Universidad de las Ciencias Informáticas

Facultad 3

Título: Herramienta para el análisis de código fuente sobre el marco de trabajo Sauxe.

Trabajo de Diploma para optar por el título de Ingeniero Informático

Autor: Evelio Alvarez Palma.

Tutor: Ing. Omar Antonio Díaz Peña.

Co-Tutor: Ing. Aneyvis Hernández Chinea.

La Habana, Junio del 2013.

"Año 55 de la Revolución"

Pensamiento

"LA INGENIERÍA NO ES SOLO UNA PROFESIÓN APRENDIDA, SINO ES UNA EN CONSTANTE APRENDIZAJE EN LA CUAL LOS PRACTICANTES PRIMERO FUERON ESTUDIANTES Y ASÍ SE MANTIENEN A LO LARGO DE SU CARRERA ACTIVA"

ANÓNIMO.

Declaración de autoría

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de 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 _____.

Evelio Alvarez Palma **Omar Antonio Díaz Peña**

_____ _____

Aneyvis Hernández Chinea

_____

Datos de contacto

Ing. Omar Antonio Díaz Peña:
Ingeniero en Ciencias Informáticas, Universidad de las Ciencias Informáticas.
Categoría Docente: Especialista.
Años de graduado: 3.
Arquitecto de Software del CEIGE.

Ing. Aneyvis Hernández Chinea:
Ingeniera en Ciencias Informáticas, Universidad de las Ciencias Informáticas.
Categoría Docente: Asistente.
Años de graduado: 6.
Administrador de Configuraciones del CEIGE.

Agradecimientos

Me gustaría mucho empezar agradeciendo a las personas que le dedico esta tesis, a mi madre Maritza, a mi padre Enrique y a mi hermano Enrique, gracias de todo corazón. Les agradezco mucho a mis abuelas Mima y Elsa que son muy especiales, a mis tías Martha y Zenaida, a mis tíos Rafael, Rubén y Manolo, todos incansables e indispensables en este logro. A mis primos Martica, Ailí, Maylín, Mailén, Raisel, Manolito, Rafelín que también confiaron en mí, incluyo a Andy en este grupo porque es como si fuera mi primo. A mi cuñada Mairelys cuya ayuda y experiencia en el medio informático me ha servido de mucho. A toda mi familia en general y a las amistades que han compartido con nosotros tristezas y alegrías, en especial a Isabel y familia que tuvieron que aguantarme durante 5 años en su casa, adoptándome como otro hijo, muchas gracias.

Le agradezco mucho a Omelio mi amigo y hermano de muchos años y por muchos años, gracias por incentivarme el tratar de ser mejor cada día. A mi otra amiga Lía muchas gracias amor por darme tu amistad y por confiar en mí. A Laura y Claudia, personas que me tienen una fe que ni yo mismo, también incansables en la tarea de darme ánimo. Por supuesto que les tengo que agradecer a sus respectivas familias que también me aportaron mucho.

Agradecimientos especiales por todo lo que me ayudaron, a Luis Angel, Wilfredo, Disnel, Marbelis y Karel. A los tutores, tanto Aneyvis como Omar, quienes tuvieron que batallar conmigo, cosa bastante difícil.

A la gente de mi aula, a todos, desde los que se fueron en los primeros años hasta los que cumplen este sueño conmigo, especialmente a; Alain, Jenny, Pedry, Jany, Lisbet, Dailin, Malena, Alejandro, Beatriz, Norge, Roider, Diogenes, Maité, Raidel, Camilo, Miladis, Alfredo, la China, Yudi, Maikel, César y Rocío.

A los integrantes de Miel con Limón que me ayudaron a realizar uno de mis grandes sueños (la música), no los olvidaré jamás. A todos los artistas aficionados de la universidad, en especial a Jose y Alex.

A las personas del edificio, que hemos pasado por tanto trabajo y tantas alegrías, no puedo dejar de mencionar a Aroldo, Yandi, Juan Carlos, David, Hernán, Ramón, Luis Carlos, el Tony y Helen. Por supuesto que a la gente del proyecto, especialmente a Leiser, Yasmany, Katia y Alejandro.

A las personas del barrio que me apoyaron cantidad y confiaron siempre en mí. A todos los mencionados anteriormente y si se me queda alguien, mil gracias también.

Dedicatoria

Le dedico esta tesis al tridente mágico, mi madre por dedicarme tanto tiempo y amor, a mi padre por todo el sacrificio realizado y por servirme de ejemplo a seguir, y a mi hermano por estar siempre brindándome su apoyo incondicional.

Resumen

Los desarrolladores de software juegan un papel fundamental en el avance tecnológico, social y cultural que han venido experimentando las personas desde hace varios años, debido a que con su esfuerzo y dedicación han creado programas que satisfacen disímiles necesidades. Para lograr una mejora en la calidad de estos se emplea a escala mundial el análisis de código fuente.

La Universidad de las Ciencias Informáticas cuenta dentro sus centros productivos con el Centro de Informatización de Gestión de Entidades (CEIGE), el cual utiliza Sauxe como marco de trabajo. De las aplicaciones que se están desarrollando con dicho marco se han identificado una serie de problemas en el código que impiden un mejor funcionamiento de estas, tales como: complejidad en la identificación de la dependencia de los componentes, baja cohesión entre las responsabilidades de las clases presentes en los componentes y desconocimiento del nivel de documentación del código fuente.

El presente trabajo consiste en desarrollar una herramienta que sea capaz de detectar estos problemas. El sistema tomará como entrada el proyecto a analizar, una vez cargado el mismo, en una estructura arbórea de sus módulos, le permitirá al usuario transitar por varios tipos de análisis de código fuente, los cuales pueden iniciarse en una complejidad simple que indicará la cantidad de clase PHP y JS que posee el módulo seleccionado, hasta una complejidad más avanzada que identificará la existencia de métodos repetidos utilizando como criterio de búsqueda el nombre de los mismos. Como puede apreciarse el sistema será de gran apoyo para la toma de decisiones, pues permitirá dotar de una herramienta muy útil al equipo de trabajo.

Palabras Claves: código fuente, calidad, análisis de código fuente.

Tabla de Contenidos

Índice de Tablas

Tabla de Ilustraciones

Introducción

Las denominadas Tecnologías de la Información y las Comunicaciones (TIC) ocupan un lugar primordial en el desarrollo de la sociedad. Su evolución ha permitido un avance tecnológico, cultural y social para las personas. Los desarrolladores de software tienen un papel fundamental en este avance, debido a que con su esfuerzo y dedicación han creado programas que satisfacen disímiles necesidades. Para lograr una mejora en la calidad de estos se emplea a escala mundial el análisis de código fuente; que no es más que una técnica que se aplica directamente sobre dicho código tal cual, sin transformaciones previas ni cambios de ningún tipo. La idea es que, en base a ese código fuente, se pueda obtener información que permita mejorar la base del código manteniendo la semántica original. (Expósito, 2011)

Cuba cuenta con varios profesionales e instituciones relacionadas con la rama de la informática y especializadas en el desarrollo del software, quienes pertenecen al Ministerio de las Comunicaciones (MC). La misión de este ministerio es impulsar el proceso de informatización de la sociedad cubana y para esto se ha trazado varias estrategias: una de estas es la creación de la Universidad de las Ciencias Informáticas (UCI), cuyo propósito es crear además de profesionales capacitados, software de alta calidad, confiables y certificados, que brinden un aporte a la economía y a la sociedad.

Uno de los centros de desarrollo de software de la universidad tiene como línea de producción la informatización de la Gestión de Entidades. En este centro se desarrollan varias aplicaciones, las cuales utilizan Sauxe como marco de trabajo, definido por la subdirección de producción ya que dicho marco apoya el desarrollo de aplicaciones web, además de contar con una gran variedad de marcos de trabajo que facilitan la implementación, como por ejemplo: Zend Framework, ExtJs, y Doctrine ORM.

A pesar de que estas aplicaciones desarrolladas se dividen en módulos para hacer más cómodo su desarrollo, hay muchas que debido al gran tamaño que poseen presentan una gran cantidad de código fuente, elemento fundamental para obtener el resultado deseado, debido a este aumento se han identificado una serie de problemas en el código que se muestran a continuación:

- Complejidad en la identificación de la dependencia de los componentes; pues no se puede conocer la dependencia que existe entre los mismos de manera directa.
- Baja cohesión entre las responsabilidades de las clases presentes en los componentes, debido a que existen clases y métodos repetidos realizando la misma función.

- Desconocimiento del nivel de documentación del código fuente, los métodos y clases no presentan una identificación sobre cuál es la función que están realizando.

Todos estos errores han influido en la ocurrencia de varias consecuencias que dificultan el correcto desempeño de las aplicaciones y la posibilidad de que estas aplicaciones sean analizadas. Las principales consecuencias son:

- Aumento de la complejidad espacial del código fuente; que no es más que la eficiencia de los algoritmos en lo relativo a su consumo de memoria.

- Disminución de la mantenibilidad de las soluciones; ya que es significativo para el mantenimiento de un sistema o componente de software lo bien documentado que se encuentre.

- Aumento del acoplamiento de los distintos componentes; ya que existe una menor cohesión, su intención y propósito no son claros, todo lo cual hace más difícil extraer comportamientos comunes y al reestructurar el código fuente descubrir componentes faltantes que puedan ser reutilizados.

Con lo planteado anteriormente se identificó como **problema a resolver:** El estado del código escrito con el marco de trabajo Sauxe, está dificultando la capacidad de ser analizado en las soluciones desarrolladas por el centro CEIGE.

Teniendo como **objeto de estudio**: Proceso de análisis de código estático.

Campo de Acción: Análisis del código de las aplicaciones desarrolladas con Sauxe en el centro CEIGE.

Idea a defender: Si se elabora una herramienta para el análisis de código fuente sobre el marco de trabajo Sauxe, se facilitará la capacidad de análisis de las soluciones desarrolladas en el centro CEIGE.

Se define como **objetivo general**: Desarrollar una herramienta para el análisis de código fuente sobre el marco de trabajo Sauxe que facilite la capacidad de análisis de las soluciones desarrolladas en el centro CEIGE.

Desglosado en los siguientes objetivos específicos:

- Elaborar el marco teórico para fundamentar la investigación.
- Definir los requisitos de la solución para satisfacer las necesidades planteadas.
- Diseñar la solución propuesta para potenciar la implementación del sistema.
- Implementar la solución propuesta para dar cumplimiento a los requisitos propuestos.
- Validar la solución desarrollada mediante la realización de pruebas de caja blanca y caja negra.

Las **tareas investigativas** que van a solucionar los objetivos previstos son:

- Estudio de los principales conceptos acerca del análisis de código fuente para obtener los conocimientos necesarios del servicio que se brindará.
- Estudio de las herramientas existentes en el mundo que permiten el análisis de código fuente.
- Especificación de los requisitos funcionales del componente.
- Realización del diagrama de clases del diseño.
- Implementación de la solución propuesta.
- Realización de la validación del componente.

Métodos teóricos de la investigación:

Analítico-Sintético: Este método fue empleado en el análisis de los documentos generados en el levantamiento y captura de requisitos, extrayendo y examinando los principales elementos relacionados con el objeto de estudio. Fue utilizado además en el análisis y selección del modelo, las herramientas y tecnologías a utilizar. También fue empleado en el análisis de soluciones existentes para determinar características comunes y problemas que estos poseen con el objetivo de determinar los elementos que puedan ser empleados en la solución en cuestión.

