Universidad de las Ciencias Informáticas

Facultad 3

Extensión del IDE Netbeans para el desarrollo de aplicaciones empleando el marco de trabajo Sauxe.

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

Autor: Dayron Limonta Gutiérrez

Tutor: Ing. René R. Bauta Camejo

Ciudad de la Habana, junio 2012

"Año 54 de la Revolución"

DECLARACIÓN DE AUTORÍA

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

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

_____	_____
Dayron Limonta Gutiérrez	Ing. René R. Bauta Camejo
Autor	Tutor

OPINIÓN DEL USUARIO DEL TRABAJO DE DIPLOMA

El Trabajo de Diploma, titulado Módulo Decodificación, fue realizado en la Universidad de las Ciencias Informáticas. Esta entidad considera que, en correspondencia con los objetivos trazados, el trabajo realizado le satisface

- □ Totalmente
- □ Parcialmente en un ____ %

Los resultados de este Trabajo de Diploma le reportan a esta entidad los beneficios siguientes (cuantificar):

_____

_____

_____

Como resultado de la implantación de este trabajo se reportará un efecto económico que asciende a:

Y para que así conste, se firma la presente a los ____ días del mes de _____ del año _____.

_____	_____
Representante de la entidad	Cargo
_____	_____
Firma	Cuño

OPINIÓN DEL TUTOR DEL TRABAJO DE DIPLOMA

Título: Extensión del IDE Netbeans para el desarrollo de aplicaciones empleando el marco de trabajo Sauxe.

Autor: Dayron Limonta Gutiérrez

El tutor del presente Trabajo de Diploma considera que durante su ejecución el estudiante mostró las cualidades que a continuación se detallan.

Por todo lo anteriormente expresado considero que el estudiante está apto para ejercer como Ingeniero en Ciencias Informáticas; y proponemos que se le otorgue al Trabajo de Diploma la calificación de _____.

_____ _____

Ing. René R. Bauta Camejo Fecha

RESUMEN

Sauxe contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. Estos componentes se desarrollan de forma manual, esto implica que se tengan que reescribir códigos en varios ficheros que nunca varían y que requieren de especial atención para su funcionamiento. Esta forma de crear la estructura básica de los componentes genera errores que se traducen en pérdidas de tiempo y atraso en los cronogramas de desarrollo. Además influye de forma negativa en la estandarización, extensión, seguridad y mantenimiento de los componentes que se implementan utilizando Sauxe.

El presente trabajo propone la implementación de una extensión para el IDE Netbeans la cual facilita y estandariza el proceso de generación de un componente utilizando el marco de trabajo Sauxe. Se abordan además temas relacionados con la validación de algunos artefactos del proceso de desarrollo y con las pruebas realizadas a la solución.

Palabras claves: componentes, Netbeans, Sauxe.

ÍNDICE

INTRODUCCIÓN

La Industria Cubana del Software (ICSW) está llamada a convertirse en una significativa fuente de ingresos para el país, como resultado del correcto aprovechamiento de las ventajas del alto capital humano disponible. Por estas razones, existen diferentes entidades que se dedican a la producción de software, una de ellas es la Universidad de las Ciencias Informáticas (UCI).

El Centro de Informatización para la Gestión de Entidades (CEIGE) es uno de los centros productivos que se encarga del desarrollo de software en la UCI haciendo uso de herramientas y tecnologías que facilitan esta tarea. Entre estos software se encuentra el marco de trabajo Sauxe. Este contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo (23).

Sauxe define una estructura básica y un conjunto de ficheros de configuración que permiten el correcto funcionamiento de los componentes creados en el. Actualmente este proceso se desarrolla de forma manual, esto implica que se reescriban códigos en varios ficheros que nunca varían y que requieren de especial atención para evitarla introducción de errores. Algunos de los errores más comunes cometidos por los desarrolladores son el incorrecto nombrado de las clases y acciones dentro de estas, el incorrecto nombrado de los ficheros de configuración y la incorrecta ubicación de directorios y ficheros que inhabilitan el funcionamiento de los componentes.

Atendiendo a la cantidad y variedad de ficheros a generar y los estándares de nomenclatura y codificación definidos, esta forma de crear la estructura básica de los componentes es un proceso largo y engorroso que puede introducir errores que se traducen en pérdidas de tiempo y atraso en los cronogramas de desarrollo. Además influye de forma negativa en la estandarización, extensión, seguridad y mantenimiento de los componentes que se implementan utilizando Sauxe.

Como en todo proceso que se realiza de forma manual hay probabilidades de que se introduzcan errores relacionados con los estándares de nomenclatura y codificación y con la ubicación de los ficheros de configuración y clases que define Sauxe para sus componentes lo que puede afectar las estimaciones de esfuerzo y tiempo hechas para la implementación de las soluciones.

Teniendo en cuenta lo anteriormente expuesto se define como **problema a resolver**: ¿Cómo evitar la introducción de errores por parte de los programadores en la creación de componentes dentro del marco de trabajo Sauxe?

Se define como **objeto de estudio** la generación de componentes y como **campo de acción** la generación de componentes en el marco de trabajo Sauxe.

A partir del problema identificado se define como **objetivo general de la investigación**: Desarrollar una extensión para el IDE Netbeans que permita la generación de componentes en el marco de trabajo Sauxe para disminuir la introducción de errores por parte de los programadores.

Para dar cumplimiento al objetivo general propuesto, se definen los siguientes **objetivos específicos**:

- Realizar el marco teórico de la generación de componentes para identificar posibles puntos de reutilización.
- Analizar y diseñar el plugin para Netbeans.
- Implementar plugin para NetBeans.
- Validar plugin para NetBeans.

La investigación parte de la siguiente **idea a defender:** si se desarrolla una extensión para el IDE Netbeans que permita la generación de componentes en el marco de trabajo Sauxe se logrará disminuir la introducción de errores por parte de los programadores.

Para una mejor comprensión de la investigación la misma se divide en 3 capítulos:

Capítulo 1: En este capítulo se hace un estudio del estado del arte de las herramientas actuales utilizadas para la generación de código. Las tendencias existentes, las técnicas utilizadas, las tecnologías y herramientas empleadas para el análisis, diseño e implementación de la solución.

Capítulo 2: En este capítulo se abordan las características de la propuesta de solución, especificando requisitos funcionales y no funcionales que debe cumplir el sistema, y definiendo el mapa conceptual, así como patrones de diseños más empleados en la solución.

Capítulo 3: En este capítulo se aborda la parte de implementación donde se muestran las dependencias entre las partes del código del sistema (diagrama de componentes), los diagramas de componentes se utilizan para modelar la vista de implementación estática de un sistema. Abordándose además la etapa en la que se realizaron las pruebas al diseño para evaluar la calidad del producto de software desarrollado.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En este capítulo se realiza un estudio acerca del estado actual de la creación de extensiones para el IDE NetBeans. Para brindar una mejor solución a dicha investigación resulta indispensable indagar acerca de los principales conceptos, tecnologías y herramientas relacionadas con el tema. Se definen además, las tecnologías a utilizar en la construcción de la extensión.

1.2. Proceso de creación de componentes en Sauxe

En la ilustración 1 y 2 de los anexos se puede observar los directorios que se crean en la estructura de un componente de Sauxe. Estos son ubicados en las carpetas ¨web¨ y ¨apps¨.

A continuación son creados y ubicados los ficheros en sus respectivos directorios. En las ilustraciones 3 y 4 de los anexos se parte desde la carpeta ¨web¨ y ¨apps¨ en el mismo orden en que son mencionadas, al crear los ficheros se debe agregar un código inicial en algunos de ellos, la parte del código sería igual para los ficheros que forman parte del esqueleto del marco de trabajo Sauxe.

Por último se debe registrar el componente en Acaxia para que este se visualice en el menú. Luego se le asocian funcionalidades que igualmente se registran en Acaxia. Las funcionalidades representan requisitos del componente que se desarrolle. Una vez terminado este proceso es necesario dar permiso al usuario para ver el componente. Esto permite al usuario ejecutar las funcionalidades definidas.

1.3. Principales conceptos

Un **plugin** es un módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande. (29)

Un **plugin** es una aplicación informática que interactuando con otra aplicación le aporta una función específica a ésta última. (40)

Un plugin no es más que un nuevo componente que se añade a un sistema existente aportándole funcionalidades específicas.

Componente no es más que un elemento que forma parte de la composición de un todo. Se trata de elementos que, a través de algún tipo de asociación o contigüidad, dan lugar a un conjunto uniforme. (38)

Paquete de componentes: Conjunto de componentes, que comparten alguna función o se complementan de algún modo y están agrupadas de una determinada manera. (12)

1.4. Ambiente de desarrollo integrado.

IDE:

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, o sea, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. (10)

1.5. Herramientas para la generación de código.

1.5.1. Generadores de código

Es una herramienta capaz de generar código de forma automática. Por lo general generan código en algún lenguaje de tercera generación, listo para compilar o interpretar. Estas herramientas hacen de forma automática el trabajo que a los programadores les tomaría mucho más tiempo lograr de forma manual. Para su trabajo los generadores parten de un modelo que puede ser un diagrama de clases o la estructura de una base de datos. El código generado va a depender del modelo que se le entregue y de los algoritmos y patrones que tenga implementado el generador. La mayoría de estas herramientas poseen plantillas de lenguajes scripts para representar los algoritmos y patrones implementados, las cuales el usuario puede modificar y adaptar según sus necesidades. (21)

1.5.1.1. Softwares generadores de códigos existentes.

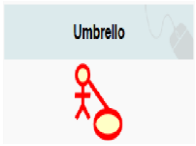
Existe una variedad de herramientas para la generación de código tanto comercial como libre. Los indicadores por los cuales se escogieron estas herramientas es debido a que son de código abierto posibilitando modificar alguna funcionalidad, que estén apoyados por comunidades, que tengan abundante documentación, y que estén entre los más utilizados. A continuación se muestran las herramientas de código abierto que más prestaciones presentan en el mercado.

ArgoUML incluye soporte para todos los estándares UML 1.4 diagramas. Se ejecuta en cualquier plataforma Java y está disponible en diez idiomas. ArgoUML 0.26 y 0.26.2 se han descargado más de 80.000 veces y están en uso en todo el mundo. (31)

Esta herramienta es de código abierto, lo que significa que cualquiera puede tener una

copia gratis del código fuente, cambiarlo y usarlo para nuevos propósitos. La única obligación es que pases el código de la misma forma a otros.

ArgoUML tiene la capacidad de generar código PHP 5. La generación de código la realiza usando la técnica de clases de nombres largos: presentacion_UCCcontrollers_gestionarPrestamo.php. Cuenta con una amplia comunidad y documentación. (32)



Umbrello es una herramienta libre para crear y editar diagramas UML, que ayuda en el proceso del desarrollo de software. Es capaz de generar código en Java , PHP, JavaScript, ActionScript, SQL, Python, Ada, IDL, XML Schema, Perl, C++. El formato de fichero que utiliza está basado en XMI. Es una herramienta case muy utilizada en el mundo, en los 6 primeros meses del año 2012 se han hecho más de 11 mil descargas como lo muestra la figura 24.Su plataforma de ejecución es Linux. Cuenta con una amplia Comunidad y documentación. (2)

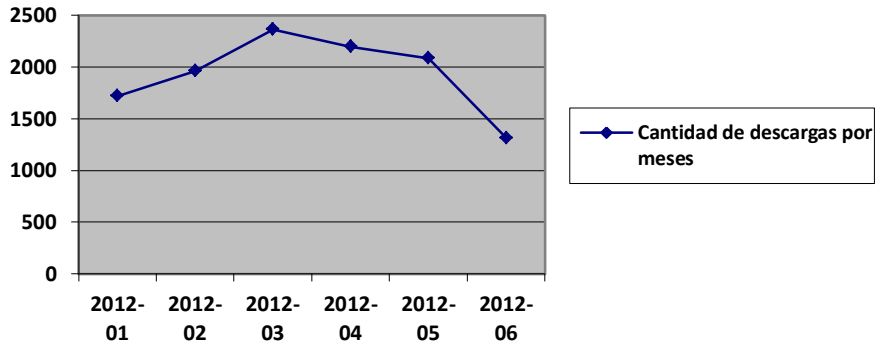


Figura: 24 Descargas del Umbrello desde enero-hasta junio. (2)

Visual paradigm Community Edition en su versión 6.4 es una herramienta Visual para el modelado UML. Está diseñada para ingenieros de software, analistas de sistemas, analistas de negocio, arquitectos y desarrolladores. Está orientada a la creación de diseños usando el paradigma de programación orientada a objetos. Con esta herramienta se puede generar código para lenguajes C#, Java, PHP. (24) Es una herramienta case muy utilizada en el mundo algunos números relacionados con la descarga de varios sitios en internet, cnet desde el 2008 se han hecho descargas 52,536 (41), en software.filestube 395 desde el 2009. (42)

1.6. Marco de trabajo Sauxe.

Es un producto desarrollado sobre tecnologías libres como el lenguaje PHP, gestor de base de datos PostgreSQL, servidor web Apache. Su administración centralizada de todos los aspectos necesarios a tener en cuenta en el desarrollo de aplicaciones de gestión lo convierte en un producto de punta en esta rama. Permite realizar un número de funcionalidades que hace esta arquitectura muy aplicable para cualquier entorno web en PHP. Permite la gestión de multi-entidad y con esta la compartimentación de la información de cada una de ellas.

1.6.1. Estructura de Sauxe

Sauxe está compuesto por varios marcos de trabajo, los cuales serán descritos a continuación, estructurados en niveles o capas como se puede apreciar en las figuras. (21)

Figura 5 Arquitectura de Sauxe. (21)

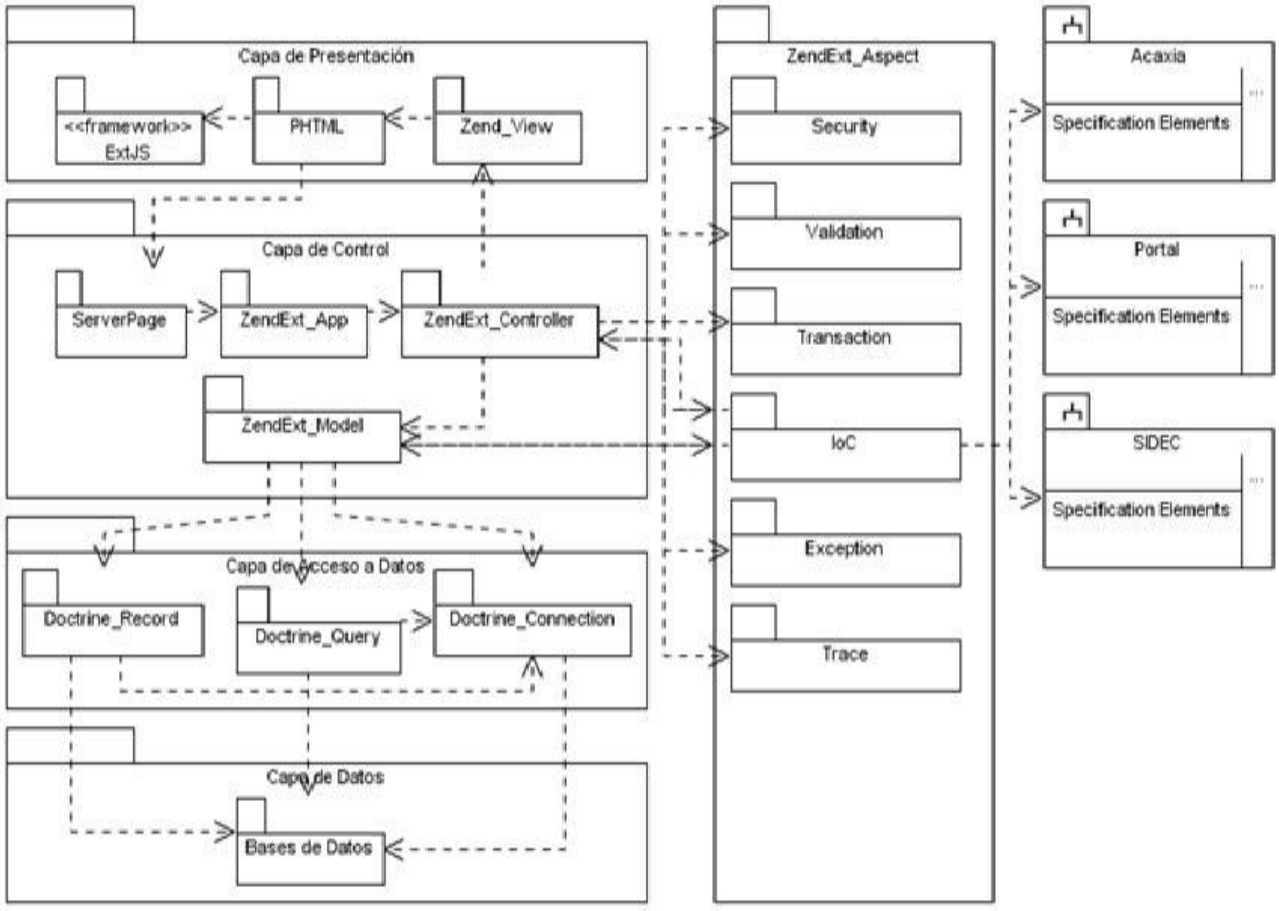


Figura 6 Estructura de Sauxe

1.7. Modelo de Desarrollo propuesto

La propuesta de modelo de desarrollo fue fabricada por el equipo de producción del centro CEIGE teniendo en cuenta los peligros con los que se cuentan en el proyecto. Este modelo está orientado a la reutilización de componentes parametrizables, donde se simplifican las actividades y se eleva el nivel de especialización de los implicados. Dicho modelo está basado en los principios de, buenas prácticas y algunos elementos de las metodologías RUP, fue elaborado teniendo en cuenta las características especiales que presenta la UCI, por ejemplo: casi todos los proyectos están compuestos en su mayoría por estudiantes. (22)

