Resultado obtenido	No se logró la comunicación entre componentes.