Histórico-Lógico: Este método se emplea para conocer los antecedentes y estado actual de los sistemas informáticos que realizan el análisis de código fuente, así como las causas de la evolución de los mismos.

Modelación: Posibilita la utilización de diagramas para expresar información de manera organizada. En este caso fue necesario el uso del método para establecer esquemas y modelar ideas de forma gráfica, ya que en el proceso de análisis y diseño el modelaje se convierte en una herramienta indispensable para el entendimiento de las diferentes etapas del desarrollo del software.

Método Empíricos:

Entrevista: En este caso se utiliza para el proceso de entendimiento de la idea a realizar y los requerimientos que el software deberá satisfacer, además del proceso de familiarización con el personal capacitado sobre el tema a tratar.

El documento está estructurado en 3 capítulos:

Capítulo 1: "Fundamentación Teórica": se incluye un estado del arte de la investigación, así como las metodologías, técnicas de programación y herramientas utilizadas para el desarrollo de la aplicación.

Capítulo 2: "Análisis y Diseño del Sistema": se brinda una visión del sistema, se definen los requisitos funcionales y no funcionales a los que se debe dar cumplimiento, se ofrece además la propuesta del sistema a desarrollar, también se realiza el diseño y descripción de la arquitectura del sistema, y se presentan además el diagrama de clases y los diagramas de diseño.

Capítulo 3: "Implementación y Prueba": se realizan los diagramas de despliegue y componentes como resultado de la implementación del sistema, además de las pruebas realizadas una vez concluido el desarrollo de la aplicación.

Capítulo 1: Fundamentación Teórica

1.1 Introducción

En el presente capítulo se analizarán los conceptos fundamentales relacionados con el tema, así como un estudio de las principales herramientas desarrolladas en el mundo para investigar cómo es que realizan el análisis de código fuente y compararlas entre ellas en cuanto a los resultados obtenidos por cada una. Se estudiarán las tecnologías y el modelo de desarrollo para la construcción de la herramienta.

1.2 Conceptos fundamentales relacionados con el código fuente

Código fuente:

1. El Ing. William Mercado lo define como un conjunto de líneas que conforman un bloque de texto, escrito según las reglas sintácticas de algún lenguaje de programación destinado a ser legible por humanos. Es un programa en su forma original, tal y como fue escrito por el programador, no es ejecutable directamente por el computador, debe convertirse en lenguaje de máquina mediante compiladores, ensambladores o intérpretes. (Mercado, 2007)

2. El Ing. José Camps definió que es un texto que se ha escrito en un lenguaje de programación concreto, y que sólo puede ser leído por un experto o programador. Estos caracteres deben ser traducidos a un lenguaje que se denomina código máquina, el cual podrá ejecutar cualquier ordenador. También puede ser traducido a un lenguaje llamado códigos de bytes, el cual podrá ser traducido por un intérprete. Este tipo de transferencias y traducciones se llaman compilación. (Camps, 2011)

3. La revista MasterMagazine plantea que es un conjunto de instrucciones que son redactadas por un usuario que tiene conocimientos del Lenguaje de Programación, y que son la base del código objeto, que es posteriormente utilizado por los dispositivos del sistema. (MasterMagazine, 2010)

Luego de conocer algunas definiciones de código fuente se decidió trabajar con la que plantea el Ing. William Mercado ya que refleja de una forma más clara y entendible el concepto estudiado.

1.3 Herramientas de análisis de código fuente

Las herramientas de análisis de código fuente son instrumentos automatizados que permiten examinar el código fuente de una aplicación con el objetivo principal de identificar errores que ayuden

a desarrollar un producto de alta calidad. Dichos instrumentos pueden ser propietarios o libres, a continuación se realiza un breve estudio tanto internacional como nacional sobre algunos de estos.

1.3.1 Herramientas de análisis de código fuente a nivel mundial

FxCop: Es una herramienta creada por Microsoft, analiza el código .NET y proporciona sugerencias para diseño, seguridad y mejoras de rendimiento. De forma predeterminada, FxCop analiza un ensamblado basado en las reglas establecidas por instrucciones de diseño para desarrollar bibliotecas de clases. Esta herramienta descubre errores en las áreas de seguridad, rendimiento, diseño de la biblioteca y localización. Además sirve de ayuda al equipo de desarrollo para seguir las mejores prácticas de la plataforma .NET. FxCop está disponible como una aplicación independiente; también incluye una implementación de la línea de comandos que facilita su conexión a un proceso de generación automática. Otra de las ventajas que posee es que su distribución es gratuita. Como desventajas se tiene que su actualización se hace en fases más largas de tiempo. (Mitchell, 2008)

PHP-SAT: Es una herramientas de análisis de código fuente creada por Eric Bouwers y Martin Bravenboer. Una de las características clave de PHP-Sat es la detección automática de diferentes tipos de vulnerabilidades, lo que le permite comprobar el código fuente para determinados fallos de seguridad. La herramienta define su funcionamiento en patrones, es decir, los que son correctos acorde a la gramática de PHP. La misma se basa en la documentación de PHP y en experiencias pasadas, además, reflejan información de posibles errores y el programador tiene que decidir cuándo un patrón constituye un error en una situación específica. Los patrones se clasifican en cuatro categorías: Corrección, Exposición de Información, Optimización y Estilo. La herramienta escribe en el mismo código analizado el resultado de su análisis. No existe para ella, actualmente, una versión estable. Su instalación es un tanto difícil, además, de la comprensión del código fuente. (EelcoVisser, 2006)

bugScout: Creada por Buguroo Offensive Security, es una herramienta diseñada para realizar el análisis del código fuente en aplicaciones. Su objetivo principal es identificar errores de seguridad y proponer acciones correctivas o mitigadoras para las vulnerabilidades encontradas, una de las más comunes que detecta bugScout es la denominada phishing, que no es más que el robo de credenciales. Permite gestionar con facilidad las vulnerabilidades, realizar informes, almacenar documentación y consultar estadísticas. Dentro de las ventajas de esta herramienta están: reduce el coste de las auditorías manuales en más de un 90% y el tiempo de espera en más de un 99%. Es una solución flexible que se adapta a las necesidades de cualquier empresa. Recibió el premio SIC 2011 al Servicio Gestionado de Análisis de Vulnerabilidades en Código Fuente. Como desventaja se

tiene que no es una herramienta libre y su licencia es costosa. (BUGUROO OFFENSIVE SECURITY S.L., 2011)

Pylint: Es una herramienta de análisis estático que tiene como fundamental objetivo detectar errores en aplicaciones desarrolladas en Python, así como también establecer un estándar de codificación. Ha tenido un desarrollo bastante continuo, empezó en el 2006 y hasta la actualidad se han desarrollado alrededor de 15 versiones. Muestra una serie de mensajes posterior al análisis donde se tienen en cuenta estadísticas tales como: número de advertencias y errores encontrados en los diferentes archivos. Esto último puede motivar a los desarrolladores a mejorar su código. Permite adicionar estilos de chequeos y su configuración es fácil. Pylint presenta una arquitectura ordenada en cuanto a sus analizadores, los cuales están habilitados por defecto. Posibilita adicionar chequeos adicionando plugins, entre los que podemos mencionar RawChecker y ASTNGChecker, compatible con cualquier versión superior a Python 2.2. Algunas de las revisiones que realiza esta herramienta al código son: revisión de variables, de clases, de diseño, de formato, conflictos entre viejo/nuevo estilo, etc. (Pylint, 2006)

Rough Auditing Tool for Security (RATS): Es una herramienta para escanear el código fuente de las aplicaciones escritas en C, C++, PHP y Python con el fin de encontrar errores de seguridad. Desarrollada originalmente por Software Security Inc, esta utiliza una base de datos para encontrar sentencias del código que pueden ser una vulnerabilidad potencial, al encontrar uno de estos provee la descripción del mismo y sugiere una solución al respecto. Además, indica el grado de severidad del problema encontrado para facilitarle al auditor la asignación de prioridades y ejecutar un análisis básico para descartar condiciones que no causan problemas de seguridad. Uno de los errores más frecuentes que detecta esta herramienta es el desbordamiento de buffer. El análisis con RATS no es completo ya que no encuentra todos los errores existentes y puede destacar como errores elementos correctos. (Filipiak, 2009)

1.3.2 Herramientas de análisis de código fuente en Cuba

En la Universidad de las Ciencias Informáticas se ha trabajado en el área del análisis de código fuente, a continuación se podrán observar algunos de estos trabajos:

Sistema de Análisis Estático de Vulnerabilidades. Módulo PHP:

Este sistema tiene como objetivo principal detectar las vulnerabilidades del código fuente en aplicaciones o scripts desarrollados en el lenguaje de programación PHP de forma flexible y amigable mediante una interfaz Web. Este sistema brinda la posibilidad de analizar tanto un proyecto como un

sencillo script, además de brindar reportes de forma detallada y en un formato entendible al usuario. De igual forma, posibilita la generación de reportes estadísticos, en sentido general de los resultados de los análisis realizados. Estos reportes informan acerca de cantidad de vulnerabilidades encontradas, cantidad de alertas y funciones malignas. El sistema detecta vulnerabilidades en el código que pueden ser utilizadas por los atacantes para causar daño, algunas de las acciones que puede utilizar estos últimos son: forzar la entrada a áreas administrativas que normalmente requerirían una contraseña, ataques de tipo XSS, que no es más que el código maligno fue previamente guardado por el atacante en la base de datos que utiliza el sitio víctima. De esta forma, cada vez que el sitio recupere y muestre registros de su base de datos, el script maligno estará allí y será ejecutado para cualquier usuario visitante. (Adrián Hernández Yeja, 2010)

Sistema de Análisis Estático de Vulnerabilidades en Python:

Este sistema tiene como objetivo mejorar la calidad con que se entregan los proyectos realizados en el lenguaje de programación Python en la Universidad de las Ciencias Informáticas. De esta forma se crea un sistema que utiliza herramientas de análisis estático de código para eliminar los problemas de seguridad, utilizando buenas prácticas de programación como son: control del desbordamiento de buffer, establecimiento de control de acceso, privilegios mínimos y criptografía segura. La herramienta permite cargar lo mismo ficheros que fragmentos de código, los cuales serán analizados y de los cuales se generará un reporte con las vulnerabilidades encontradas que luego podrá ser descargado por el usuario. En los reportes generados se puede observar información acerca de posibles excepciones causadas por falta de coincidencia de tipos, atributos que no se encuentran, u otros tipos de excepciones planteadas por cada función, cuántos módulos se analizaron, cuántas líneas de código en total, el posible tipo de retorno de la función, en donde fue llamada dicha función, etc. (Teudis Naranjo Ortiz, 2010)

Plataforma para el Análisis Estático de Vulnerabilidades del Código

La plataforma permite analizar aplicaciones en varios lenguajes de programación mediante el análisis estático. Además, brinda la posibilidad de incluir otras herramientas de validación de código fuente, posibilitando la corrección del código en aplicaciones de diferentes lenguajes. Una vez subido el fichero, el sistema debe seleccionar el servidor adecuado para ejecutar la herramienta que realizará el análisis pertinente, actualizar en tiempo real el estado de su análisis, brindar además los resultados de las vulnerabilidades obtenidas y permitir cancelar un determinado análisis en curso. Dicho sistema debe ser capaz de analizar tanto un proyecto como un script. Aclarar que esta versión de la plataforma solo tiene incluido el análisis de vulnerabilidades para el código PHP utilizando la herramienta “Sistema de Análisis Estático de Vulnerabilidades. Módulo PHP”, por lo que los

resultados que la plataforma va a devolver son los mismos que devuelve el sistema anteriormente mencionado, con la particularidad de que a la plataforma se le pueden añadir sistemas para analizar código de otros lenguajes. (Yaritza Montero Vaillant, 2011)

1.3.3 Valoración de las herramientas estudiadas.

Tabla 1: Características de las herramientas estudiadas.