Características

- Centrado en la arquitectura

La arquitectura determina la línea base, los elementos de software estructurales a partir de los elementos de la arquitectura de negocio. Interviene en la gestión de cambios y diseña la evolución e integración del producto. La arquitectura orienta las prioridades del desarrollo y resuelve las necesidades tecnológicas y de soporte para el desarrollo. (22)

- Orientado a componentes

Las iteraciones son orientadas por el nivel de significancia arquitectónicas de los componentes, los mismos son abstracciones arquitectónicas de los procesos de negocio y requisitos asociados que modelan, el componente es la unidad de medición y ordenamiento de las iteraciones. (22)

- Iterativo e incremental

Las iteraciones son planificadas y coordinadas con el equipo de arquitectura, los clientes y la alta gerencia. Cada iteración constituye el desarrollo de componentes, los cuales son integrados al término de la iteración, permitiendo de esta manera la evolución incremental del producto. (22)

- Ágil y adaptable al cambio

El desarrollo de las partes formaliza solamente las características principales de la solución, priorizando los talleres y las comunicaciones entre las personas. Los clientes y funcionales están involucrados en el proyecto y poseen parte de la responsabilidad del éxito del mismo. Los cambios son conciliados semanalmente, discutidos y aprobados. (22)

1.8. Tecnologías y herramientas para el desarrollo.

Se realizara un análisis de las tecnologías y herramientas que se utilizaran en el desarrollo de nuestra solución, se mostrara las características de cada una de ellas más afines con nuestra solución.

1.8.1. Netbeans 6.9

Netbeans es un proyecto de código abierto para desarrolladores de software. Puede obtener todas las herramientas que necesite para crear aplicaciones profesionales para el escritorio, la empresa, la web y equipos móviles con el lenguaje Java, C/C++, y Ruby. NetBeans IDE es fácil de instalar y de uso instantáneo y se ejecuta en varias plataformas incluyendo Windows, Linux y Mac OS X y Solaris. (48).

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de Java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software. (48).

1.8.2. NetBeans Platform

NetBeans Platform es un framework con una amplia variedad de APIs[1] que resuelven gran cantidad de problemas a la hora de construir una aplicación. Él es el corazón sobre el cual se construye, entre otras aplicaciones, NetBeans IDE. (11)

NetBeans Platform hace fuerte hincapié sobre la construcción del software de forma modular, módulo sobre módulo, y es ahí precisamente donde mayor provecho se puede sacar de esta plataforma, puesto que ofrece implementados los mecanismos de descubrimiento de nuevos módulos (y de actualizaciones de los existentes) desde repositorios remotos, resolución de dependencias, activación/desactivación de módulos en caliente, comunicación entre los mismos, etc. permitiendo preocuparse por la lógica y rápidamente desplegar aplicaciones, pudiendo ir extendiendo su funcionalidad a medida que pasa el tiempo. (11)

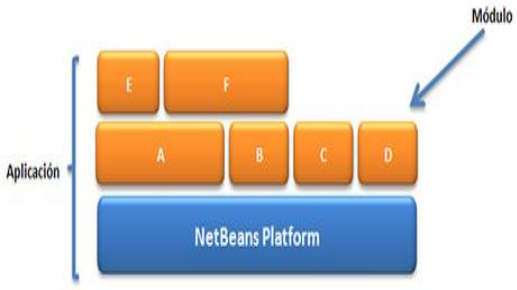


Figura 1: Nuevo módulo agregado a una aplicación ver anexo 5(11)

[1] Interfaz de Programación de Aplicaciones

Una gran ventaja de la construcción modular es que se puede crear una aplicación conformada por x cantidad de módulos diferentes, cada uno responsable de llevar a cabo determinadas responsabilidades, y según el rol de la persona que la va a utilizarla solo se carga en la aplicación los módulos que permiten cumplir con su tarea, permitiendo tener un abanico de aplicaciones sin tener que programar una sola línea de código adicional. Por ejemplo: (11)

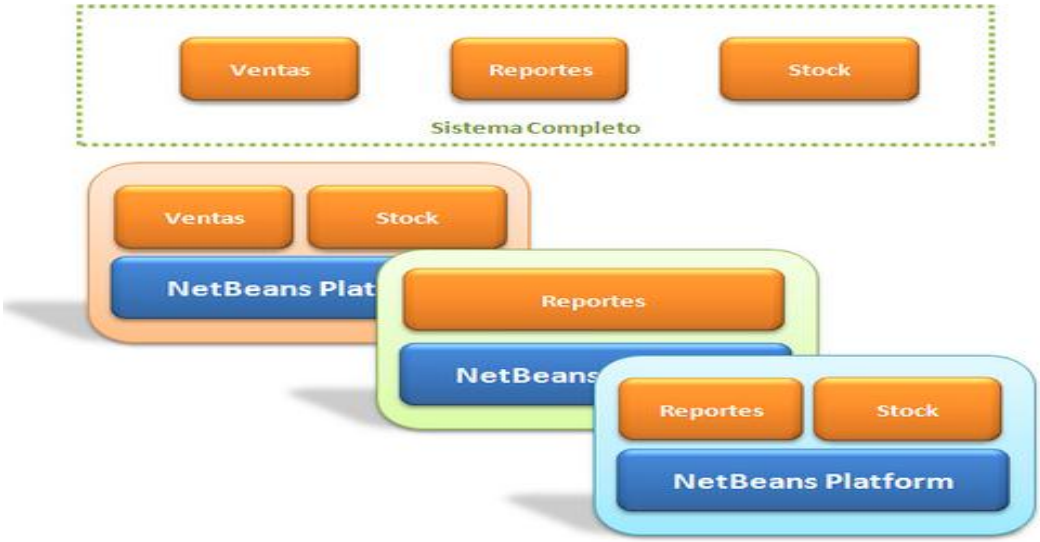


Figura 2: Creación de una aplicación a partir de otros módulos ver anexo 5(11)

Otras características que hacen interesante la elección de NetBeans Platform son las siguientes: (11)

- Los proyectos desarrollados no dejan de ser multiplataforma, y poseen lanzadores para cada plataforma.
- Sistema de ventanas práctico para desarrollar las interfaces de usuario.
- Sistema de ficheros virtual donde se montan los diferentes módulos y se van adaptando automáticamente los menús, barra de herramientas, menús contextuales, de la aplicación desarrollada.
- Su licencia permite construir tanto aplicaciones open source como comerciales.
- Es compatible con Java Web Start.

- No es obligatorio que una aplicación deba tener interfaz de usuario grafica (GUI), ya que la plataforma permite dejar de lado la misma y seguir disfrutando del resto de los beneficio, por ejemplo la actualización de módulos desde un repositorio remoto.
- Soporte completo para desarrollar desde NetBeans IDE, por lo que no necesita otra herramienta adicional para el desarrollo.

Tipos de Proyectos (11)

Existen cuatro tipos de proyectos que se empezaron a crear a partir de la versión 6.1 del IDE:

- Módulo
- Módulo Envoltorio de Librería
- Suite de Módulo
- Aplicación sobre NetBeans Platform

Módulo: (11)

Un módulo es simplemente un típico JAR (Java Archive) con cierta meta información almacenada en el manifiesto. Los módulos poseen la extensión NBM (NetBeans Module). Entre la meta información se encuentra la versión del módulo, las dependencias, descripción de funcionalidad, datos del autor, etc. Un módulo puede ser usado en cualquier proyecto desarrollado sobre NetBeans Platform (siempre que se cumplan las restricciones de dependencias del mismo), incluso sobre NetBeans IDE.

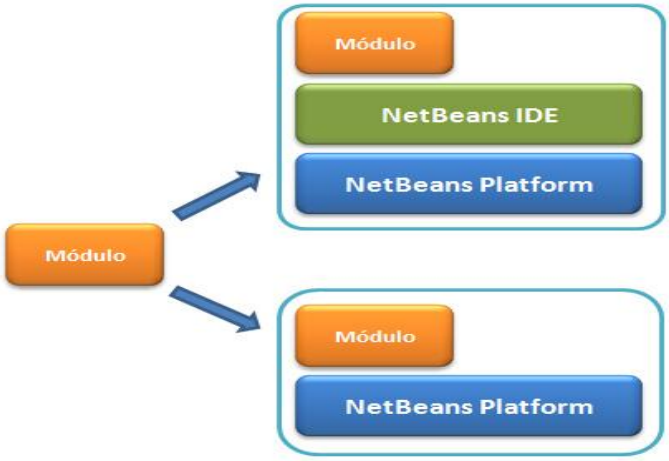


Figura 3: Módulo para el IDE Netbeans o para otra aplicación desarrollada en Netbeans

Módulo Envoltorio de Librería: (11).

Permite que una librería externa (.JAR) sea vista desde los otros módulos como un módulo estándar. Básicamente se comporta igual que un Módulo, pero este no contendrá lógica alguna, simplemente expondrá la interfaz de la librería envuelta.

Figura 4: Módulo Envoltorio de Librería ver anexo 5(11)

Tanto a los Módulos como a los Módulo Envoltorio de Librería se les denomina de forma genérica como módulos o Plugins.

Suite de Módulos (11).

Es una colección de módulos sobre la cual desarrollar la aplicación. Por defecto, una Suite de Módulos trae habilitados todos los módulos que conforman tanto a NetBeans Platform como a NetBeans IDE.

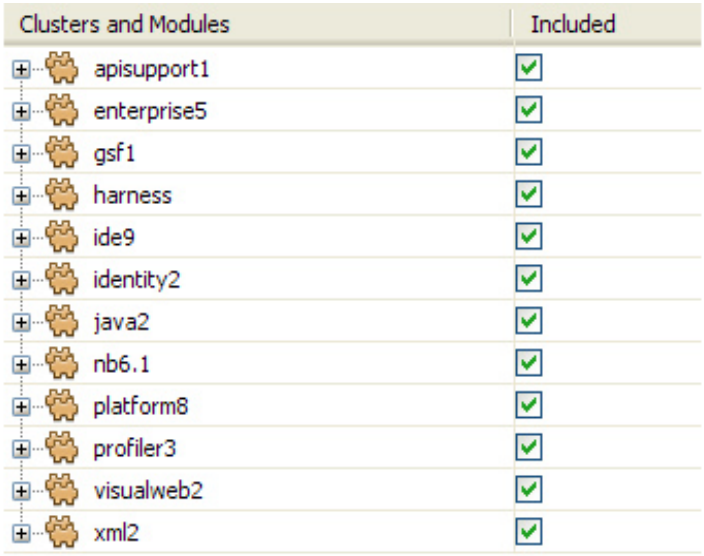


Figura 5: Colección de módulos sobre la cual se desarrollaría la aplicación.ver anexo 5(11)

El trabajo consistiría en deshabilitar los módulos que no se utilicen y agregar los nuevos (Módulos o Módulo Envoltorio de Librería) para de esa forma crear aplicaciones

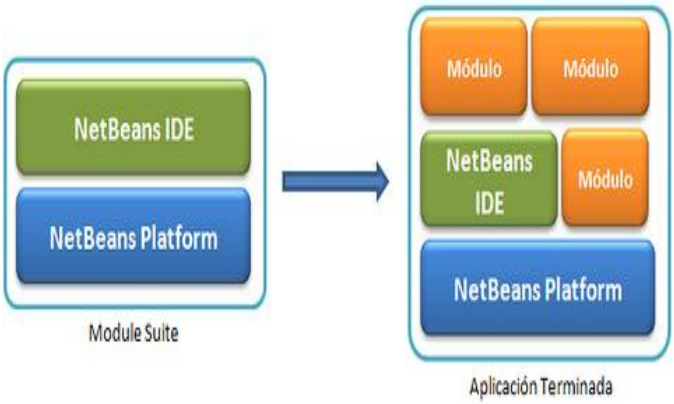


Figura 6: Trabajando con la Suite de Módulos. Ver anexo 5

Aplicación sobre NetBeans Platform. (11)

Básicamente también crea un Module Suite, pero por defecto esta suite aparece configurado con la mínima cantidad de módulos habilitados necesarios para iniciar. Por su configuración inicial lo llamaremos también como “Suite de Módulos vacío”.

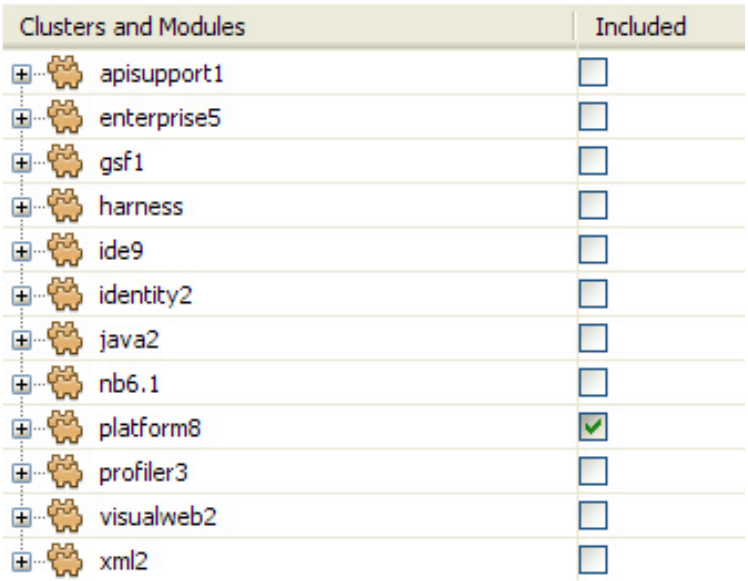


Figura 7: Suite de Módulos vacio ver anexo 5(11)

De forma contraria a un Suite de módulos, el trabajo consistirá en habilitar los módulos que se vayan requiriendo para que los módulos (ya sean Módulos o Módulo Envoltorio de Librería) cumplan sus dependencias, y así crear la aplicación.