Herramientas	Indicadores		
	Lenguajes	Tipo de análsis	Licencia
FxCop	C#, Visual Basic, C++, Python	Descubrir errores en las áreas de seguridad, rendimiento, diseño de la biblioteca y localización.	CPOL
PHP-SAT	PHP	Detección de diferentes tipos de vulnerabilidades, para determinar fallos de seguridad.	LGPL
bugScout	Java, C#, Visual, Basic, PHP	Identificar errores de seguridad y proponer acciones correctivas o mitigadoras para las vulnerabilidades encontradas.	CLUF
Pylint	Python	Detección de errores, revisión de código y establecimiento de un estándar de codificación.	GPL
RATS	C, C++, PHP, Python	Identificar errores de seguridad.	GPL
Sistema de Análisis Estático de Vulnerabilidades. Módulo PHP	PHP	Detectar vulnerabilidades del código para mejorar la seguridad.	UCI
Sistema de Análisis Estático de Vulnerabilidades en	Python	Detectar vulnerabilidades del código para eliminar los problemas de seguridad.	UCI

Python			
Plataforma para el Análisis Estático de Vulnerabilidades del Código	PHP	Detectar vulnerabilidades del código para mejorar la seguridad.	UCI

Luego de estudiar las principales herramientas que realizan el análisis de código fuente en el mundo y en Cuba se tiene como resultado que:

- Ninguna presenta la posibilidad de analizar código JS.
- Ninguna analiza las dependencias que existen entre los componentes, el nivel de documentación del código fuente y las responsabilidades entre las clases presentes en los componentes.
- Ninguna se desarrolló para soportar las características que presenta el marco de trabajo Sauxe tales como: la organización de carpetas y la sentencia integrator para identificar las dependencias entre los componentes.

Por lo que se concluye que no se emplearán estas herramientas en la realización de la presente investigación debido a que es más factible realizar una solución propia que cumpla con las necesidades antes mencionadas.

1.4 Técnicas de captura de requisitos estudiadas

La obtención de requisitos es el proceso mediante el cual los interesados en un sistema de software descubren sus requisitos utilizando técnicas que posibilitan que el proceso de recopilación de requisitos se realice de forma más eficiente. (Toro, 2000) De las ténicas existentes de captura de requisitos se analizaron las siguientes:

Entrevistas: técnica de captura de requisitos más utilizada, son prácticamente inevitables en cualquier desarrollo ya que son una de las formas de comunicación más naturales entre personas. Estas se pueden identificar en tres fases: preparación, realización y análisis. Para cada una de estas fases previamente mencionadas se llevan a cabo una serie de acciones que logren la obtención de resultados por fases. Una vez realizada la entrevista es necesario leer las notas tomadas, pasarlas a limpio, reorganizar la información, contrastarla con otras entrevistas o fuentes de información, etc.

Una vez elaborada la información, se puede enviar al entrevistado para confirmar los contenidos. También es importante evaluar la propia entrevista para determinar los aspectos mejorables. (Toro, 2000)

Joint Application Development: la técnica denominada JAD (Desarrollo Conjunto de Aplicaciones), es una alternativa a las entrevistas individuales que se desarrollan a lo largo de un conjunto de reuniones en grupo durante un período de 2 a 4 días. En estas reuniones se ayuda a los clientes y usuarios a formular problemas y explorar posibles soluciones, involucrándolos y haciéndolos sentirse partícipes del desarrollo. Esta técnica se base en cuatro principios: dinámica de grupo, el uso de ayudas visuales para mejorar la comunicación (diagramas, transparencias, multimedia, herramientas CASE, etc.), mantener un proceso organizado y racional, y una filosofía de documentación WYSIWYG (What You See Is What You Get, lo que se ve es lo que se obtiene), por la que durante las reuniones se trabaja directamente sobre los documentos a generar. Debido a las necesidades de organización que requiere y a que no suele adaptarse bien a los horarios de trabajo de los clientes y usuarios, esta técnica no suele emplearse con frecuencia, aunque cuando se aplica suele tener buenos resultados. (Toro, 2000)

Brainstorming: también conocida como tormenta de ideas, es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios. Las sesiones de brainstorming suelen estar formadas por un número de cuatro a diez participantes, uno de los cuales es el jefe de la sesión, encargado más de comenzar la sesión que de controlarla. Esta técnica puede ayudar a generar una gran variedad de vistas del problema y a formularlo de diferentes formas, sobre todo al comienzo del proceso, cuando los requisitos son todavía muy difusos, además tiene la ventaja de que es muy fácil de aprender y requiere poca organización. La tormenta de ideas se divide en 4 fases, esenciales para el proceso de obtención de los requisitos, tales como: Preparación, Generación, Consolidación y Documentación. (Toro, 2000)

1.5 Modelo de desarrollo de software estudiado

El proceso de desarrollo de un producto de software es difícil de controlar, si no se lleva un modelo de desarrollo por el cual apoyarse. Para la presente investigación se decidió la utilización del Modelo de Desarrollo del CEIGE el cual propone una serie de harramientas y lenguajes que posibilitarán la realización de aplicaciones.

El modelo de desarrollo está propuesto por la Subdirección de Producción del CEIGE, describe una secuencia de actividades para desarrollar soluciones mostrando los artefactos correspondientes a

medida que se vaya construyendo el software. Un factor importante es que este modelo incluye el desarrollo basado en componentes; lo que admite reutilizar piezas de código pre-elaborado que permiten realizar diversas tareas, conllevando a diversos beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo, el mayor retorno sobre la inversión, simplicidad en el mantenimiento del sistema y simplicidad en las pruebas. Mediante el uso de varios componentes simples se pueden construir proyectos bastante complejos los cuales si se realizaran desde el principio tomarían demasiado tiempo. (Terreros, 2011)

De los disímiles artefactos que genera este modelo, para esta investigación se decidió la utilización solo de algunos artefactos que eran imprescindibles para la realización de la herramienta. En la fase del Estudio preliminar no se generó ningún artefacto ya que no eran necesarios para el proceso de construcción del software. Ya en la fase de Desarrollo, en la disciplina de Modelado del negocio se generó el documento modelo conceptual, por ser de vital importancia debido a que la herramienta no presenta proceso de negocio. En la disciplina de Requisitos se generó el artefacto: especificación de requisitos de software, el cual sirve de guía para el desarrollo de la aplicación. Se generaron los artefactos, diagrama de clases del diseño, diagrama de interacción y diseño de casos de prueba como parte de la disciplina Análisis y diseño. En la disciplina de Implementación se generaron artefactos que no exigía el modelo de desarrollo pero era necesarios para un mejor entendimiento de la herramienta, estos fueron: diagrama de componentes y diagrama de despliegue. Se generó el acta de liberación en la disciplina de Pruebas de liberación para certificar el correcto funcionamiento de dicha harramienta.

1.6 Lenguaje de modelado

UML (Lenguaje Unificado de Modelado): Es un lenguaje para la especificación, visualización, construcción y documentación de los artefactos de un proceso de sistema intensivo. Es utilizado para la modelación de sistemas con características orientadas a objetos. Facilita a los integrantes de un equipo multidisciplinario participar e intercomunicarse fácilmente, promueve la reutilización e incrementa la capacidad de lo que se puede hacer con otros métodos de análisis y diseño orientados a objetos. (Cornejo, 2010)

1.7 Lenguaje de programación

Java: Es un lenguaje de programación puntero en cuanto al desarrollo de herramientas y aplicaciones de negocio. Es rápido, seguro, multiplataforma, fiable, su distribución es gratuita, brinda soporte a las técnicas de desarrollo orientada a objetos y permite la reutilización de componentes de software. Otro de los factores más importantes de Java es que no se quiebra fácilmente ante errores

de programación, es muy difícil que permita escribir en áreas arbitrarias de memoria ni realizar operaciones que corrompan el código. Puede aplicarse a la realización de aplicaciones en las que ocurra más de una función a la vez, lo que se denomina multihilos. (Oracle Corporation, 2011)

1.8 Herramienta CASE para el modelado del sistema

CASE: Las herramientas de Ingeniería de Software Asistida por Ordenador (Computer Aided Software Engineering, CASE por sus siglas en inglés) son sistemas de software que intentan proporcionar ayuda automatizada a las actividades del proceso de software. Estas herramientas tienen vital importancia en múltiples aspectos del ciclo de vida del desarrollo del software, como son: mejora la productividad en el mantenimiento y desarrollo del software, mejora del tiempo, costo de desarrollo y mantenimiento de los sistemas informáticos y aumenta la calidad del software. (Rudder, 2011)

Visual Paradigm 8.0 for UML: Es una herramienta de diseño UML y herramienta CASE UML diseñado para ayudar al desarrollo de software. Es compatible con los equipos de desarrollo de software en la captura de requisitos, planificación del software, código de la ingeniería, modelado de clases, modelado de datos, etc. (Visual Paradigm International, 2010) Ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas. Esta herramienta está enfocada al negocio que genera un software de mayor calidad, es fácil de instalar y actualizar, presenta la posibilidad de generar código para el lenguaje Java y utiliza un lenguaje estándar común para todo el equipo de desarrollo que facilita la comunicación.

1.9 Herramienta de programación

IDE (Integrated Development Environment): Un entorno de desarrollo integrado es un entorno de programación que ha sido empaquetado como un programa de aplicación, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDE pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes, estos ofrecen un marco de trabajo amigable para la mayoría de los lenguajes de programación, gran parte de estos IDE son multiplataforma, depuradores y reconocedores de sintaxis. (Oracle Corporation, 2011)

NetBeans 7.0: Es una herramienta que permite de una forma rápida y fácil desarrollar aplicaciones Java de escritorio, móviles y aplicaciones web utilizando la plataforma Java, es gratuito y de código abierto, y tiene una gran comunidad de usuarios y desarrolladores de todo el mundo. Proporciona soporte de primera clase para las últimas tecnologías Java y las mejoras más recientes antes de

otros IDE. NetBeans además contiene un depurador que permite colocar puntos de interrupción en el código fuente, añadir relojes de campo, pasos a través del código, dirigido a los métodos y a tomar instantáneas. Puede ser instalado con comodidad en cualquier sistema operativo y puede trabajar sin problemas con aplicaciones que realicen varias tareas a la misma vez, los denominados multihilos. (Oracle Corporation, 2011)

1.10 Estilo arquitectónico

De acuerdo al Software Engineering Institute (SEI), la Arquitectura de Software se refiere a las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos. Los elementos pueden ser entidades que existen en tiempo de ejecución (objetos, hilos), entidades lógicas que existen en tiempo de desarrollo (clases, componentes) y entidades físicas (nodos, directorios). (Cervantes, 2011)

Por capas: La arquitectura basada en capas se enfoca en la distribución de roles y responsabilidades de forma jerárquica proveyendo una forma muy efectiva de separación de responsabilidades. El rol indica el modo y tipo de interacción con otras capas, y la responsabilidad indica la funcionalidad que está siendo desarrollada. Esta arquitectura presenta una alta cohesión ya que cada capa contiene funcionalidad directamente relacionada con la tarea de dicha capa. Las capas inferiores no tienen ninguna dependencia con las capas superiores, permitiéndoles ser reutilizables en otros escenarios. Como beneficios, dicha arquitectura al distribuir las capas entre múltiples sistemas (físicos) puede incrementar la escalabilidad, la tolerancia a fallos y el rendimiento, también presenta una mejora en las pruebas ya que tiene interfaces bien definidas para cada capa, permite aislar los cambios en tecnologías a ciertas capas para reducir el impacto en el sistema total y las capas permiten que se realicen cambios en un nivel abstracto. (Pelaez, 2009)

1.11 Conclusiones parciales

Finalizado el estudio de este capítulo se pudo llegar a las conclusiones siguientes:

- ✓ Al realizar un estudio de las herramientas de análisis de código fuente en el mundo se demostró la necesidad de desarrollar una solución que cumpla con las necesidades que se requieren.
- ✓ Se describen los lenguajes, las herramientas y el modelo de desarrollo utilizados para la guiar el proceso de construcción del software, ajustándose a lo definido por el CEIGE.

Capítulo 2: Análisis y Diseño del Sistema

2.1 Introducción

En el presente capítulo se especifican puntos importantes para llevar a cabo la implementación de la herramienta, comenzando por el levantamiento de los requisitos funcionales y no funcionales, lo que permite dar paso al diseño donde se realizan los diagramas de clases, modelo conceptual y diagramas de interacción.

2.2 Propuesta del sistema

Para darle solución a la problemática que se presenta, se propone la realización de una herramienta informática que permitirá analizar el código fuente de un proyecto, con el objetivo de detectar problemas existentes en su código. El sistema tomará como entrada el proyecto a analizar, una vez cargado el mismo, en una estructura arbórea de sus módulos, le permitirá al usuario transitar por varios tipos de análisis del código fuente, los cuales pueden iniciarse en una complejidad simple que indicará la cantidad de clase PHP y JS que posee el módulo seleccionado, hasta una complejidad más avanzada que identificará las dependencias entre componentes. Como puede apreciarse el sistema será de gran de ayuda para la toma de decisiones, pues permitirá dotar de una herramienta muy útil al equipo de trabajo.

2.3 Modelo conceptual

Explica cuáles son y cómo se relacionan los conceptos relevantes y sus atributos en la descripción del problema. Este modelo captura los tipos más importantes de objetos en el contexto del sistema, los cuales representan los elementos que existen o los eventos que suceden en el entorno en el que trabaja el sistema. (López, Dailyn Sosa, 2011) A continuación se muestra en la Ilustración 1: Modelo conceptual. el modelo conceptual de esta investigación.

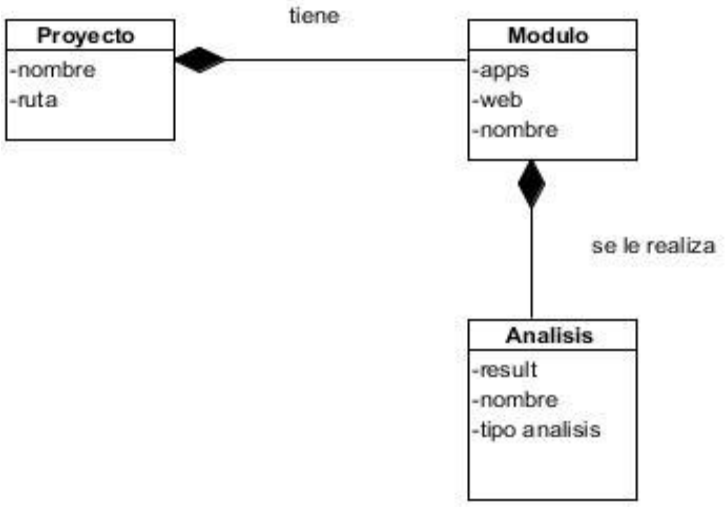

Ilustración 1: Modelo conceptual.

2.4 Técnica de captura de requisitos

Para la presente investigación se dicidió la utilización del brainstorming o tormenta de ideas para la identificación de los requisitos debido a que seleccionaron en el centro CEIGE un grupo de especialistas a ser partícipes en la realización de la técnica. De este grupo se eligió al Msc. Osmar Leyet Fernandez como responsable de la sesión en la que se reunieron todos estos profesionales con el objetivo de generar ideas, de las cuales se eligieron las más significativas y que daban respuesta a una situación problémica, estas ideas seleccionadas vinieron a conformar los requisitos del sistema. Para finalizar, los requisitos identificados fueron recogidos en un documento por el responsable de la sesión.

2.5 Especificación de los requisitos de software

El proceso de captura de requisitos tiene gran importancia en el desarrollo del software, ya que a través de estos se identifica lo que se desea y de esta forma se obtiene un producto de calidad.

Los requisitos se pueden clasificar en:

- ✓ Funcionales.
- ✓ No Funcionales.

2.5.1 Requisitos funcionales

Los requisitos funcionales no son más que capacidades o condiciones que el sistema debe cumplir, estos especifican comportamientos particulares de un sistema. En la presente investigación se identificaron 6 requisitos que darán solución a la aplicación, los cuales de describen a continuación:

RF1. Realizar análisis de código fuente.

Descripción del RF1

Precondiciones	Que se hayan descargado del repositorio el código fuente del proyecto a analizar.
Flujo de eventos	
Flujo básico	
1	Se carga el proyecto de la dirección donde este se encuentre descargado.
2	El sistema muestra el proyecto cargado en forma arbórea.
3	Se escoge él(los) módulo(s) que se va(n) a analizar.
4	El sistema activa el menú Análisis.
5	Se selecciona el tipo de análisis se desea realizar: -Mostrar la cantidad de clases .php y .js existentes. -Mostrar la dependencia entre componentes.

	-Mostrar los métodos que no estén comentados. -Mostrar los métodos que se repiten. -Mostrar el tamaño operacional.
6	El sistema muestra una ventana para escribir el nombre del análisis que se va a realizar.
7	El usuario escribe el nombre y presiona el botón analizar.
8	El sistema valida los datos introducidos.
9	El sistema muestra una barra de progreso durante la realización del análisis.
10	Se muestra el análisis guardado en la aplicación con los siguientes datos: - Nombre. -Tipo. -Fecha. -Módulos.
11	El usuario selecciona el análisis realizado de la tabla de Análisis Realizados.
12	El sistema muestra en la ventana de Salida la información del análisis seleccionado.
13	Concluye el requisito.
Pos-condiciones	
1	El sistema muestra los resultados obtenidos.
Flujos alternativos	
Flujo alternativo *.a El usuario cancela la acción	
1	Concluye el requisito.
Pos-condiciones	
1	No se realiza el análisis.
Flujo alternativo 7.a El usuario elige la opción Cancelar	
1	Concluye el requisito.
Pos-condiciones	
1	No se realiza el análisis.
Flujo alternativo 8.a Datos incompletos	
1	El sistema informa la existencia de datos vacíos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 7 del flujo básico.
Pos-condiciones	
	N/A

Flujo alternativo 8.a Datos repetidos		
1	El sistema informa la existencia de datos repetidos y permite corregirlos.	
2	El usuario corrige los datos.	
3	Volver al paso 7 del flujo básico.	
Pos-condiciones		
	N/A	
Validaciones		
1	Se validan los datos según lo establecido en el Modelo conceptual	
Relaciones	**Requisitos Incluidos**	N/A
	Extensiones	N/A
Conceptos	**Análisis**	Nombre del Análisis.
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

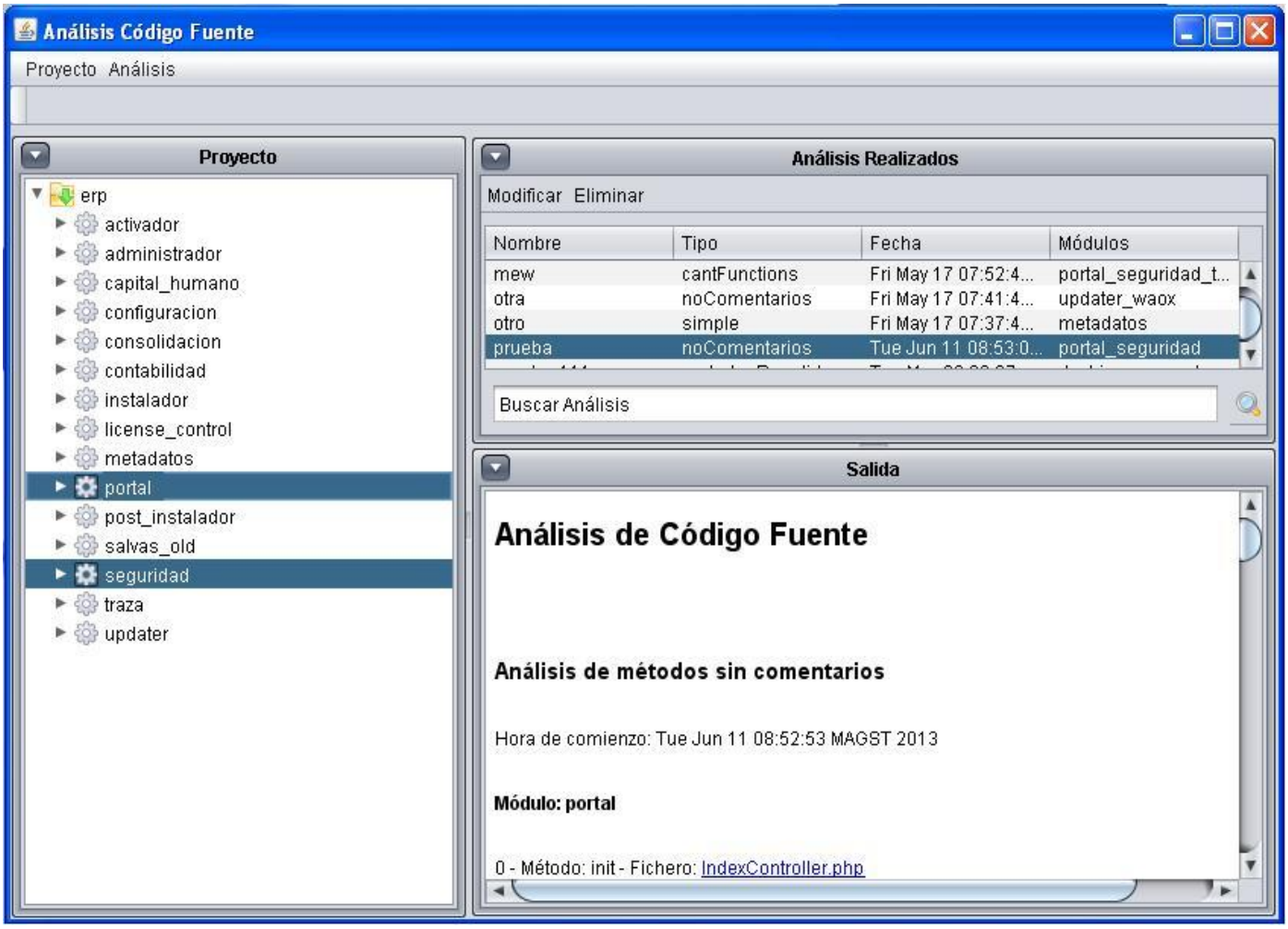

Ilustración 2: Prototipo de interfaz de usuario "Realizar análisis de código fuente".