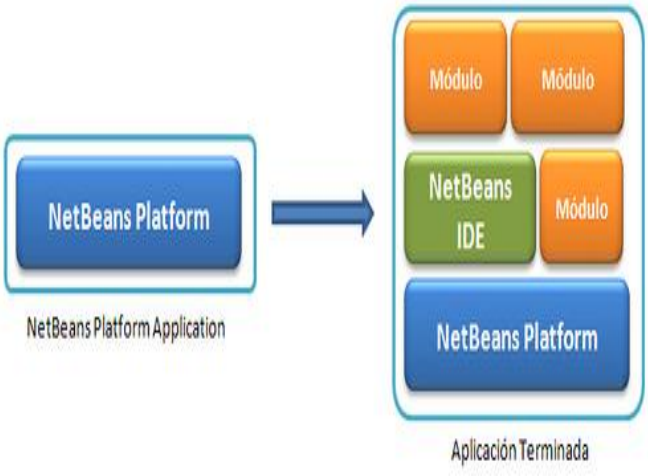


Figura 8: Aplicación de NetBeans plataforma ver anexo 5(11)

Cualquier aplicación que se desarrolle comenzando con un Module Suite puede iniciarse con un NetBeans Platform Application, solo variará la forma de realizarlo.
De ser posible, siempre que se desarrolle una aplicación basada en NetBeans Platform es preferible hacerlo con un NetBeans Platform Application ya que evitará olvidar deshabilitar un módulo que realmente no se utilice. (11)

Netbeans Plataforma permite crear 4 tipos de proyectos de estos se escogió para la realización de la herramienta el módulo, que permite crear archivos java con extensión nbm, estos archivos pueden ser incluidos en cualquier herramienta echa con Netbeans Plataforma.

1.9. Herramientas CASE.

Las herramientas CASE (Ingeniería de Software Asistida por Ordenador por sus siglas en ingles) o de modelado son un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo del sistema de información que permiten el incremento en la velocidad de desarrollo de los sistemas. A los analistas, le permite, tener más tiempo para el análisis y diseño y minimizar el tiempo

para codificar y probar, aumentando de esta forma la productividad, a través de la generación de código y la reutilización de objetos o módulos. (34)

1.9.1. Visual Paradigm.

Herramienta multiplataforma para el modelado UML(Lenguaje Unificado de Modelado por sus siglas en ingles) que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Ayuda a una más rápida construcción de aplicaciones de calidad y a un menor coste.(5)

Características (5)

- Es una potente herramienta CASE para visualizar y diseñar elementos de software, para ello utiliza UML.
- Proporciona a los desarrolladores una plataforma que les permite diseñar un producto con calidad de una forma rápida.
- Facilita la interoperabilidad con otras herramientas CASE como el Rational Rose y se integra con las siguientes herramientas Java: Eclipse/IBM WebSphere, Jbuilder, NetBeans IDE, Oracle Jdeveloper, BEA Weblogic.
- Está disponible en varias ediciones: Enterprise, Professional, Community, Standard, Modeler y Personal.
- Genera código y realiza ingeniería inversa para diez lenguajes de programación, Java, C++, CORBA IDL, PHP,XML Schema y ADA.
- Genera código para C#, Visual Basic.net, Object Definition Lenguage(ODL), Flasch Action Script, Delphi, Perl yPhyton.
- Se integra con el Visio para importar imágenes del mismo para realizar los diagramas de despliegue.
- Genera documentación para el proyecto en HTML, MS Word y PDF. Además exporta e importa los diagramas en el estándar XML y como imágenes (ya sea con extensiones jpg o png).
- Es gratis en su edición Community

1.10. Herramientas de desarrollo colaborativo

Se hará uso de las herramientas, control de versiones Subversión y su plataforma visual RapidSVN, estas permiten tener siempre asegurada los últimos cambios del plugin para Netbeans en desarrollo.

1.10.1. Subversión

Se basa en un repositorio central que actúa como un servidor de ficheros, con la capacidad de recordar todos los cambios que se hacen tanto en sus directorios como en sus ficheros. (28)

Características: (34)

- Mantiene versiones no sólo de archivos, sino también de directorios
- Es un sistema de control de versiones libre bajo una licencia de tipo Apache/BSD (Distribución de Software Berkeley) y se le conoce también como SVN por ser ese el nombre de la herramienta de línea de comandos.
- Es un sistema general que se puede utilizar para administrar cualquier conjunto de ficheros.
- Los archivos versionados no tienen un número de revisión independiente.
- Mejor uso del ancho de banda, ya que en las transacciones se transmiten sólo las diferencias y no los archivos completos.
- Mayor eficiencia en la creación de ramas y etiquetas que en CVS.
- Realiza el seguimiento de las versiones de proyectos completos.
- Realiza el seguimiento del código modular (un archivo que se reutiliza, o se comparte, en varios proyectos).

1.10.2. RapidSVN.

Es una plataforma visual para el sistema Subversión escrito en C++, con código abierto y software libre bajo la licencia GNU General Public License (GPL) versión 3. Está disponible en varios idiomas diferentes y actúa de cliente gráfico para el acceso al repositorio SVN, tanto si éste es remoto como si es local. Es de fácil manejo para los usuarios principiantes pero lo suficientemente potente y con herramientas

interesantes para los usuarios avanzados. El programa funciona en cualquier plataforma y se puede ejecutar en Linux, Windows, Mac OS / X, Solaris, etc.

Los dos software analizados se seleccionaron debido a las características que presentan, las cuales son afines con el proyecto a desarrollar. El SVN es uno de los sistemas de control de versiones más utilizados actualmente y el RapidSVN es uno de los clientes más completos con se cuenta para su uso. (26)

1.10.3. Lenguajes de Modelado.

Se denomina lenguaje de modelado de objetos al conjunto estandarizado de símbolos y las distintas combinaciones de ellos para modelar un diseño de software. (34)

UML.

Es un Lenguaje de Modelado Unificado basado en una notación gráfica la cual permite: especificar, construir, visualizar y documentar los objetos de un sistema programado. Este modelado visual es independiente del lenguaje de implementación, de tal forma que los diseños realizados usando UML se puedan implementar en cualquier lenguaje que soporte las posibilidades de UML (principalmente lenguajes orientados a objetos). El lenguaje de modelado unificado (UML) es una de las herramientas de modelado más utilizadas para la definición de la arquitectura de una aplicación que utiliza una gran variedad de diagramas (clases, secuencias, actividades, componentes, etc.).(34)

1.10.4. Lenguaje de desarrollo.

Java.

Java es un lenguaje de programación orientado a objetos, que permite a los programadores realizar aplicaciones de múltiples tipos, ya sean de escritorio o web. Se caracteriza por ser un lenguaje simple, robusto y poderoso que se torna fácil de aprender, debido a que elimina sentencias de bajo nivel además del Garbage Collector (en español: Recolector de Basura) haciendo transparente para los programadores el manejo de la memoria. Se destaca por ser un lenguaje de código abierto, multiplataforma por lo cual ha logrado una gran expansión por todo el mundo. En la actualidad incluye un gran número de librerías para múltiples trabajos como: el trabajo con la red, tratamiento de excepciones, hilos para el procesamiento concurrente entre otras. (8)

XML Lenguaje de Marcado Extensible

Este lenguaje se ha presentado como sucesor de HTML como lenguaje para presentación de contenidos en Internet. Pero XML es mucho más: es un metalenguaje que sirve para describir nuevos lenguajes cada cual adaptado a un grupo de contenidos especial. Así, con XML se pueden crear lenguajes para documentos que describan noticias, o ecuaciones, o dibujos. (12)

La información con que cuenta el plugin para NetBeans está guardada en varios xmls es por esto que se escoge el lenguaje XML, algunas características de java como ser un lenguaje multiplataforma, abundante documentación, de código y por ser el lenguaje donde el desarrollador tiene más experiencia.

1.11. Conclusiones parciales

El estudio del estado del arte de los temas relacionados con la investigación permitió identificar conceptos fundamentales que sentaron las bases para el desarrollo de la misma.

El estudio realizado sobre el marco de trabajo Sauxe permitió conocer cuáles son los pasos a seguir para la generación de componentes y las ventajas que proporcionaría contar con una herramienta que permitiera realizar este proceso.

El estudio de los principales generadores de código permitió identificar que estos no podía ser usados porque no generan la estructura base de los componentes de Sauxe y el código que generan no cumple con los estándares de nomenclatura y codificación definidos por el marco de trabajo Sauxe.

Para el desarrollo de la solución se emplearán como herramientas las definidas por el Departamento para el proceso de desarrollo.

CAPÍTULO 2: CARACTERÍSTICAS DE LA EXTENSION

2.1 Introducción

En el presente capítulo se realiza el análisis y diseño de la extensión del IDE NetBeans. Se exponen los artefactos generados para estos flujos de trabajo según el modelo de desarrollo definido para el Departamento de Tecnología que constituyen las entradas para la fase de implementación.

2.2 Modelo de dominio

El primer paso en el proceso de desarrollo de software es precisamente alcanzar cierto nivel de conocimientos sobre el problema en cuestión.

En el presente trabajo de diploma se intenta capturar los tipos más importantes de objetos que existen o los eventos que suceden en el entorno donde estará el sistema, por lo que decide realizar un modelo de dominio. Para conformar dicho modelo se ha elaborado un modelo conceptual que contiene los objetos y conceptos que se enmarcan en la problemática expuesta.

2.2.1 Modelo Conceptual

El Modelo Conceptual es una representación visual de los conceptos u objetos del mundo real, significativos para un problema o área de interés. El objetivo de la creación de este artefacto es aumentar la comprensión del problema y contribuir a esclarecer la terminología o nomenclatura del dominio. (23)

El modelo conceptual se representa usando el lenguaje de modelado (UML) como se muestra en la Figura 22, donde se observan conceptos del dominio y sus relaciones y atributos.

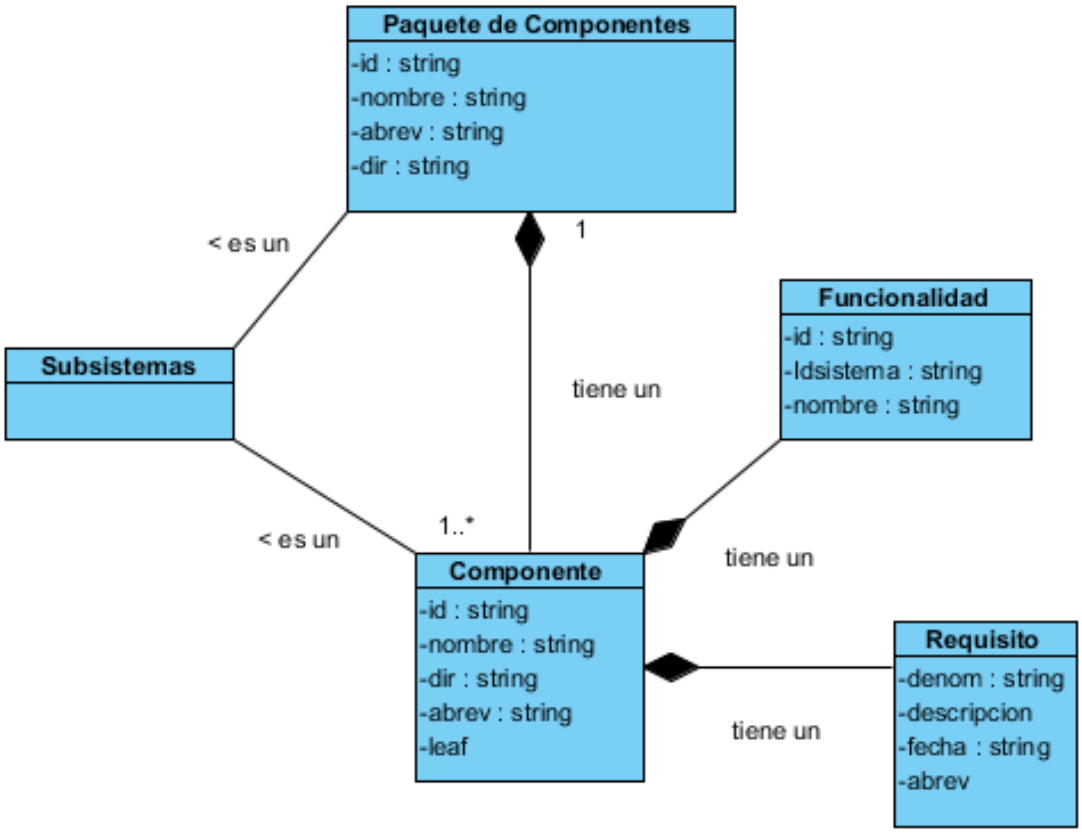


Figura:12 Modelo Conceptual

2.1 Técnicas de captura de requisitos empleadas.

La técnica que se utilizó fue Tormenta de ideas para aplicarla se hizo una reunión con varios programadores experimentados en el uso del marco de trabajo Sauxe, primeramente se definieron los objetivos y el modelador para el desarrollo de la reunión, este sería el que guiara la reunión, como resultado se identificaron 15 requisitos formando 5 agrupaciones de requisitos esta son gestionar subsistemas, gestionar componentes, gestionar paquete de componente .gestionar requisitos y gestionar funcionalidades.

2.2 Requisitos Funcionales

Los requisitos funcionales denotan una funcionalidad del sistema, son característica requerida que expresan una capacidad de acción del mismo o una funcionalidad; generalmente expresada en una declaración en forma verbal. (36)

Sirve de base para estimar el coste, el tiempo necesario para desarrollar el sistema y para planificar los contenidos técnicos de las iteraciones posteriores. A partir de la descripción de como se aplicó la técnica de captura de requisito, se identificaron los requisitos a cumplir por el sistema, los cuales se muestran a continuación:

RF1 Gestionar subsistema

RF 1.1 Adicionar subsistema

FR 1.2 Eliminar subsistema

RF 1.2 Modificar subsistema

RF2 Gestionar componente

RF 2.1 Adicionar componente

RF 2.2 Eliminar componente

RF 2.3 Modificar componente

RF3 Gestionar paquete del componente

RF 3.1 Adicionar paquete del componente

RF 3.2 Eliminar paquete del componente

RF 3.3 Modificar paquete del componente

RF4 Gestionar funcionalidad del componente

RF 4.1 Adicionar funcionalidad del componente

RF 4.2 Eliminar funcionalidad del componente

RF 4.3 Modificar funcionalidad del componente

RF 4.4 Listar funcionalidades del componente

RF 5 Gestionar requisito del componente

RF 5.1 Adicionar requisito del componente

RF 5.2 Eliminar requisito del componente

RF 5.3 Modificar requisitos del componente

RF 5.4 Listar requisitos del componente

2.3. Especificación del requisito Adicionar componente

Tabla 1Descripción del requisito Adicionar componente

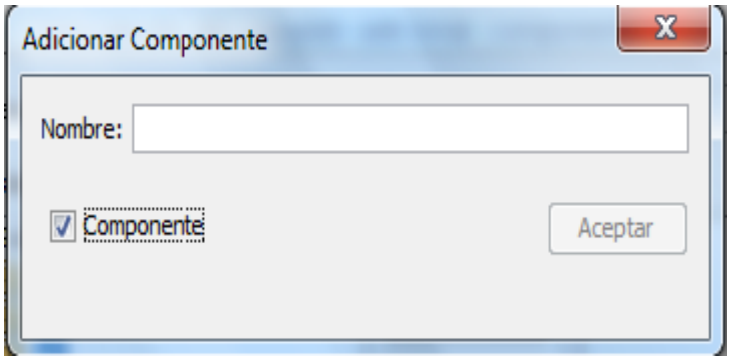
Precondiciones	**Debe haberse seleccionado un paquete de componentes**
Flujo de eventos	
Flujo básico Adicionar componente	
1- El usuario selecciona el botón adicionar.	
2- El usuario selecciona la opción Componente.	
3- El usuario introduce el nombre del componente y selecciona el botón aceptar.	
4- El sistema crea dentro de la carpeta apps una carpeta con el mismo nombre de la descripción del componente creado.	
5- El sistema crea dentro de la carpeta web una carpeta con el nombre del componente creado.	
6- El sistema crea dentro de la carpeta apps del componente la carpeta común la cual contiene los XML del componente.	
7- El sistema crea dentro de la carpeta apps del componente la carpeta controllers donde se almacenan todas las clases controladoras del componente.	
8- El sistema crea dentro de la carpeta apps la carpeta models que contiene las carpetas domain donde se programan las consultas, business donde se programan las modificaciones, como por ejemplo los métodos invocados desde la clase controladora.	

9- El sistema crea dentro de la carpeta apps la carpeta views que es donde se recopilan los ficheros que van a gestionar la capa de presentación, estos son idioma y scripts. Dentro de la carpeta idioma existirán dos subcarpetas, una **“es”** donde se almacenan los archivos que gestionan una presentación en español como por ejemplo ficheros de tipo json que recopilan etiquetas para mostrar mensajes en el idioma correspondiente, y otra carpeta llamada **“en”** donde se gestiona la presentación en inglés. Dentro de la carpeta scripts se incluyen todas las vistas, para ello se crea una carpeta para cada clase controladora y dentro se incluye la vista o script. Estos no son más que archivos de extensión phtml donde se especifica el título de la página que se gestiona y se carga el archivo js que mostrará la presentación como tal.
10- El sistema crea dentro de la carpeta web, la carpeta con el nombre del componente, donde estarán css y js.
11- El sistema valida el nombre del componente.
12- Si el dato es correcto el sistema lo registra.
13- Concluye el requisito.
Poscondiciones
1 N/A
Flujos alternativos
Flujo alternativo: el usuario selecciona la opción Componente
1- El sistema adiciona un componente.
2- Ir al paso 4
Poscondiciones
1 N/A
Flujo alternativo 3.2 el usuario no selecciona la opción Componente
1- El sistema adiciona un paquete de componentes.
2- Ir al paso 4.
Poscondiciones
1- N/A

Flujo alternativo: el usuario cancela la acción		
1- Concluye el requisito.		
Poscondiciones		
1- No se adiciona el componente.		
Validaciones		
1- El nombre del componente debe contener solo caracteres alfabéticos, sin espacios.		
Conceptos	**Componente**	Visibles en la interfaz: Adicionar
Requisitos especiales		N/A
Asuntos pendientes		N/A

Prototipo de interfaz de usuario: Las descripciones textuales y los diagramas no son suficientemente buenos para expresar los requisitos de la interfaz. La construcción de prototipos evolutivos con la participación del usuario final es la forma más sensata de desarrollar una interfaz. Los usuarios deben estar implicados en la evaluación y evolución del prototipo. (37)

2.3.1. Prototipo de interfaz gráfica de usuario



Prototipo de Interfaz de usuario Adicionar Componente.

Solamente se muestra una descripción de requisitos las demás se encuentra en el anexo 7.

2.4. Requisitos no funcionales.

Son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable.(23)

Usabilidad

La extensión podrá ser usada por cualquier persona que posea conocimientos básicos en el manejo de IDE Netbeans y el marco de trabajo Sauxe.

Rendimiento (REN)

Los tiempos de respuesta y velocidad de procesamiento de la información serán rápidos, no mayores de 3 segundos.

Soporte (SOP)

La aplicación contará antes de su puesta en marcha con un período de pruebas, se le dará mantenimiento, configuración y se brindará el servicio de instalación.

Software (SFT)

- **Para el cliente:**
  - Sistema operativo Windows 98 o superior ó Linux.
  - Netbeans 6.9

Hardware (HDW)

- **Para el cliente:**
  - Requerimientos mínimos: Procesador Pentium III a 1GHz, 512 Mb de memoria RAM.

2.5. Patrones de diseños:

Los patrones de diseño son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan. (28)

Patrón Singleton (solitario): Patrón creacional diseñado para restringir la creación de objetos pertenecientes a una clase. Su objetivo es garantizar que una determinada clase solo tenga una instancia y proporcionar un punto de acceso global a esta. Es un patrón muy sencillo de diseñar y a menudo es implementado por otros patrones. (38)

Ejemplo de código de la solución donde se evidencia el patrón Singleton:

private static Principal instance;

Patrón Experto

Este patrón permite asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. La responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo o implementarlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantendrá encapsulada. (38)

Ventajas:

- Se conserva el encapsulamiento.
- Soporta un bajo acoplamiento, lo que favorece al desarrollo de sistemas más robustos y de fácil mantenimiento.
- El comportamiento se distribuye entre las clases que cuentan con la información requerida. Brinda soporte a una alta cohesión. (38)
- Se pone de manifiesto en la clase

2.6. Diagramas de clases de diseño

Una clase de diseño es una construcción similar en la implementación del sistema cuyo lenguaje de diseño es el mismo que el lenguaje de programación. Las operaciones, atributos, tipos, visibilidad (public, protected y private),se pueden especificar con la sintaxis del lenguaje elegido.

Las relaciones entre estas clases se traducen de manera directa al lenguaje: generalización, herencia, asociaciones, agregaciones, atributos. Los métodos tienen correspondencia directa con el correspondiente método en la implementación de las clases. Se pueden postergar algunos requisitos a implementación (por ejemplo: manera de nombrar los atributos y operaciones). Además una clase de diseño puede proporcionar interfaces si tiene sentido hacerlo en el lenguaje de programación. Específicamente los diagramas de clases de diseño son muy útiles porque muestran a través de atributos y métodos la estructura de las clases que después serán escritas en algún lenguaje de programación.

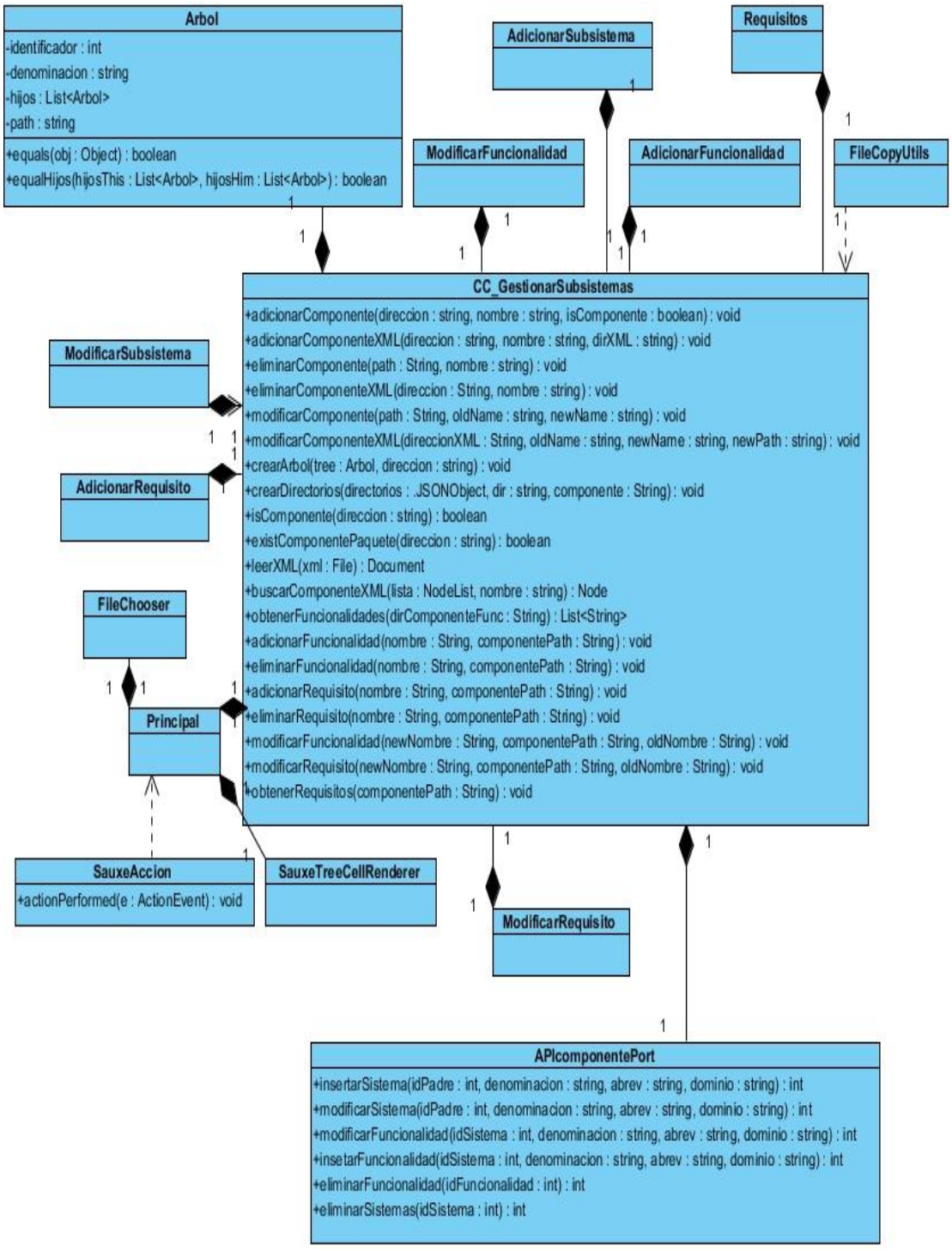


Figura13 Diagrama de clases de mi solución.

2.7. Conclusiones parciales

La aplicación de la técnica de captura de requisitos Tormentas de ideas permitió identificar los requisitos funcionales del sistema.

Se realizó la descripción de los requisitos funcionales y no funcionales que permitieron llevar a cabo el diseño de la solución.

Se generaron los artefactos requeridos por el modelo de desarrollo que documentan la solución y facilitan su mantenimiento posterior.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS

3.1. Introducción

En este capítulo se muestra el modelo de implementación que pone en práctica el diseño de la solución realizado en el capítulo anterior y se especifica el conjunto de validaciones y pruebas que evalúan la calidad del sistema.

3.2. Diagrama de componentes.

El diagrama de componentes muestra las relaciones entre las partes físicas y reemplazables del sistema, por tanto, expresa las dependencias existentes entre estas estructuras denominadas componentes. Estos diagramas contienen relaciones de dependencia que se utilizan para indicar que un componente se refiere a los servicios ofrecidos por otro componente. (39)

En el siguiente diagrama se muestra cómo interatúan los componentes de la solución.

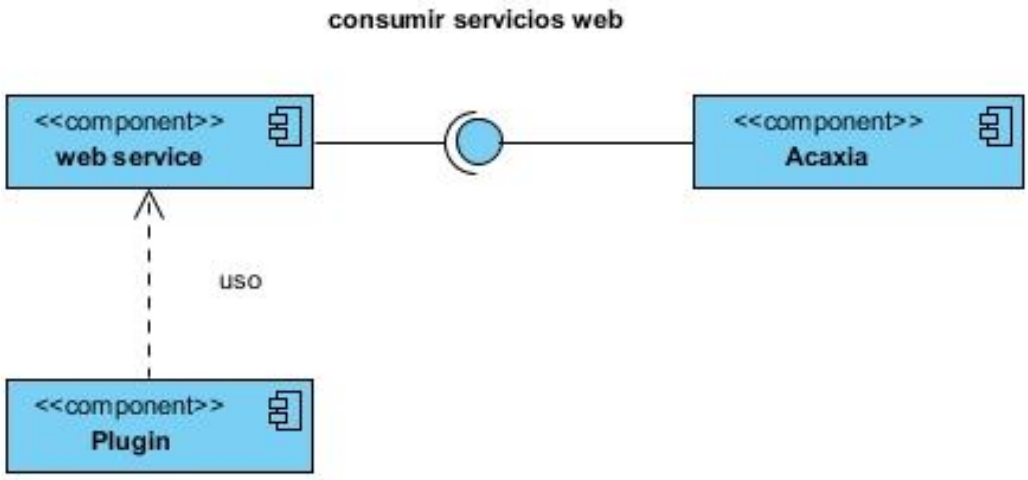


Figura: 23 diagrama de componente de la solución

Servicios web:

insertarSubsitemas: es el servicio que se encarga de adicionar un subsistemas el cual se le necesita que se les pase por parámetro el idPadre, una dirección, dominio estos datos toma del xml que se encuentra en Sauxe, una denominación y abreviatura con estos datos el adiciona en Sauxe desde Netbeans.

insertarFuncionalidad: es el servicio que se encarga de adicionar una funcionalidad el cual se le necesita que se les pase por parámetro el idSubsistema, una dirección, dominio estos datos toma del xml que se encuentra en Sauxe, una denominación y abreviatura con estos datos el adiciona en Sauxe desde Netbeans.

modificarSubsitemas: es el servicio que se encarga de modificar un subsistemas el cual se le necesita que se les pase por parámetro el idPadre, una dirección, dominio estos datos toma del xml que se encuentra en Sauxe, una denominación nueva y abreviatura nueva, el te muestra la denominación y la abreviatura antigua con estos datos el modifica en Sauxe desde Netbeans.

modificarFuncionalidad: es el servicio que se encarga de modificar una funcionalidad el cual se le necesita que se les pase por parámetro el idSubsistema, una dirección, dominio estos datos toma del xml que se encuentra en Sauxe, una denominación nueva y abreviatura nueva además de mostrarte la antigua denominación y abreviatura con estos datos el modifica en Sauxe desde Netbeans.

eliminarFuncionalidad: es el servicio que se encarga de eliminar una funcionalidad el cual se le necesita que se les pase por parámetro el idFuncionalidad, estos datos toma del xml que se encuentra en Sauxe, con estos datos el elimina en Sauxe desde Netbeans.

eliminarSubsistemas: es el servicio que se encarga de eliminar una funcionalidad el cual se le necesita que se les pase por parámetro el idSubsistema, estos datos toma del xml que se encuentra en Sauxe, con estos datos el elimina en Sauxe desde Netbeans.

3.3. Normas y estándares de codificación.

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. Un estándar de programación define la nomenclatura de las variables, objetos, métodos y funciones, teniendo que ver además, con el orden y legibilidad del código escrito.

Notación Húngara: Esta convención se basa en definir prefijos para cada tipo de datos y según el ámbito de las variables. También es conocida como notación REDDICK (por el nombre de su creador). La idea de esta notación es la de dar mayor información al nombre de la variable, método o función definiendo en ella un prefijo que identifique su tipo de dato y ámbito. (23)

Notación PascalCasing: Es como la notación húngara pero sin prefijos. En este caso, los identificadores y nombres de variables, métodos y funciones están compuestos por múltiples palabras juntas, iniciando cada palabra con letra mayúscula.

Notación CamelCasing: Es parecido al Pascal-Casing con la excepción que la letra inicial del identificador no debe estar en mayúscula.(23)

3.3.1. Nomenclatura de las clases. (23)

- **Clases controladoras:**

Las clases controladoras después del nombre llevan la palabra:"Controller".

Ejemplo: CasoController

3.3.2. Nomenclatura de los comentarios. (23)

Deben ser claros y precisos de forma tal que se entienda el propósito de los que se está desarrollando. Antes de declarar una clase se brinda una descripción de esta donde se explica el propósito de la misma y se escribe de la siguiente manera:

Ejemplo: **/***

*** Clase Controladora GestionarSubsistemas.**

**** @package cu.uci.sauxe;**

/**

*****

*** @author dayron**

***/**

3.3.3. Nomenclatura de las variables. (23)

El nombre a emplear para las variables se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing*, y comenzando con un prefijo según el tipo de datos.

Ejemplo: arrMoneda

Prefijos para los tipos de datos

Los prefijos a utilizar en la creación de variables serán los siguientes:

Tabla 2 Prefijos para los tipos de datos.

Tipos de Datos	Prefijos
Arreglos	arr
Objetos	obj
Enteros	int
Cadena	str
float	flt
Boolean	Boo

3.3.4. Métricas de validación para el diseño.

Las métricas han de ser utilizadas para el control de los proyectos. No son ni estándares ni universales, sino que cada proyecto debe seleccionar sus propias métricas en dependencia de sus características. Las métricas de software se pueden clasificar como: (32)

- Métricas orientadas a la función y Métricas orientadas al tamaño.
- También se pueden clasificar según la información que entregan:
- Métricas de productividad, las que se centran en el rendimiento del proceso de ingeniería de software.
- Métricas de calidad, proporcionan una indicación de cómo se ajusta el software a los requisitos explícitos e implícitos del cliente.
- Métricas técnicas, que se centran más en el software que en el proceso a través del cual se ha desarrollado (por ejemplo grado de modularidad o grado de complejidad lógica).