RF2. Listar análisis de código fuente.

Descripción del RF2

Precondiciones	Se haya registrado al menos un análisis de código fuente en el sistema.
Flujo de eventos	
Flujo básico	
1	El sistema muestra un listado con los análisis del código fuente guardados en la aplicación.
2	Concluye el requisito.
Pos-condiciones	
	N/A
Flujos alternativos	
Flujo alternativo	
	N/A

Pos-condiciones		
N/A		
Validaciones		
N/A		
Relaciones	**Requisitos Incluidos**	N/A
	Extensiones	N/A
Conceptos	N/A	N/A
Requisitos especiales	Son los requisitos no funcionales específicos para el requisito.	
Asuntos pendientes	Posibles mejoras al requisito.	

Ilustración 3: Prototipo de interfaz de usuario "Listar análisis de código fuente".

RF3. Buscar análisis de código fuente.

Descripción del RF3

Precondiciones	Se haya registrado al menos un análisis de código fuente en el sistema.
Flujo de eventos	
Flujo básico	
1	El usuario escribe el nombre del análisis que desea encontrar en el campo de

	texto.	
2	El sistema valida los datos introducidos.	
3	Si los datos son correctos el sistema señala cual es el análisis que concuerda con el criterio de búsqueda.	
4	Concluye el requisito.	
Pos-condiciones		
	N/A	
Flujos alternativos		
Flujo alternativo 3.a Datos incompletos		
1	El sistema señala los datos vacíos y permite corregirlos.	
2	El usuario corrige los datos.	
3	Volver al paso 2 del flujo básico.	
Pos-condiciones		
1	N/A	
Flujo alternativo *.a El usuario cancela la acción		
1	Concluye el requisito.	
Pos-condiciones		
1	No se realiza la búsqueda.	
Flujo alternativo 4.a No existen datos que cumplan con los criterios especificados		
	El sistema notifica que no existen datos que cumplan con los criterios especificados desmarcando el análisis que se encuentre marcado.	
Pos-condiciones		
	N/A	
Validaciones		
	N/A	
Relaciones	**Requisitos Incluidos**	N/A
	Extensiones	N/A
Conceptos	**Análisis**	Nombre
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

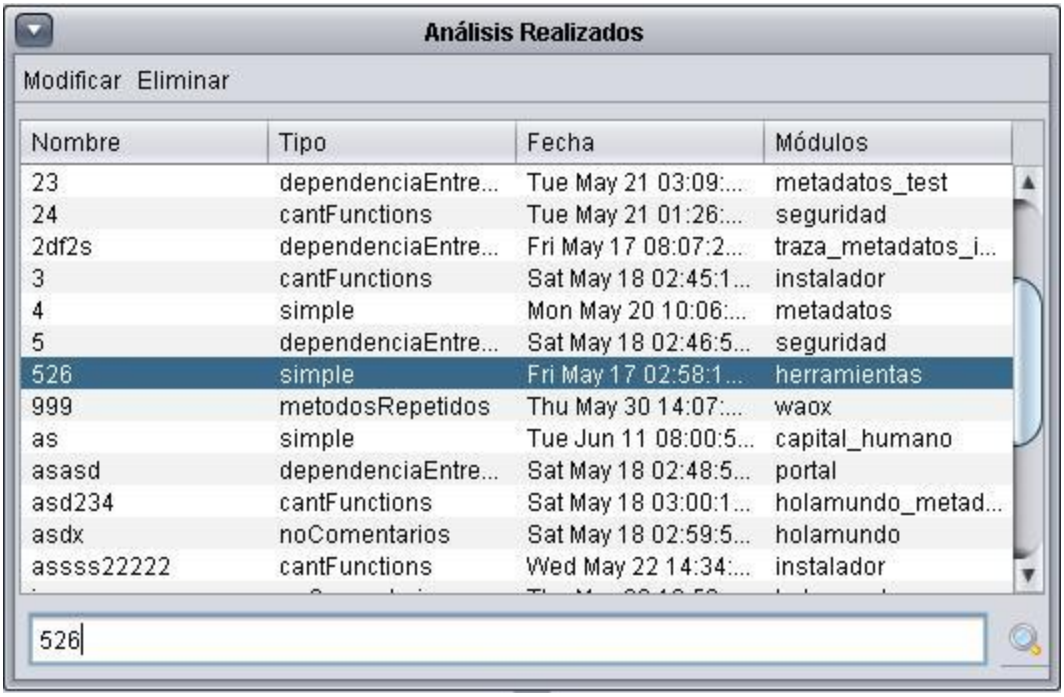

Ilustración 4: Prototipo de interfaz de usuario "Buscar análisis de código fuente".

RF4. Vista previa.

Descripción del RF3

Precondiciones	Se haya registrado al menos un análisis de código fuente en el sistema.	
Flujo de eventos		
Flujo básico		
1	Se selecciona el análisis de la lista de análisis guardados en el sistema.	
2	El sistema muestra la información perteneciente al análisis seleccionado en la ventana de salida.	
3	Concluye el requisito.	
Pos-condiciones		
	Se muestra la información perteneciente al análisis seleccionado.	
Flujos alternativos		
	N/A	
Validaciones		
N/A		
Relaciones	**Requisitos Incluidos**	N/A
	Extensiones	N/A

Conceptos	N/A	N/A
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

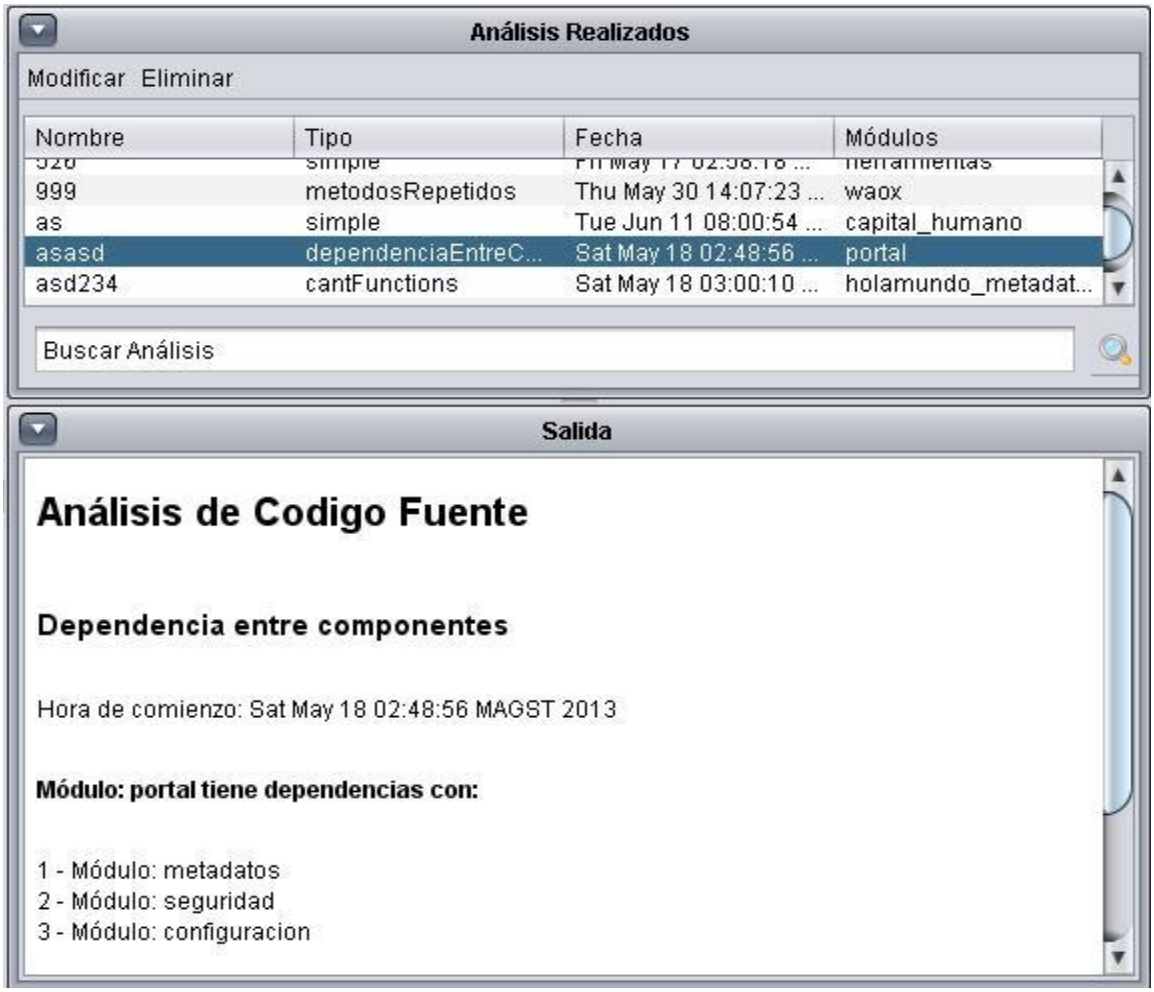

Ilustración 5: Prototipo de interfaz de usuario "Vista Previa".

RF5. Eliminar análisis de código fuente.

(Ver Anexo 1)

RF6. Modificar análisis de código fuente.

(Ver Anexo 2)

2.5.2 Requisitos no funcionales

Son aquellos requisitos que no se refieren directamente a las funciones específicas que entrega el sistema, sino a las propiedades que este debe tener. Se identificaron varios requisitos no funcionales, estos son:

Usabilidad

RNF1. El sistema debe ser fácil de utilizar para los usuarios que tengan niveles básicos de computación.

Rendimiento

RNF2. Los tiempos de respuesta y velocidad de procesamiento de la información no serán mayores de 5 segundos para cargar un proyecto, eliminar, buscar y modificar un análisis, y de 3 horas para realizar los análisis.

Portabilidad

RNF3. La herramienta desarrollada deberá ser multiplataforma teniendo un correcto funcionamiento tanto en Linux como en Windows.

Hardware

RNF4. Como requerimientos mínimos que se necesitan para el correcto funcionamiento de la aplicación tenemos:

- Memoria RAM de 1GB en adelante.

- Disco duro de 80GB o mayor.

- Microprocesador a 2.0 GHz.

Software

RNF5. La computadora donde se utilice la herramienta deberá tener instalada la máquina virtual de Java.

Apariencia e Interfaz

RNF6. La aplicación debe contar con un diseño sencillo y explícito que permita al usuario interactuar con el sistema sin necesitar un entrenamiento profundo, debe ser un diseño formal con toda la seriedad que una aplicación de este tipo requiere.