Se estudió el estándar de nomenclatura y codificación definido para el desarrollo con el marco de trabajo Sauxe para lograr la correcta generación del código.

3.3.5. Métricas Orientadas a Objeto

Las métricas orientadas a objetos se han introducido para ayudar a un ingeniero del software a usar el análisis cuantitativo, para evaluar la calidad en el diseño antes de que un sistema se construya. El enfoque de las métricas orientadas a objetos está en la clase, piedra fundamental en la arquitectura orientada a objetos. (23)

El Software Orientado a Objetos (OO[i2]) es fundamentalmente distinto del software que se desarrolla utilizando métodos convencionales. Las métricas para sistemas OO deben de ajustarse a las características que distinguen el software OO del software convencional. Estas métricas hacen hincapié en el encapsulamiento, la herencia, complejidad de clases y polimorfismo. Por lo tanto las métricas OO se centran en métricas que se pueden aplicar a las características de encapsulamiento, ocultamiento de información, herencia y técnicas de abstracción de objetos que hagan única a esa clase. (19)

Se conoce que las medidas y las métricas son componentes clave de cualquier disciplina de la ingeniería; la ingeniería de software orientada a objetos no es una excepción. Lamentablemente, la utilización de métricas para sistemas orientados a objetos ha progresado con mucha más lentitud que la utilización de los demás métodos OO. Sin embargo, a medida que los sistemas OO van siendo más habituales, resulta fundamental que los ingenieros del software dispongan de mecanismos cuantitativos para estimar la calidad de los diseños y la efectividad de los programas OO. (19)

- Los objetivos principales de las métricas orientadas a objetos son los mismos que los existentes para las métricas surgidas para el software estructurado: (19)
- Comprender mejor la calidad del producto.
- Estimar la efectividad del proceso.
- Mejorar la calidad del trabajo realizado en el nivel del proyecto.

3.3.6. Características de las métricas orientadas a objeto

Las métricas para cualquier producto de ingeniería son reguladas por las características únicas del producto. El software orientado a objetos es fundamentalmente diferente del software desarrollado con el uso de métodos convencionales. Por esta razón, las métricas para sistemas orientados a objetos deben ser afinadas a las características que distinguen al software orientado a objeto del software convencional. (29)

[2] Orientado a Objetos

Existen 5 características definidas por Berard que regulan las métricas especializadas: Localización, Encapsulación, Ocultamiento de Información, Herencia y Técnicas de Abstracción de Objetos. (29)

- **Localización**: La localización es una característica que indica la manera en que la información se concentra en un programa. (29)

- **Encapsulamiento**: Empaquetamiento o ligamento de una colección de elementos. Ejemplos de bajo nivel de encapsulación incluyen registros y matrices, y subprogramas (procedimientos, funciones, subrutinas y párrafos), son mecanismos de nivel medio para la encapsulación. Para los sistemas orientados a objetos, la encapsulación engloba las responsabilidades de una clase, incluyendo sus atributos (y otras clases para objetos agregados) y operaciones, y los estados de las clases, definidos por valores de atributos específicos. (29)

La encapsulación influye en las métricas, cambiando el enfoque de las mediciones de un módulo simple, a un paquete de datos (atributos) y modelos de procesos (operaciones). (29)

- **Ocultación de la Información**: La ocultación de la información suprime (u oculta) los detalles operacionales de un componente de programa. Solo se proporciona la información necesaria para acceder al componente a aquellos otros componentes que deseen acceder. Un sistema orientado a objetos bien diseñado debe implementar ocultación de información. Por esta razón, las métricas que proporcionan una indicación del grado de ocultación logrado suministran un indicio de la calidad del diseño orientado a objetos. (29)

- **Herencia:** La herencia es un mecanismo que habilita las responsabilidades de un objeto, para propagarse a otros objetos. La herencia ocurre a través de todos los niveles de una jerarquía de clases. Ya que la herencia es una característica vital en muchos sistemas orientados a objetos, muchos métodos orientados a objetos se centran en ella.

- **Abstracción**: La abstracción es un mecanismo que permite al desarrollador concentrarse en los detalles esenciales de un componente de programa (ya sean datos o procesos), presentando poca atención a los detalles de bajo nivel. Como Bernard declara << la abstracción es un concepto relativo. A medida que se mueve a niveles más altos de abstracción, se ignoran más y más detalles, es decir, se tiene una visión más general de un concepto o elemento. A medida que se mueve a niveles de abstracción más bajos, se introducen más detalles, es decir, se tiene una visión más específica de un

concepto o elemento. (29)

3.3.7. Métricas Orientadas a Clases. (29)

Otra de las propuestas de métricas más aplicadas son las métricas propuestas por Lorenz y Kidd, separándolas en cuatro amplias categorías:

- Tamaño
- Herencia
- Valores internos
- Valores externos

Las métricas escogidas como instrumento para evaluar la calidad del diseño descrito anteriormente y su relación con los atributos de calidad son Tamaño Operacional de Clases y Relaciones entre clases.

3.4. Validación del diseño propuesto.

A continuación se presentarán las métricas Tamaño operacional de clase (TOC) y Relaciones entre clases (RC), necesarias para evaluar la calidad del diseño propuesto a nivel de componentes. Ambas métricas no fueron aplicadas durante el diseño procedimental, sino que fueron retrasadas hasta tener disponible el código fuente, pues la complejidad del negocio exigía la incorporación de diversas subrutinas necesarias para la implementación de cada procedimiento descrito en el diseño de clases. Las pruebas que se le hicieron a la aplicación arrojaron un resultado bueno, liberándose la aplicación y demostrando que esta cumple con los estándares de calidad definidos en el CEIGE.

3.4.1. Tamaño operacional de clase (TOC):

Está dado por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad:

Atributo de calidad	Modo en que lo afecta
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución del grado de reutilización de la clase.

Tabla 3 Atributos de calidad evaluados por la métrica TOC

Para los cuales están definidos los siguientes criterios y categorías de evaluación:

Atributo	Categoría	Criterio
Responsabilidad	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Complejidad implementación	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Reutilización	Baja	>2*Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	<=Promedio

Tabla 4 Criterios de evaluación para la métrica TOC

Resultados obtenidos de la aplicación de la métrica TOC

Responsabilidad	Cantidad de clases	Porciento
Baja	2	66,66666667
Media	1	33,33333333
Alta	0	0

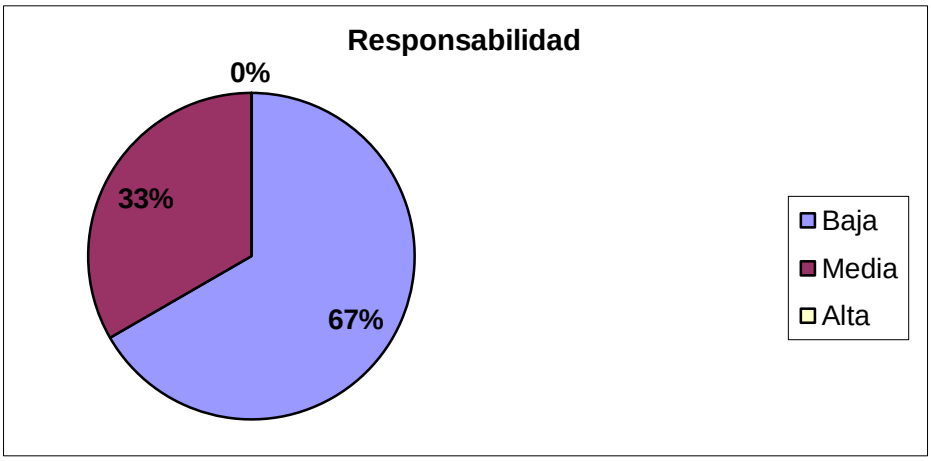


Figura: 14 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad.

Complejidad	Cantidad de clases	Porciento
Baja	2	66,7
Media	1	33,3
Alta	0	0

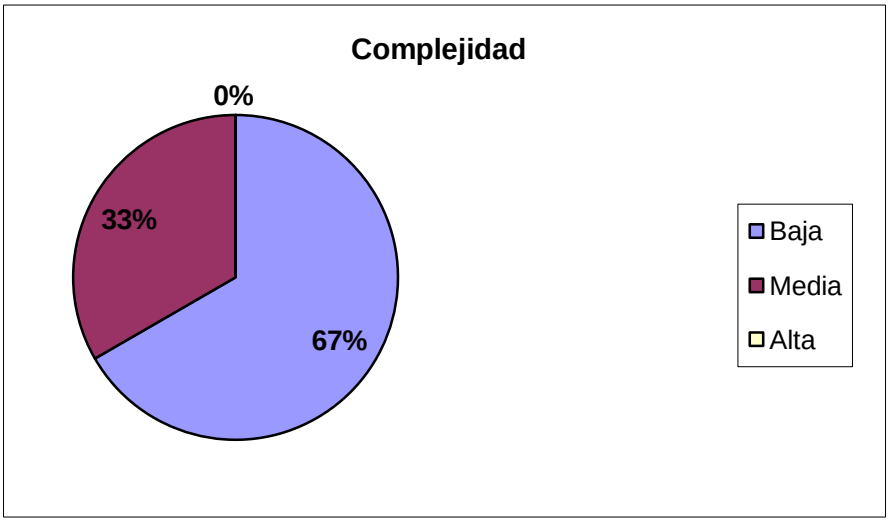


Figura: 15 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación.

Reutilización	Cantidad de clases	Porciento
Alta	2	66,7
Media	1	33,3
Baja	0	0

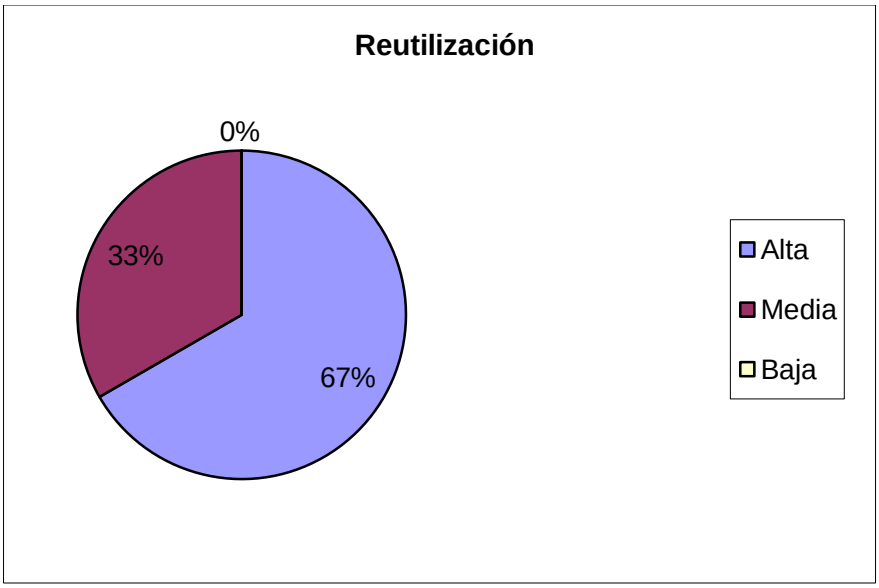


Figura: 16 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Los resultados obtenidos durante la evaluación del instrumento de medición de la métrica TOC demuestran que el diseño propuesto para la solución fue satisfactorio ya que si analizamos la Tabla 5 "Atributos de calidad evaluados por la métrica TOC" con los resultados de los instrumentos de medición de la métrica se confirmaría un resultado positivo que se traduce en una elevada reutilización, baja complejidad y responsabilidad en el diseño propuesto.

3.5. Relaciones entre clases (RC)

Está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad:

Atributo de calidad	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del Acoplamiento de la clase.
Complejidad de mantenimiento	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
Reutilización	Un aumento del RC implica una disminución en el grado

	de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 6 Atributos de calidad evaluados por la métrica RC

Para los cuales están definidos los siguientes criterios y categorías de evaluación:

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
Complejidad de mantenimiento	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Reutilización	Baja	>2*Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	<=Promedio
Cantidad de pruebas	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio

Tabla 7 Criterios de evaluación para la métrica RC.

Acoplamiento	Cantidad de clases	Porciento
Ninguno	1	33,3
Bajo	0	0
Medio	2	66,7
Alto	0	0

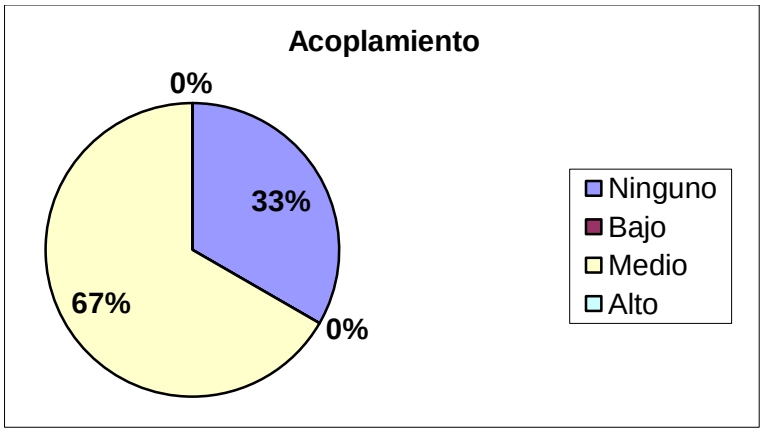


Figura: 17Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.

Complejidad de Mantenimiento	Cantidad de clases	Porciento
Baja	1	33,3
Media	2	66,7
Alta	0	0

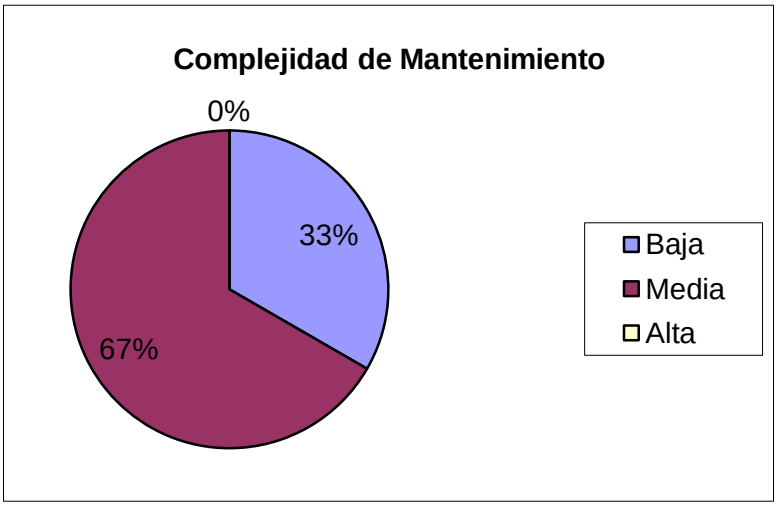


Figura: 18 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento.