2.6 Diagrama de clases del diseño

El diagrama de clase es el diagrama principal del diseño y análisis para un sistema. En él, se describen gráficamente las especificaciones de las claces de software y las interfaces mediante

relaciones entre clases y estructuras de herencia. Durante el análisis del sistema, el diagrama se desarrolla buscando una solución ideal. Durante el diseño, se usa el mismo diagrama, y se modifica para satisfacer los detalles de las implementaciones. (Thames, 2011) El diagrama de esta investigación se muestra a continuación en la Ilustración 6: Diagrama de clases del diseño..

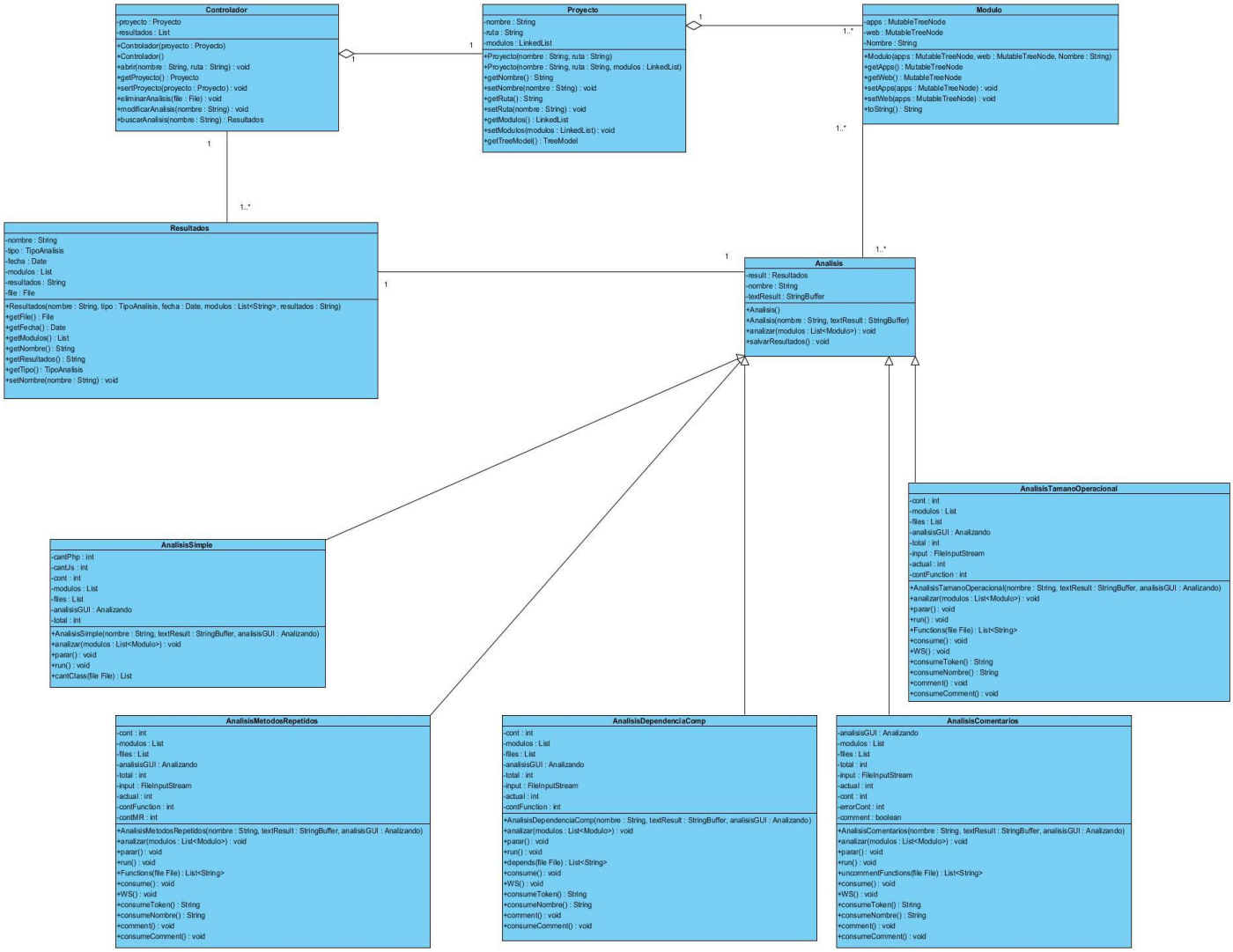

Ilustración 6: Diagrama de clases del diseño.

2.7 Validación de los requisitos y del diseño

Para realizar la validación de los requisitos funcionales que presenta el sistema fueron empleados los prototipos de interfaz de usuario y las pruebas de aceptación, estas últimas consisten en revisar con el cliente los requisitos funcionales, los cuales ya fueron validados y se emitió un documento dejando constancia de los hechos, firmado por el Jefe de Departamento de la Subdirección de Producción.

Para validar el requisito no funcional rendimiento se emplearon las pruebas de rendimiento, la cual constó de tres iteraciones, seleccionando para su aplicación el proyecto ERP desarrollado sobre el

marco de trabajo Sauxe. El tipo de análisis utilizado en cada una de las iteraciones fue el análisis de métodos repetidos. En la primera iteración se analizaron todos los módulos de dicho proyecto, lo cual devolvió un tiempo de respuesta de una hora y 27 minutos. Luego, en la segunda iteración se decidió duplicar la cantidad de módulos, el tiempo de respuesta que se obtuvo fue de dos horas y 24 minutos. Para la tercera de estas iteraciones se decidió reducir la cantidad de módulos existentes a la mitad, devolviendo un tiempo de respuesta de 45 minutos. Estos resultados permitieron validar el requisito no funcional de rendimiento debido a que los tiempos de respuestas fueron menores de los propuestos en un inicio. Además, queda validada la variable capacidad de análisis debido a que antes de la implementación de la herramienta, las revisiones que se realizaban podían tomar varios días y al utilizar la herramienta el tiempo de respuesta es de hasta tres horas, por lo que la capacidad de análisis aumenta.

Para validar el diseño se seleccionó la métrica TOC (Tamaño Operacional de Clases), la cual realiza una evaluación por clase de la presencia de atributos de calidad como: Responsabilidad, Complejidad de implementación y Reutilización.

- ✓ 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.

A continuación, algunas ilustraciones que muestran los resultados obtenidos al aplicar dicha métrica.

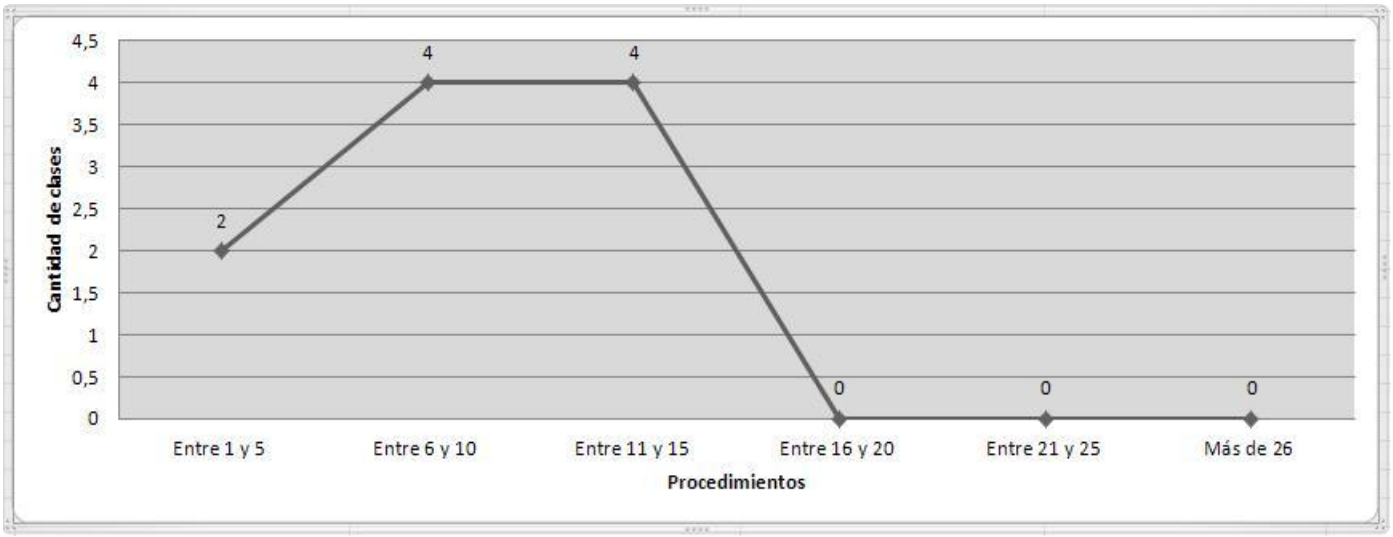

Ilustración 7: Representación de las clases según la cantidad de operaciones.

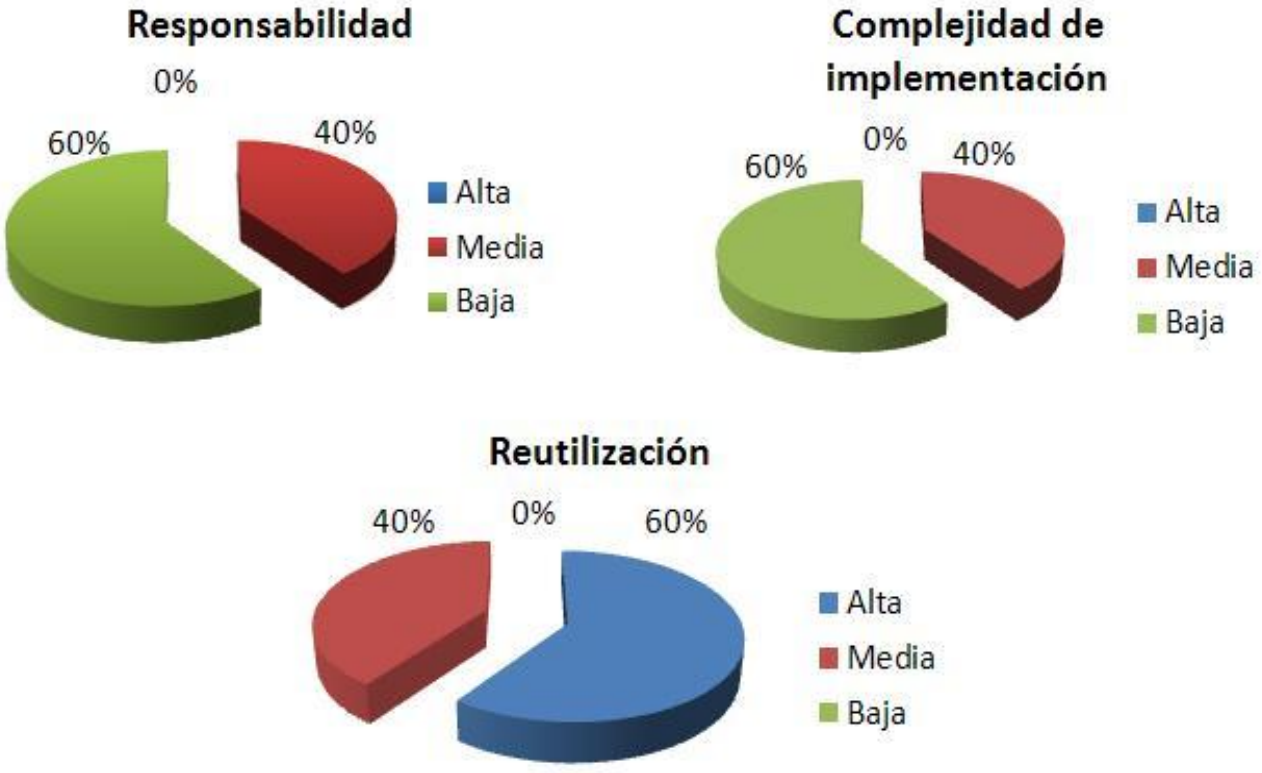

Ilustración 8: Resultados de la evaluación de la métrica TOC en los 3 atributos.