Cantidad de Pruebas	Cantidad de clases	Porciento
Baja	1	33,3
Media	2	66,7
Alta	0	0

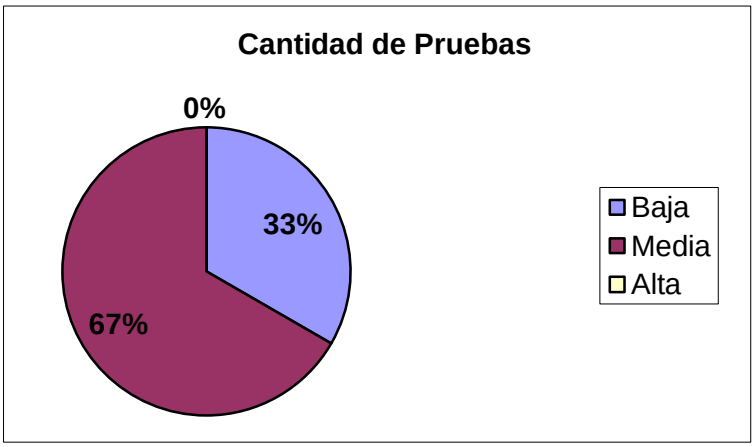


Figura: 19 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

Reutilización	Cantidad de clases	Porciento
Baja	0	0
Media	1	33,3
Alta	2	66,7

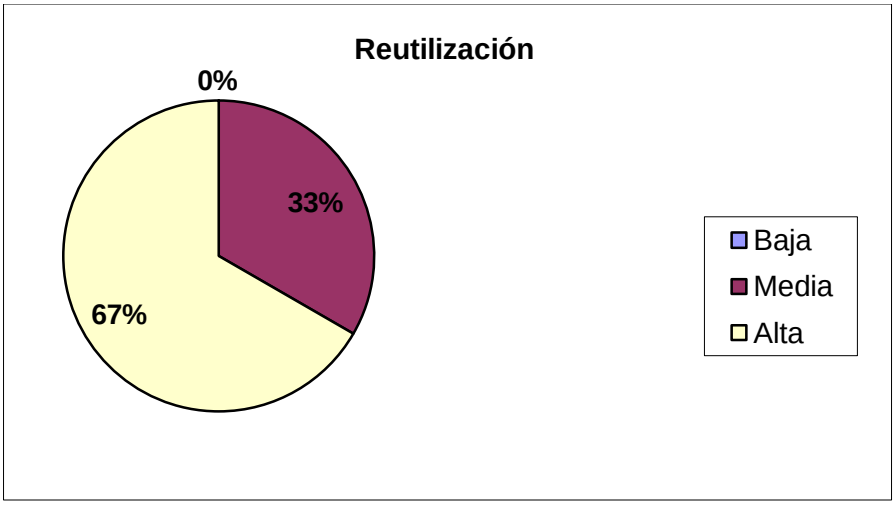
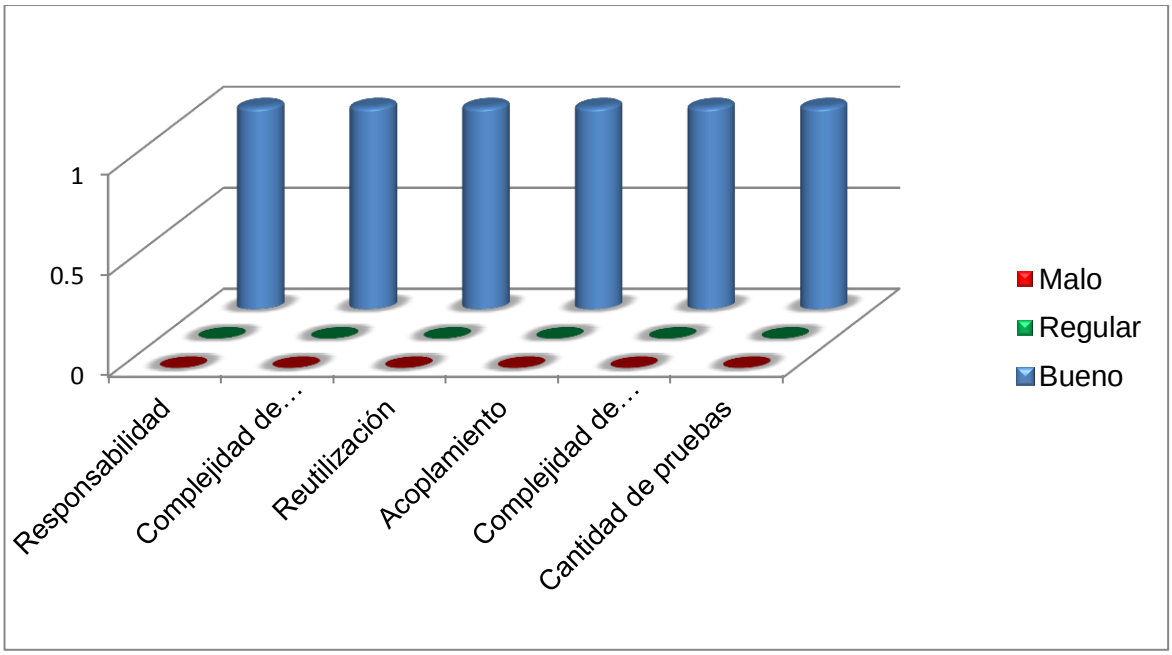


Figura: 20 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización.

Si analizamos la Tabla 14 “Atributos de calidad evaluados por la métrica RC” los resultados obtenidos durante la evaluación del instrumento de medición de la métrica RC demuestran que los atributos de calidad arrojaron un resultado positivo, donde se obtuvo un bajo acoplamiento, complejidad, cantidad de pruebas y una alta reutilización. Estos resultados garantizan que el proyecto pueda ser movido de lugar con facilidad, que se utilicen sus clases o sus métodos en otras aplicaciones, códigos de fácil entendimiento.

Atributos	***Malo***	***Regular***	***Bueno***
Responsabilidad	0	0	1
Complejidad de Implementación	0	0	1
Reutilización	0	0	1
Acoplamiento	0	0	1
Complejidad de Mantenimiento	0	0	1
Cantidad de pruebas	0	0	1

Tabla 8 Matriz de cubrimiento.



Las métricas de software aplicadas posibilitaron estimar la calidad de los atributos internos del producto, demostrando una aceptable calidad del diseño. Esto se debe a que en las TOC la reutilización dio alta esto es positivo ya que dice que esta clase puede ser reutilizada con facilidad dentro del diseño de software, la complejidad de implementación que se obtuvo fue baja garantizando que se facil implementar una clase del diseño, su código se puede entender con facilidad, responsabilidad baja garantiza que si borras una clase no se afectaría tanto tu diseño.

La métrica RC también arrojo resultados satisfactorios, esto se evidencia en el resultado de sus pruebas donde el acoplamiento dio bajo esto garantiza que se puede mover con facilidad el módulo de lugar, complejidad de mantenimiento bajo esto garantiza que pueda ser optimizado el código con facilidad o arreglado algún error, cantidad de pruebas dio bajo garantizando que se le tenga que hacer pocas pruebas a tu diseño; como se puede apreciar la evaluación de los atributos de calidad arrojaron un resultado satisfactorio.

3.6. Pruebas:

Las pruebas en un proceso de desarrollo de software son los diferentes procesos que se deben realizar con el objetivo de asegurar la terminación y calidad de la solución propuesta. Existen diferentes métodos de pruebas, entre ellos se pueden encontrar:

Las pruebas de caja negra: verifican las especificaciones funcionales sin tener en cuenta la estructura interna del programa y son realizadas sin el conocimiento interno del producto. El objetivo es demostrar que las funciones del software son operativas, que las entradas se aceptan de forma adecuada y se produce un resultado correcto, además que la integridad de la información externa se mantiene, saber qué es lo que hace el software pero sin entrar en detalles de código, es decir, que es lo que hace, y no cómo lo hace. Por ello se realizan sobre la interfaz del sistema controlando los datos de entrada y de salida. (39)

Las pruebas de caja blanca: denominadas pruebas 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. En estas pruebas se examina el programa en varios puntos sobre el código para determinar si el estado real coincide con el esperado. Así como las pruebas de caja negra ejercitan los requisitos funcionales desde el exterior del módulo, las de caja blanca están dirigidas a las funciones internas. Estas pruebas se llevan a cabo en primer lugar sobre un módulo concreto, para luego realizar las de caja negra sobre varios subsistemas.

El objetivo de las pruebas es la detección de defectos en el software (descubrir un error es el éxito de una prueba). Con el diseño de las pruebas se pretende encontrar y documentar los defectos que puedan afectar la calidad del software, asegurar que el software trabaje como fue diseñado, además de validar que los requisitos fueron implementados correctamente.

3.6.1. Pruebas de caja blanca

Objetivo

El objetivo de realizar este tipo de prueba al sistema es que se garantice que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo o método, todos los bucles en sus límites operacionales así como las estructuras internas de datos para asegurar su validez. (23)

Alcance

El proceso de pruebas de caja blanca se va a concentrar principalmente en validar a través del framework de software libre JUnit, que cada uno de los módulos o segmentos de códigos funcione apropiadamente.

Descripción

La prueba de caja blanca es considerada como uno de los tipos de pruebas más importantes que se le aplican a los software, logrando como resultado que disminuya en un gran por ciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad. (23)

Para el desarrollo de estas pruebas unitarias se utilizó framework JUnit, ya que ofrece las funcionalidades necesarias para implementar pruebas en un proyecto desarrollado en Java. Además cuenta con una interface simple que informa si cada una de las pruebas realizadas o conjunto de pruebas falló, pasó o fue ignorada.

Inicialmente se definieron los casos de prueba, uno por cada clase implementada. Luego se definieron e implementaron los ficheros de configuración necesarios para establecer la comunicación entre las diferentes capas de la aplicación. Se identificaron los métodos a probar dentro de cada caso de prueba, así como el resultado esperado en cada uno. Y por último se procedió a realizar los casos de prueba diseñados.

Con el objetivo de mostrar la realización de los casos de prueba, a continuación se muestran un grupo de imágenes con sus descripciones.

```
/**
 * Test of obtenerEstructura method, of class CC_GestionarSubsistemas.
 */
@Test
public void testObtenerEstructura() throws Exception {
    System.out.println("obtenerEstructura");
    CC_GestionarSubsistemasPruv instance = new CC_GestionarSubsistemasPruv();
    instance.SAUXE_PATH = "M:/Sauxe";
    Arbol expResult = null;
    Arbol hijo1 = new Arbol(1, "Seguridad", "M:/Sauxe/apps/seguridad");
    Arbol hijo2 = new Arbol(1, "Metadatos", "M:/Sauxe/apps/metadatos");
    Arbol hijo3 = new Arbol(1, "Traza", "M:/Sauxe/apps/traza");
    Arbol root = new Arbol(0, "Subsistemas", "M:/Sauxe/apps");
    List<Arbol> list = new ArrayList<Arbol>();
    list.add(hijo1);
    list.add(hijo2);
    list.add(hijo3);
    root.setHijos(list);
    Arbol result = instance.obtenerEstructura();
    expResult = root;
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
}
```

Figura: 31 Implementación del método testObtenerEstructura ().

En las imágenes anteriores se muestran una prueba al método *testObtenerEstructura ()*, primero se llenan las variables con los datos que se desean probar. En la última línea del método de prueba se hace la llamada al método *assertEquals* el cual recibe como parámetros el resultado esperado y la ejecución del método a probar para que este evalúe si coinciden ambos.

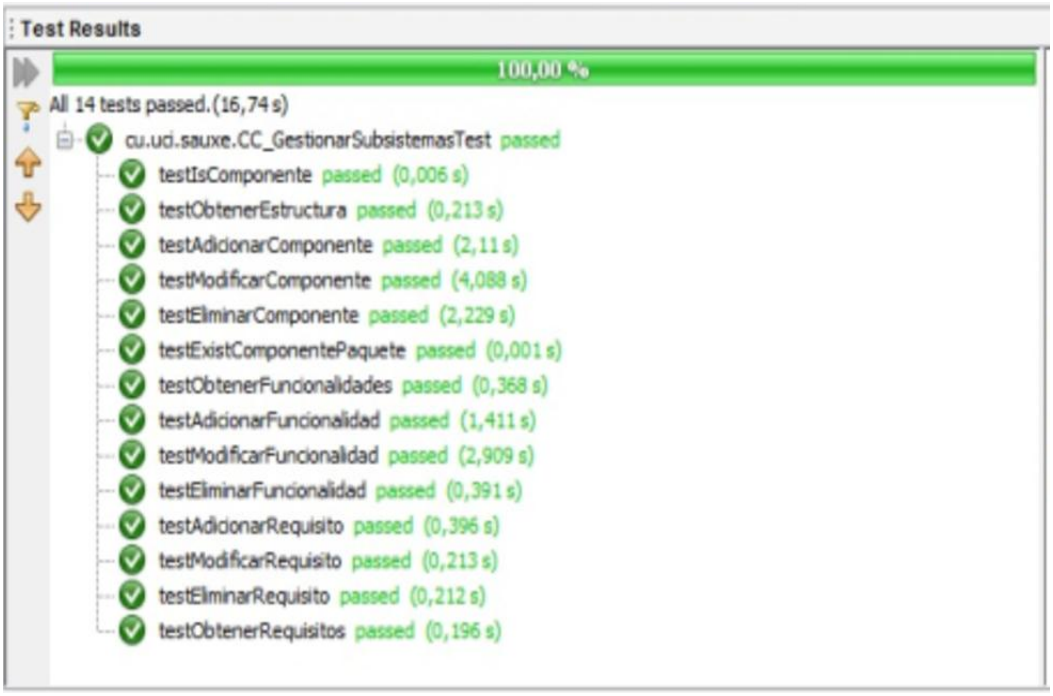


Figura: 22 Resultado de la ejecución de las pruebas

Como se puede apreciar en la figura 22 todas las pruebas pasaron debido a que los resultados esperados coincidieron con los resultados de la ejecución de los métodos.

3.6.2. Pruebas de caja negra

Objetivo

El objetivo de realizar este tipo de prueba al sistema es para detectar el incorrecto o incompleto funcionamiento de este, así como los errores de interfaces, rendimiento y errores de inicialización y terminación.

Alcance

El proceso de pruebas de caja negra se va a centrar principalmente en los requisitos funcionales del software para verificar el comportamiento de la unidad observable externamente y la calidad funcional.

Descripción

Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales del programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

La prueba de caja negra no es una alternativa a las técnicas de prueba de la caja blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la caja blanca. (15)

Existen varias técnicas para desarrollar la prueba de caja negra, entre ellas están: (32)

- Técnica de la Partición de Equivalencia: divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. En esencia esta técnica se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.
- Técnica del Análisis de Valores Límites: los errores tienden a darse más en los límites del campo de entrada que en el centro. Esta técnica lleva a una elección de casos de prueba que ejerciten los valores límites.
- Condiciones sublímite: las condiciones límite normales son las más obvias de descubrir. Estas son definidas en la especificación o son evidentes al momento de utilizar el software. Algunos límites, sin embargo, son internos al software, no son necesariamente aparentes al usuario final pero aun así deben ser aprobadas por el probador. Estas son conocidas como condiciones sublímite o condiciones límite internas.
- Técnica de prueba basada en grafos: permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

- Modelado del flujo de transacción: los nodos representan los pasos de alguna transacción, y los enlaces representan las conexiones lógicas entre los pasos.
- Modelado de estado finito: los nodos representan diferentes estados del software observables por el usuario, y los enlaces representan las transiciones que ocurren para moverse de estado a estado.
- Modelado de flujo de datos: los nodos objetos de datos y los enlaces son las transformaciones que ocurren para convertir un objeto de datos en otro.
- Modelado de planificación: los nodos son objetos de programa y los enlaces son las conexiones secuenciales entre esos objetos. Los pesos de enlace se usan para especificar los tiempos de ejecución requeridos al ejecutarse el programa.
- Gráfica Causa-efecto: la gráfica Causa-efecto representa una ayuda gráfica en seleccionar, de una manera sistemática, un gran conjunto de casos de prueba. Tiene un efecto secundario beneficioso en precisar estados incompletos y ambigüedades en la especificación.

  De las técnicas de prueba de caja negra antes mencionadas se utilizó la de Partición de Equivalencia. Esta técnica permite obtener un conjunto de pruebas que reducen el número de casos de prueba que se deben realizar para evaluar correctamente el software, esto se debe a que se centra en la evaluación de clases de equivalencia que representan un conjunto de estados válidos o no válidos para condiciones de entradas existentes en el software.

Para definir las clases de equivalencia se tuvieron en cuenta un conjunto de reglas: (16)

- Si una condición de entrada especifica un rango, entonces se confeccionan una clase de equivalencia válida y dos inválidas.
- Si una condición de entrada especifica la cantidad de valores, se identifica una clase de equivalencia válida y dos inválidas.
- Si una condición de entrada especifica un conjunto de valores de entrada y existen razones para creer que el programa trata en forma diferente a cada uno de ellos, se identifica una clase válida para cada uno de ellos y una clase inválida.
- Si una condición de entrada especifica una situación de tipo “debe ser”, se identifica una clase válida y una inválida.

Luego de tener las clases válidas e inválidas definidas, se diseñaron los casos de prueba para todas las agrupaciones de requisitos Gestionar Componente, Gestionar Subsistemas, Gestionar Paquete de Componente, Gestionar Funcionalidad, Gestionar Requisitos, estos puede ser consultado en el anexo 4. Cada uno de estos casos de prueba se definió teniendo en cuenta lo siguiente: (15)

- Escribir un nuevo caso de prueba que cubra tantas clases de equivalencia válidas no cubiertas como sea posible hasta que todas las clases de equivalencia hayan sido cubiertas por casos de prueba.
- Escribir un nuevo caso de prueba que cubra una y solo una clase de equivalencia inválida hasta que todas las clases de equivalencias inválidas hayan sido cubiertas por casos de prueba.

Estas pruebas fueron realizadas por un equipo de especialistas del Departamento de Tecnología que validaron el cumplimiento de los requisitos identificados. Se realizaron 3 iteraciones quedando finalmente liberada la solución.

3.7. Conclusiones Parciales

Se realizaron los artefactos definidos por el modelo de desarrollo que permitieron la correcta implementación del sistema.

Se validó el diseño de la solución empleando las métricas que proponen Kidd y Lorenz, TOC y RC las cuales avalúan los atributos de calidad reutilización, acoplamiento, responsabilidad, cantidad de pruebas, complejidad de mantenimiento, complejidad de implementación las que arrojaron resultados satisfactorios.

Se validó la extensión mediante la realización de pruebas estructurales y funcionales las arrojaron resultados satisfactorios desde el punto de vista interno y funcional.

CONCLUSIONES GENERALES.

En el presente trabajo de diploma se reflejó el estudio realizado que permitió la creación de una extensión para el IDE NetBeans que genera componentes utilizando el marco de trabajo Sauxe. En el mismo se obtuvieron los siguientes resultados:

- Se realizó un estudio del framework Sauxe que permitió comprender las ventajas que proporcionaría contar con un plugin para realizar todo el proceso de creación de un componente. Se realizó un estudio de los principales generadores de código que permitió llegar a la conclusión que estos no podía ser usados porque no generan la estructura base de los componentes de Sauxe y el código que generan no cumple con los estándares de nomenclatura y codificación definidos por el marco de trabajo Sauxe.
- Se realizó el análisis y el diseño de la extensión, cumpliendo con todos los requerimientos especificados, aplicándose diversos patrones para mejorar su calidad.
- Se realizó la implementación de las funcionalidades necesarias para la creación de componentes, paquetes de componentes, funcionalidades y requisitos de los componentes.
- Se realizó la validación del diseño de la solución aplicando las métricas TOC y RC las que arrojaron que el diseño tenía una calidad aceptable pues los atributos (reutilización, acoplamiento, responsabilidad, cantidad de pruebas, complejidad de mantenimiento, complejidad de implementación) tuvieron un impacto positivo en el mismo.
- Se realizó la validación de la solución propuesta mediante pruebas estructurales y funcionales donde todas arrojaron resultados satisfactorios desde el punto de vista interno y funcional.

RECOMENDACIONES

Se recomienda para próximas versiones de la solución incluir la asignación de permisos desde el plugin.

REFERENCIAS

1. **Alejandro Álvarez Hernández, Humberto Rodríguez Peláez.** Plugin para la gestión de pruebas en la herramienta REDMINE. Ciudad de la Habana : s.n., Junio de 2011.

3. **Camejo, René R. Bauta.** Desarrollo de una herramienta generadora de ficheros de mapeo, para la persistencia de objetos en esquemas relacionales basada en Doctrine. 2009.

4. **Carlos Omar Meza Ortíz, Rafael Rivera López, José de Jesús Ceballos Mejía y Abelardo Rodríguez León.** Una Arquitectura Modular para el desarrollo de un IDE que apoye a la Enseñanza de los Fundamentos de la Programación Orientada a Objetos. 2010.

41 cnet. (n.d.). Retrieved 6 2, 2012, from cnet: http://download.cnet.com/Visual-Paradigm-for-UML-Enterprise-Edition/3000-2383_4-10074323.html

5. **Carlos González Iglesias, Edel Moreno Lemus.** Software para el Estudio de Sistemas Biológicos versión 2.0. Ciudad Habana : s.n., JUNIO 2010.

7. **Daiamna Pi Acuña, Julio Cesar Sanz Medrano.** Automatización del subsistema DASNA. Análisis y Diseño del módulo de control de la Formación Emergentes de Auditores. Ciudad de la Habana : s.n., 2008.

39. Departamento de Informática, Chile. [En línea] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/15-Implementacion.pdf.

8. Desarrollo Web. Que es Java? [Online] [Cited: diciembre 12, 2011.] http://www.desarrolloweb.com/articulos/497.php.

9. **Eduardo Aranda Pierre, Valentín de León Torres.** Implantación de un entorno de Integración Continua en el proyecto SIGEP. Ciudad de la Habana : s.n., Mayo 2010.

10.EcuRed. [Online] [Cited: diciembre 12, 2011.] http://www.ecured.cu/index.php/IDE_de_Programaci%C3%B3n.

11. **Funes, Luis Enrique.** Conociendo Netbeans Platform Introduccion. [Online] [Cited: diciembre12, 2011.] http://mendozajug.com.ar/portal/index.php?option=com_content&task=view&id=64&Itemid=2.

13. Geneura. Geneura. [En línea] [Citado el: 18 de 03 de 2012.] http://geneura.ugr.es/CUR/XML/.

14. **Gutierrez, Cadete Leordan Valdés.** Diseño e implementación del componente de configuración. Ciudad de la Habana : s.n., 2010.

15. Guía Ubuntu. [En línea] [Citado el: 10 de diciembre de 2009.] http://www.guia-ubuntu.org/index.php?title=NetBeans.

16. Grupo Alarcos - Universidad de Castilla-La Mancha. [En línea] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema09.pdf.

17. **Ivar Jacobson, Grady Booch, James Rumbaugh**. El Proceso Unificado de Desarrollo de Softwares l.: Pearson Educación, S.A, 2000. 84-7829-036-2.

37. Ingenieria en Sistemas 2009 UNL. (n.d.). Retrieved 6 10, 2012, from http://sistemas2009unl.wordpress.com/prototipos-informaticos/

18. **Jacobson, I. and Booch, G. y Rumbaugh J.** El Proceso Unificado de Desarrollo de software. s.l. : Addison-Wesley., 2012.

36. **Jose Alberto Arauzo, J. J.** (2010). Propuesta de una ontología para la especificación de procesos. Valladolid.

37. **Joaquin Gracia Murugarren.** Prácticas y métodos para mejorar el desarrollo de Proyectos de Software. Patrones de diseño. Diseño de Software Orientado a Objetos. [En línea] Mayo de 2005. http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

19. **Maribel Silva Muñoz, Sándor Rodríguez Prieto**. Diseño e Implementación de un sistema informático integrado para la Gestión de Compras de Bienes y Contratación de Servicios en los Registros y las Notarías de la República Bolivariana de Venezuela. La Habana : s.n., 2008.

20. Métodos de prueba de caja negra. [En línea] http://gemini.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node28.html.

29. masadelante. (n.d.). Retrieved 5 13, 2012, from masadelante.com: http://www.masadelante.com/faqs/plug-in

21. Métricas para Sistemas Orientados a Objetos. [En línea] [Citado el: 8 de abril de 2009.]http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/gonzalez_d_h/capitulo6.pdf.

22. **Peña, Omar A. Díaz.** Diseño arquitectónico de una plataforma para la arquitectura distribuida en PHP basada en Servicios Web. Ciudad de la Habana : s.n., 2010.

23. **Padrón, Mayara López.** Implementación de una herramienta para el Análisis de las trazas en el Marco de trabajo Sauxe. habana : s.n., Junio, 2011.

24. **Pérez, M.S**., Propuesta de modelo de desarrollo de software tecnológico del Centro de Soluciones de Gestión. 2009.

39. Pressman, Roger S. Ingeniería del Software. Un enfoque práctico. España : McGraw-Hill, 2002

25. **Pressman, Roger S.** Ingenería de Software. Un enfoque práctico. 1998. Vol.

26. Reasons to Choose Visual Paradigm. Visual Paradigm. [Online] 2012. http://www.visual-paradigm.com/aboutus/10reasons.jsp.

27. Rios, S. S. (n.d.). slideshare. Retrieved mayo 13, 2012, from slideshare: http://www.slideshare.net/SergioRios/unidad-5-mad-modelado-analisis-modelo-conceptual

28 **René R. Bauta Camejo, Ariel Torres Galvez.** GENERACIÓN DE FICHEROS DE MAPEO MEDIANTE EL USO DE UNA HERRAMIENTA BASADA EN EL FRAMEWORK DOCTRINE. Habana : s.n., 2010.

2. sourceforge.net. (n.d.). Retrieved 6 10, 2012, from sourceforge.net: :http://sourceforge.net/projects/uml/files/stats/timeline?dates=2012-01-01+to+2012-06-20

30. Siberia. [Online] enero 2010. [Cited: enero 18, 2012.] http://www.saberia.com/2010/01/que-es-un-plugin/

32. scribd. (n.d.). Retrieved from scribd: pagina http://es.scribd.com/doc/50190811/ArgoUML

42. Software.filestube. (n.d.). Retrieved 6 2, 2012, from software.filestube: http://software.filestube.com/software,6394a768,Visual+Paradigm+for+UML+%28Community+Edition%29.html

40. Softwarelogia. (n.d.). Retrieved 6 1, 2012, from Softwarelogia: http://softwarelogia.com/2008/07/25/%C2%BFque-es-un-plugin/

31. tigris.org. (n.d.). Retrieved 5 20, 2012, from tigris.org: http://argouml.tigris.org/

36. Tecnología y Synergix. [En línea] [Citado el: 25 de 5 de 20011.] http://synergix.wordpress.com.

33. Universidad de Belgrano. [Online] [Cited: diciembre 10, 2011.] http://www.ub.edu.ar.

34. Visual Paradigm. [Online] [Cited : 12 de diciembre de 2011.] http://www.visual-paradigm.com.

38. **Visconti, Marcello y Astudillo, Hernán.** Fundamentos de Ingeniería de Software. Patrones de Diseño. [En línea] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.

35. **Yunier René Pérez Valdés, Julio Antonio Villaverde Martínez, Aurelio Antelo Collado, Luis Grabiel García.** SERVICIOS DE LA PLATAFORMA GRAPH-TOOL. Ciudad de La Habana, Cuba: s.n.

ANEXOS

Anexo 1: Fotos de directorio que forman el Frameworks Sauxe

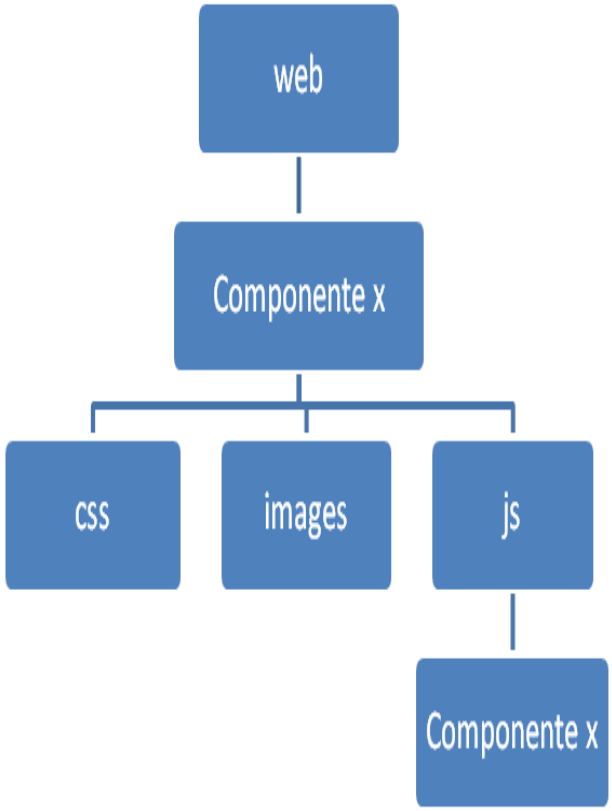


Figura: 1 Directorios ubicados en la carpeta Web

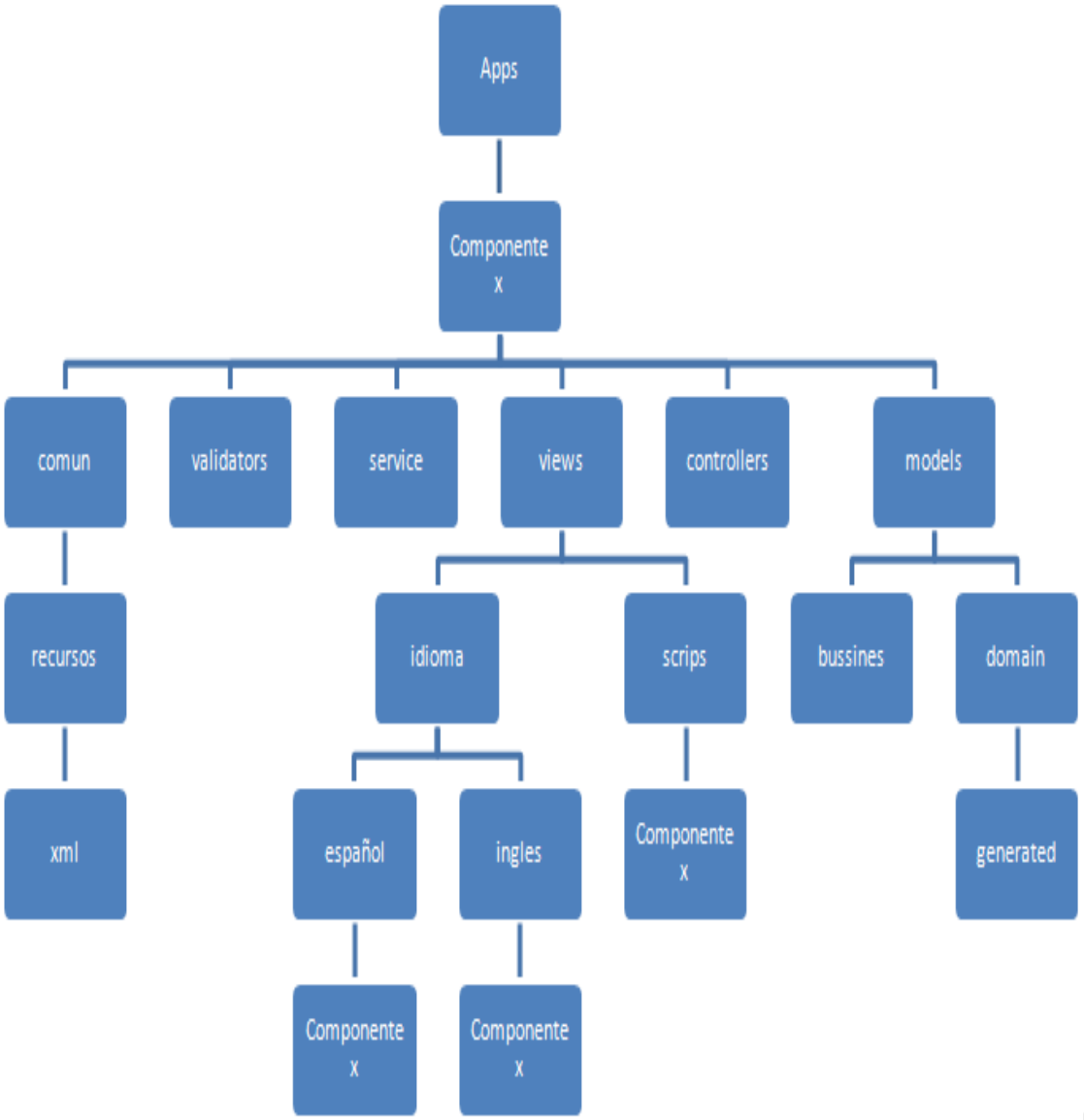


Figura: 2 Directorios ubicados en los directorios que estén dentro de la carpeta apps