Resultados Obtenidos

Como resultados se obtuvieron que de un total de 10 clases, el 60% de estas presentan una baja Responsabilidad y Complejidad de implementación, y una alta Reutilización. El 40% restante se comportó en la media de los 3 atributos de calidad anteriormente mencionados por lo que se puede concluir que el sistema presenta una correcta distribución operacional por clases.

2.8 Diagramas de interacción

Los diagramas de interacción se utilizan para modelar los aspectos dinámicos de un sistema. Muestran una interacción, que consiste de un conjunto de objetos y sus relaciones, incluyendo los mensajes que puedan ser realizados entre ellos. Son importantes para construir sistemas ejecutables a través de ingeniería directa e ingeniería inversa. Los diagramas de interacción están conformados por los diagramas de secuencia y los diagramas de colaboración. (Avila, 2009) En el presente trabajo se decidió utilizar el primero de los diagramas antes mencionados, a continuación se muestra el diagrama de secuencia del requisitos Realizar análisis de código fuente, el resto se podrá observar en el Anexo 3.

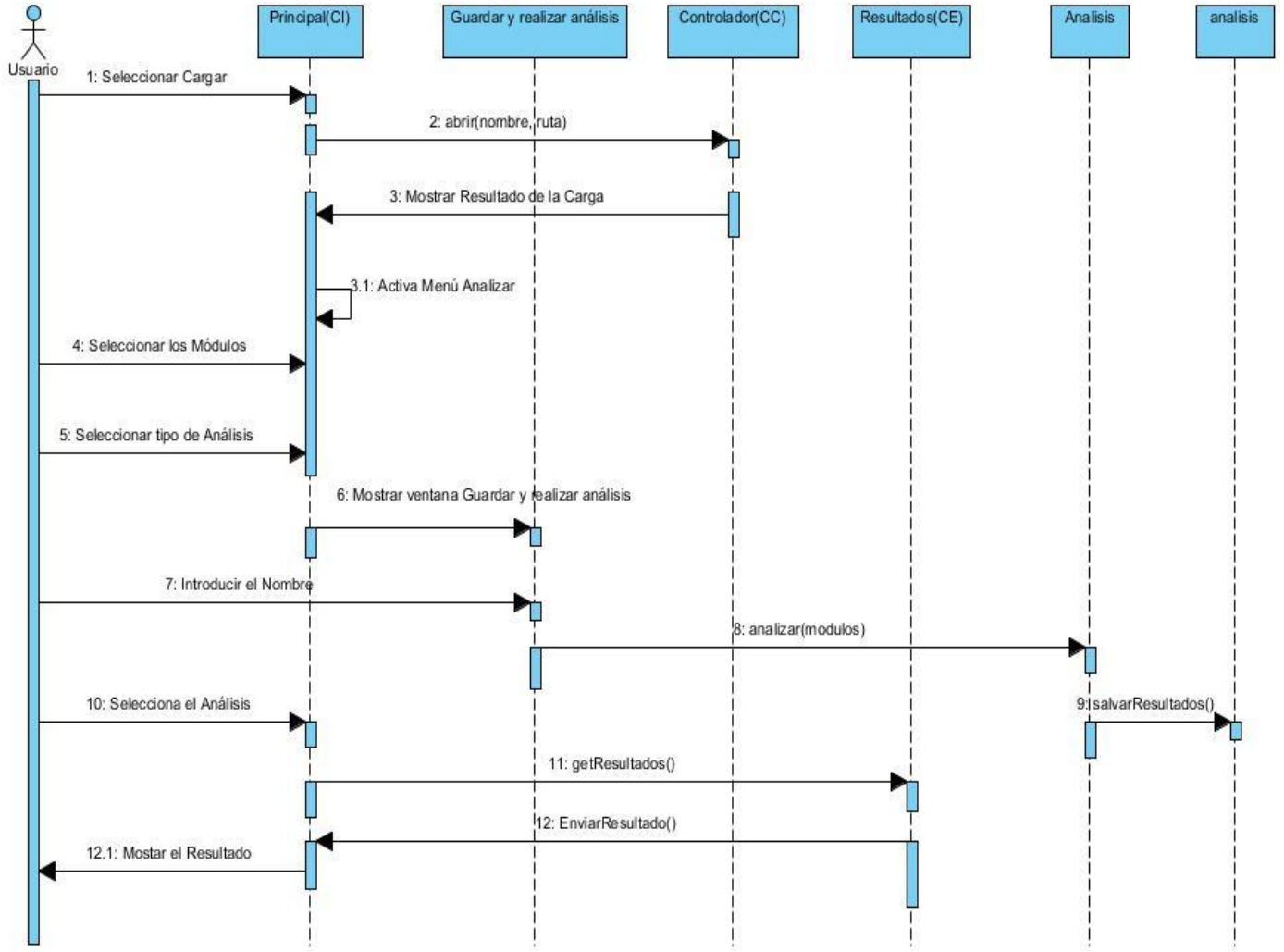

Ilustración 9: Diagrama de secuencia "Realizar análisis de código fuente".

2.9 Patrones de diseño

El diseño de la solución fue elaborado siguiendo patrones que constituyen soluciones simples y elegantes a problemas específicos y comunes del diseño. En este caso se emplearon los patrones siguientes:

Alta Cohesión: Este se identifica en la clase AnalisisSimple debido a que esta última realiza una labor única dentro del sistema no desempeñada por el resto como es la de realizar un análisis simple.

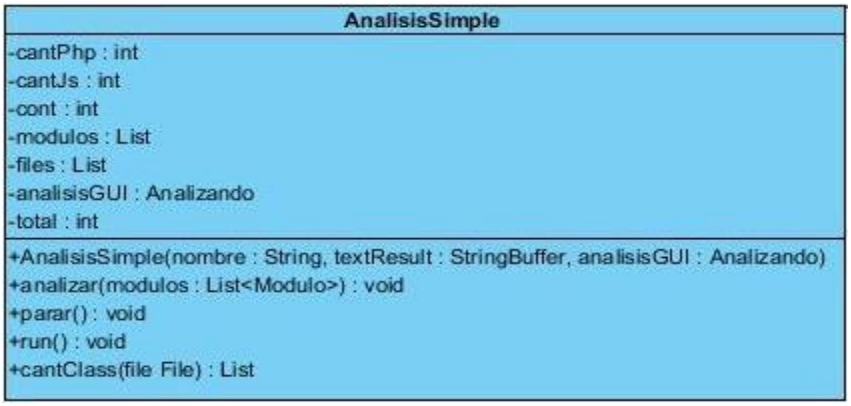

Ilustración 10: Presencia del patrón Alta Cohesión.

Experto: Dicho patrón está presente en las clases AnalisisSimple, AnalisisMetodosRepetidos, AnalisisDependeciaComp, AnalisisComentarios, AnalisisTamanoOperacional debido a que cada una de estas clases contiene los atributos y métodos que le son suficientes para realizar la función que tienen encomendada.

Bajo Acoplamiento: Al presentar el diseño pocas dependencias entre las clases se refleja la presencia de este patrón.

Creador: Entre las clases Controlador y Proyecto se identifica la presencia de este patrón a consecuencia de que la primera de las antes mencionadas presenta la posibilidad de crear un objeto de la clase Proyecto debido a que la contiene.

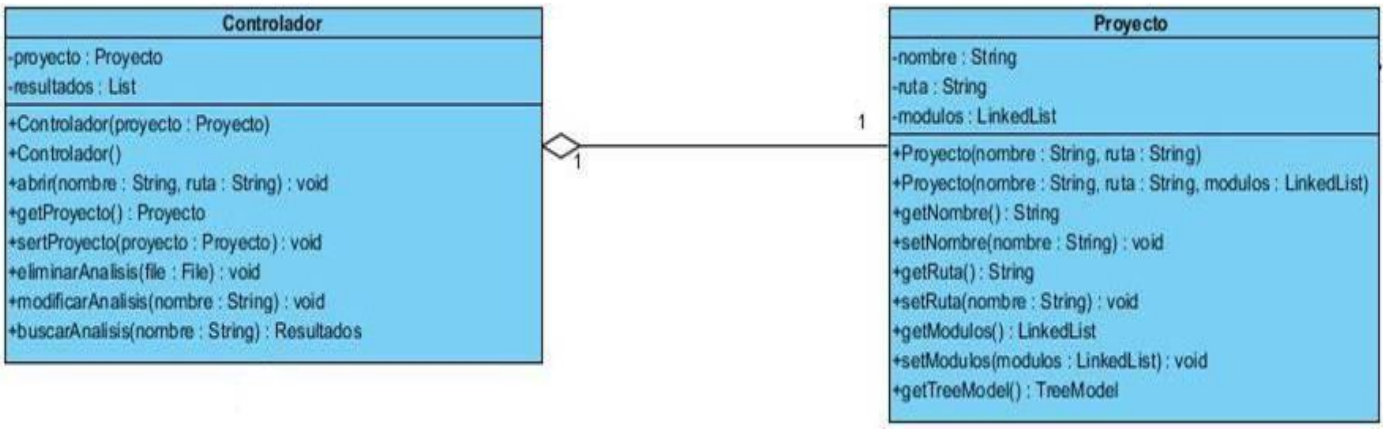

Ilustración 11: Presencia del patrón Creador.

Controlador: Este patrón está presente en el diseño debido a que la clase Controlador delega en la clase Analisis la responsabilidad de analizar manteniendo con esta última una relación fuerte.