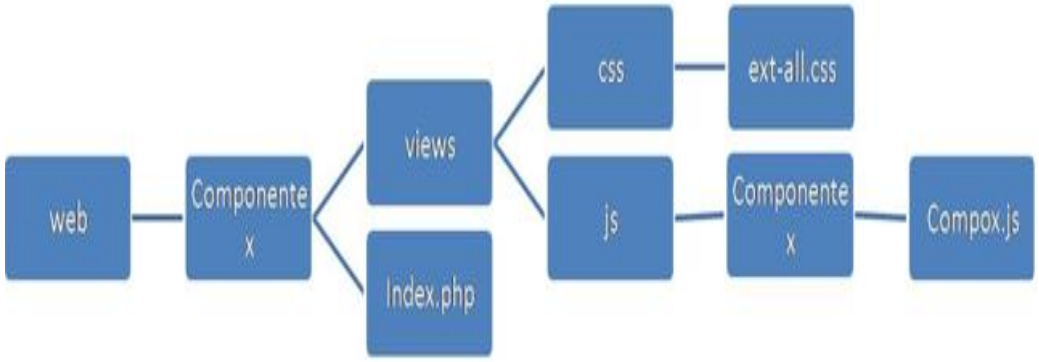


Figura: 3 Ficheros ubicados en los directorios que estén dentro de la carpeta web

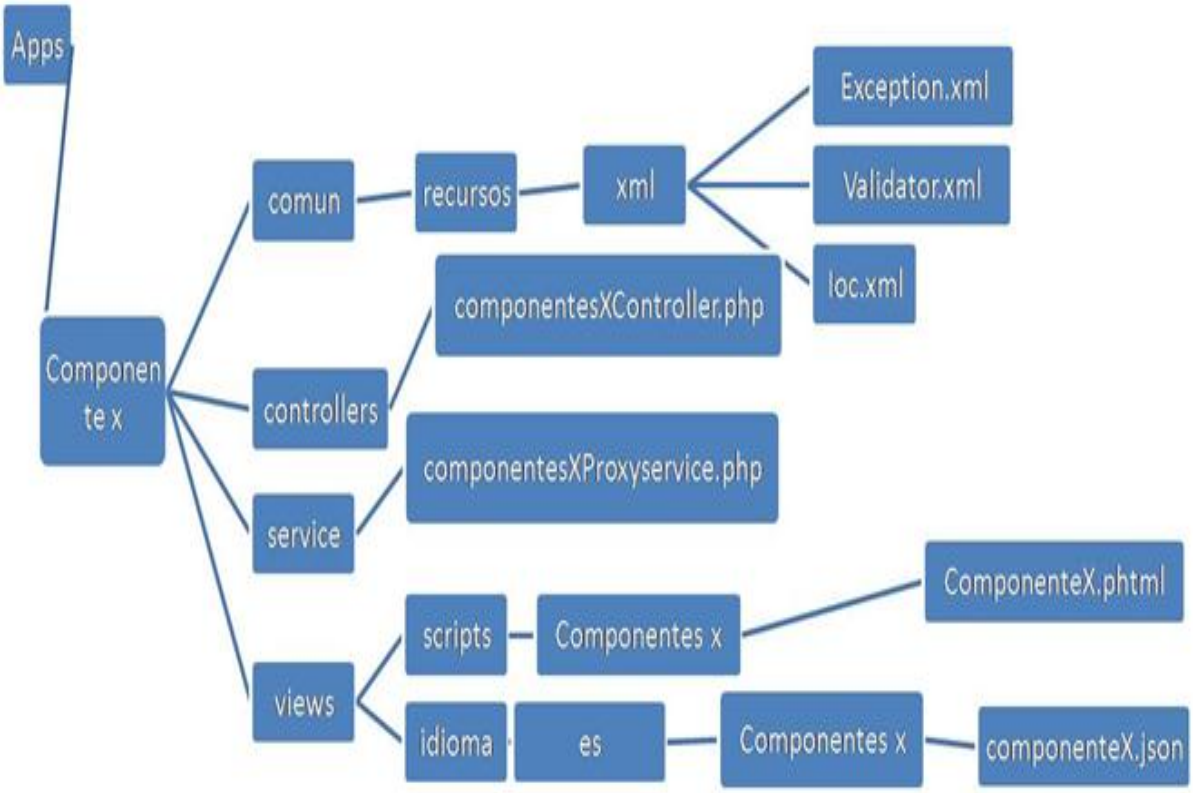


Figura:4 Ficheros ubicados en los directorios que estén dentro de la carpeta Apps

Anexos 2: Instrumento de medición de la métrica relaciones entre clase (RC).

Clase	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad Mant.	Reutilización	Cantidad de Pruebas
GestionarSubsistemas_Controller	2	Medio	Media	Baja	Media
SauxeAction	2	Medio	Media	Baja	Media
Árbol	0	Ninguno	Baja	Baja	Baja

Tabla 9 Resultados de la evaluación de la métrica RC y su influencia en los atributos de calidad (Acoplamiento, Complejidad de Mantenimiento, Reutilización y Cantidad de Pruebas).

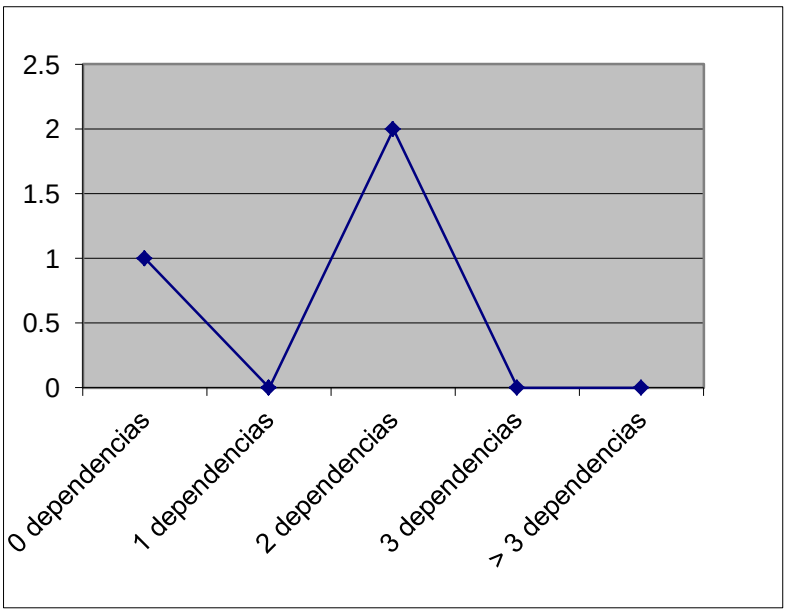


Figura: 41 Representación de los resultados de la evaluación de la métrica RC agrupados por la tendencia de los valores.

Anexos 3 Instrumento de medición de la métrica Tamaño operacional de clase (TOC).

Subsistema	Clase	Cant. Procedimientos	Responsabilidad	Complejidad	Reutilización
Sauxe	**GestionarSubsistemas_Controller**	20	Media	Media	Media
Sauxe	**SauxeAction**	1	Baja	Baja	Alta
Sauxe	**Árbol**	10	Baja	Baja	Alta

Tabla 10 Resultados de la evaluación de la métrica TOC y su influencia en los atributos de calidad (Complejidad de Implementación y Reutilización).

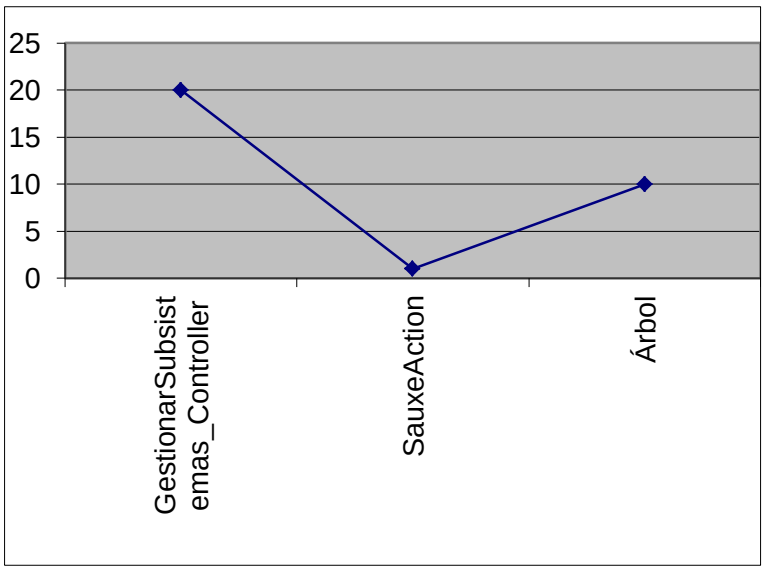


Figura: 22 Representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

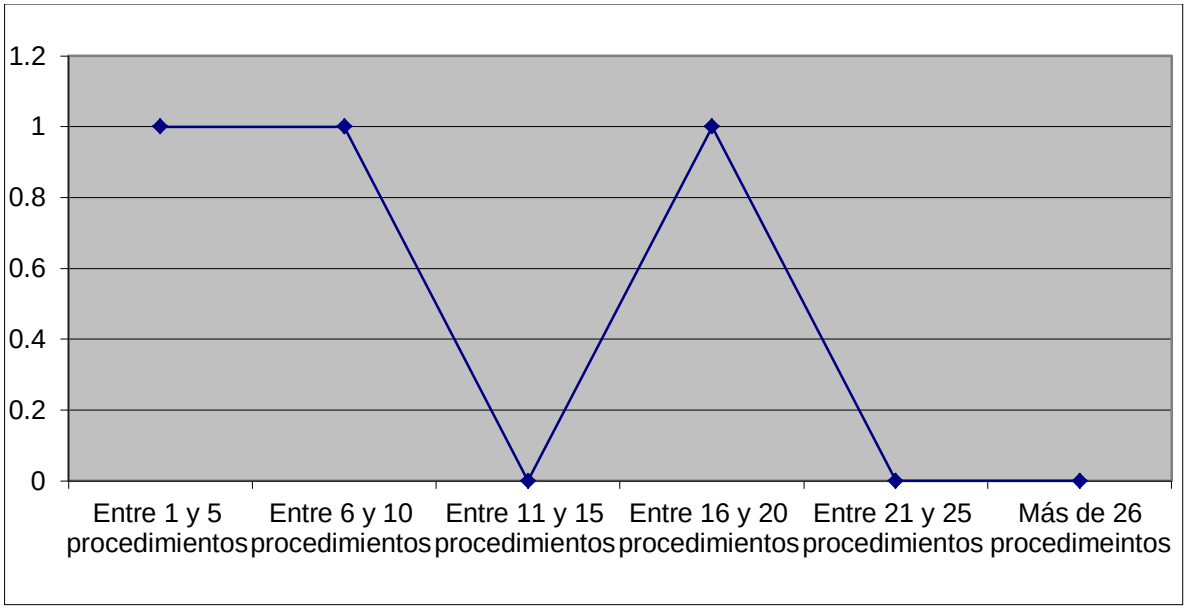


Figura: 23 Gráfica de los resultados de la evaluación de la métrica TOC y su influencia en los atributos de calidad (Complejidad de Implementación y Reutilización),

***Anexo 4: Caso de prueba de caja negra del caso de uso Gestionar componente**.*

1. Descripción General.

- Se realiza el registro, eliminación y actualización de un componente.

2. Secciones a probar en el Caso de Uso.

Nombre de la sección	Escenarios de la sección	Flujo central de eventos
S 1: Gestionar componente.	EC 1.1 Adicionar componente con éxito.	- El usuario selecciona la opción “Adicionar Componente”. - El sistema muestra la interfaz para Adicionar Componente. - El usuario introduce el nombre del componente y marca el checkbox “Componente” y selecciona la

Nombre de la sección	Escenarios de la sección	Flujo central de eventos
		opción “Aceptar”.
	EC 1.2 Cancelar Adicionar componente	- El usuario selecciona la opción “Adicionar Componente”. - El sistema muestra la interfaz para Adicionar Componente. - El usuario introduce el nombre del componente y marca el checkbox “Componente” y cierra la interfaz. - Finaliza la operación Adicionar Componente.
	EC 1.3 Adicionar componente con datos incorrectos.	- El usuario selecciona la opción “Adicionar Componente”. - El sistema muestra la interfaz para Adicionar Componente. - El usuario introduce el nombre del componente y marca el checkbox “Componente” y cierra la interfaz. - El sistema muestra el mensaje: “Nombre de componente o paquete incorrecto”. - El usuario introduce el nombre correcto y selecciona la opción “Aceptar”.
	EC 1.4 Actualizar Adicionar componente	- El usuario selecciona un componente y escoge la opción “Modificar Componente”. - El sistema muestra la interfaz para cambiar el nombre del componente. - El usuario introduce el nuevo nombre del componente y selecciona la opción “Aceptar”.
	EC 1.5 Cancelar	- El usuario selecciona un componente y escoge la opción “Modificar Componente”.

Nombre de la sección	Escenarios de la sección	Flujo central de eventos
	Actualizar Adicionar componente	- El sistema muestra la interfaz para cambiar el nombre del componente. - El usuario introduce el nuevo nombre del componente y cierra la interfaz. - Finaliza la operación Modificar Componente.
	EC 1.6 Actualizar Adicionar componente con datos incorrectos.	- El usuario selecciona un componente y escoge la opción “Modificar Componente”. - El sistema muestra la interfaz para cambiar el nombre del componente. - El usuario introduce el nuevo nombre del Componente. - El sistema muestra el mensaje: “Nombre de componente o paquete incorrecto”. - El usuario introduce el nombre correcto y selecciona la opción “Aceptar”.
	EC 1.8 Eliminar Regla semántica.	- El usuario selecciona un Componente y escoge la opción “Eliminar”.

3. Descripción de variable

No.	Nombre de campo	Clasificación	Valor Nulo	Descripción
[1]	Nombre del componente	Campo de texto	No	Se introduce el nombre del componente.

4. Clases validas e inválidas

Condición de entrada	Tipo	Clases validas	Clases inválidas
Nombre valido para el componente.	Valor	1. Cualquier nombre que no contenga numero, ni empiece con mayúsculas y sin caracteres extraños de ningún tipo.	2. Cualquier nombre que contenga numero, empiece con mayúsculas y contenga caracteres extraños de ningún tipo.

5. Matriz de Datos

Escenario	Clase de equivalencia	Nombre del componente	Respuesta del Sistema	Resultado de la Prueba
EC 1.1 Adicionar componente con éxito.	1	casoPrueba	El sistema adiciona el nuevo componente y actualiza el árbol de paquetes y componentes.	Satisfactorio

Escenario	Clase de equivalencia	Nombre del componente	Respuesta del Sistema	Resultado de la Prueba
EC 1.2 Cancelar Adicionar componente.		casoPrueba	El sistema cierra la ventana adicionar componente.	Satisfactorio
EC 1.3 Adicionar Regla semántica con datos incorrectos.	2	CasoPrueba2	El sistema al teclear el nombre, muestra un mensaje “Nombre de componente o paquete incorrecto”.	Satisfactorio
EC 1.4 Actualizar componente.	1	casoPruebaMod	El Sistema actualiza el componente y actualiza el árbol de paquetes y componentes.	Satisfactorio

Escenario	Clase de equivalencia	Nombre del componente	Respuesta del Sistema	Resultado de la Prueba
EC 1.5 Cancelar Actualizar componeente.		casoPruebaMod	El sistema cierra la ventana modificar componente.	Satisfactorio
EC 1.6 Actualizar componente con datos incorrectos.	2	CasoPruebaMod	El sistema al teclear el nombre, muestra un mensaje “Nombre de componente o paquete incorrecto”.	Satisfactorio
EC 1.8 Eliminar componeente.		NA	El Sistema elimina el componente y actualiza el árbol de paquetes y componentes.	Satisfactorio