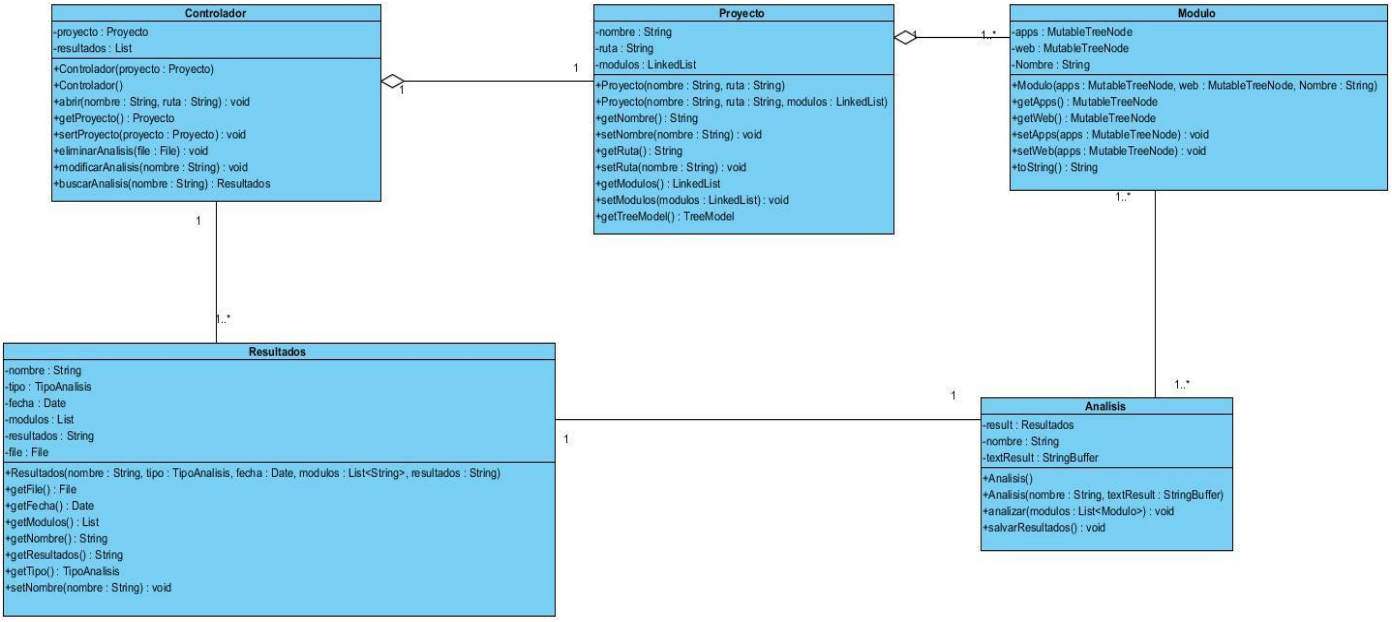

Ilustración 12: Presencia del patrón Controlador.

2.10 Conclusiones parciales

Al finalizar la realización del presente capítulo se obtuvieron las siguientes conclusiones:

- ✓ Quedaron definidos los requisitos que serán objeto de informatización y la descripción de los mismos, lo que permitió sentar las bases para las restantes fases de diseño e implementación.
- ✓ Se obtuvo el diagrama de clases correspondiente al sistema, permitiendo una cobertura total de los requisitos del mismo.
- ✓ Se realizó la validación del diseño mediante la aplicación de la métrica TOC, obteniendo como resultado que las clases del diseño presentan una baja Responsabilidad y Complejidad de implementación, y una alta Reutilización.
- ✓ Se utilizaron patrones de diseño, los cuales permitieron la elaboración del diagrama de clases del diseño y de diagramas de secuencias más fáciles de comprender.

Capítulo 3: Implementación y Prueba

3.1 Introducción

Una vez realizado el análisis y diseño del sistema se propuso en el presente capítulo llevar a cabo la implementación del mismo. Para ello se modelarán los correspondientes diagramas de despliegue y de componentes, conformando los modelos de implementación. Una vez concluida esta fase, se le realizarán diferentes pruebas a la aplicación, imprescindibles para el proceso de validación de la misma, dentro de estas pruebas están las de caja negra, las cuales se le realizan a las interfaces del sistema y las de caja blanca, aplicadas al código.

3.2 Diagrama de componentes

En el diagrama de componentes se muestra cómo el sistema está conformado por componentes y las dependencias entre ellos. Además proveen una vista arquitectónica de alto nivel del sistema. Ayuda a los desarrolladores a visualizar el camino de la implementación. Permite tomar decisiones respecto a las tareas de implementación. (Alva, 2008)

Ilustración 13: Diagrama de componentes.

3.3 Diagrama de despliegue

Un diagrama de despliegue modela la arquitectura en tiempo de ejecución de un sistema. Esto muestra la configuración de los elementos de hardware (nodos) y muestra cómo los elementos y artefactos del software se trazan en esos nodos. (Sparx Systems, 2007) En la Ilustración 14: Diagrama de despliegue. se muestra el diagrama de despliegue del sistema.

Ilustración 14: Diagrama de despliegue.

3.4 Tratamiento de errores

Un aspecto importante a tener en cuenta a la hora de desarrollar un software, es el tratamiento de errores, debido a que muchos son los errores cometidos por los usuarios, unos accidentalmente y otros no tan accidentales. Para garantizar el correcto funcionamiento del sistema, es fundamental identificar y controlar los posibles problemas que se pueden presentar a la hora de interactuar con el software.

En la implementación del presente trabajo se realizaron diferentes validaciones, las cuales se llevan a cabo en varias clases, con el uso de funciones propias del lenguaje. En caso de que exista algún error, es notificado al usuario para que pueda corregirlos. Además, se realizaron validaciones en las vistas, con el objetivo de garantizar la entrada correcta de los datos a la aplicación, evitando que se introduzcan datos vacíos en los diferentes campos de las ventanas. A continuación se muestran imágenes del código, donde se evidencian algunas validaciones presentes en el sistema.

En la Ilustración 15: Fragmento del código donde se tratan errores. se puede observar la validación siguiente: a la hora de ordenar la realización del análisis tiene que haber algún módulo seleccionado.

```
private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
    if (!trProyecto.getSelectionModel().isSelectionEmpty()) {
        List<Modulo> modulos = new LinkedList<Modulo>();
        for (int i = 0; i < trProyecto.getSelectionModel().getSelectionCount(); i++) {
            modulos.add((Modulo) ((DefaultMutableTreeNode) trProyecto.getSelectionModel().getSelectionPaths()[i]
        }
        AnalisisWizard analisis = new AnalisisWizard(TipoAnalisis.simple, modulos, this, false);
        analisis.setVisible(true);
        updateTable();
        updateProyecto();
    } else {
        JOptionPane.showMessageDialog(this, "No hay módulos seleccionados.", "Información", JOptionPane.INFORMAT
    }
}
```

Ilustración 15: Fragmento del código donde se tratan errores.

En la Ilustración 16: Fragmento del código donde se tratan errores. se puede observar otra de las validaciones, en este caso consiste en que a la hora de buscar un análisis no se puede dejar el campo del nombre vacío.

```
private void buscar() {
    for (int i = actual; i < tableModel.getRowCount(); i++) {
        String nombre = ((String) tableModel.getValueAt(i, 0)).toLowerCase();
        String texto = jTextField1.getText().toLowerCase();
        if (jTextField1.getText().isEmpty()) {
            JOptionPane.showMessageDialog(this, "No se puede dejar el nombre vacio.", "Información", JOptionPane
            break;
        }
        if (nombre.contains(texto.subSequence(0, texto.length()))) {
            jTable1.clearSelection();
            DefaultListSelectionModel list = new DefaultListSelectionModel();
            list.addSelectionInterval(i, i);
            jTable1.scrollRectToVisible(new Rectangle(0, (i + 1) * jTable1.getRowHeight(), 0, 0));
            jTable1.setSelectionModel(list);
            actual = i + 1;
            return;
        } else {
            jTable1.clearSelection();
        }
    }
    actual = 0;
}
```

Ilustración 16: Fragmento del código donde se tratan errores.

En la Ilustración 17: Fragmento del código donde se tratan errores mediante el uso de try-catch. se muestra un fragmento de código, donde se realizan validaciones con la utilización de instrucciones try catch, para el manejo de excepciones y errores. En este caso, específicamente a la hora de cargar un proyecto, pues para realizar esta operación se necesita que el proyecto a cargar esté desarrollado sobre Sauxe.

```
private void cargarProyecto(String nombre, String ruta) {
    try {
        IOConfig.setConfig(editor, ruta);
        controlador.abrir(nombre, ruta);
        proyecto = controlador.getProyecto();
        trProyecto.setModel(proyecto.getTreeModel());
        updateProyecto();

    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this, "Proyecto no válido. Asegúrese seleccionar un proyecto desarrollado
        try {
            IOConfig.setConfig(editor, "");
        } catch (FileNotFoundException ex1) {
            Logger.getLogger(Principal.class.getName()).log(Level.SEVERE, null, ex1);
        } catch (IOException ex1) {
            Logger.getLogger(Principal.class.getName()).log(Level.SEVERE, null, ex1);
        }
    }
}
```

Ilustración 17: Fragmento del código donde se tratan errores mediante el uso de try-catch.

3.5 Pruebas al sistema

El único instrumento adecuado para determinar el estatus de la calidad de un producto de software es el proceso de pruebas. En este proceso se ejecutan pruebas dirigidas a componentes del software o

al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos. (PRUEBASDESOFTWARE, 2005) Las pruebas que se aplicarán en esta investigación para validar la aplicación serán:

- ✓ Pruebas de caja negra.
- ✓ Pruebas de caja blanca.

3.5.1 Pruebas de caja negra

Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, por lo que los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa se mantiene. Esta prueba examina algunos aspectos del modelo, fundamentalmente del sistema, sin tener mucho en cuenta la estructura interna del software. (UCI, 2010)

Para desarrollar la prueba de caja negra existen varias técnicas entre las cuales están:

- ✓ Partición Equivalente.
- ✓ Análisis de Valores Límite.
- ✓ Grafos de Causa-Efecto.
- ✓ Pruebas de Comparación.

La técnica de caja negra Partición Equivalente fue la que se decidió escoger para aplicarla al sistema. Esta divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Dicha técnica se dirige a una definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. (UCI, 2010)

A continuación se mostrará la aplicación de la técnica de caja negra Partición Equivalente sobre algunos casos de prueba, el resto de casos de prueba se pueden revisar en el Anexo 4.

Caso de prueba de caja negra "Realizar análisis de código fuente"

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1: Realizar análisis de código fuente.	Se le realiza un análisis a él(los) módulo(s)	EP 1.1: Realizar un análisis a él(los) módulo(s)	Se carga el proyecto de la dirección donde este se encuentre descargado. Se escoge él(los) módulo(s) que se

seleccionado(s).	seleccionado(s).	va(n) a analizar. Se selecciona que tipo de análisis se desea realizar. Se inserta el nombre para guardar el análisis y se presiona el botón Analizar. Se presiona el análisis realizado de la tabla de Análisis Realizados
	EP 1.2: Realizar un análisis a él(los) módulo(s) seleccionado(s) dejando el campo vacío.	Se carga el proyecto de la dirección donde este se encuentre descargado. Se escoge él(los) módulo(s) que se va(n) a analizar. Se selecciona que tipo de análisis se desea realizar. Se deja el campo del nombre vacío y se presiona el botón Analizar.
	EP 1.3: Realizar un análisis a él(los) módulo(s) seleccionado(s) introduciendo datos ya existentes.	Se carga el proyecto de la dirección donde este se encuentre descargado. Se escoge él(los) módulo(s) que se va(n) a analizar. Se selecciona que tipo de análisis se desea realizar. Se introduce en el campo del nombre uno que ya se encuentre guardado y se presiona el botón Analizar.
	EP 1.4: Cancelar	Se carga el proyecto de la dirección donde este se encuentre descargado. Se escoge él(los) módulo(s) que se va(n) a analizar. Se selecciona que tipo de análisis se desea realizar. Se inserta o no el nombre para guardar el análisis y se presiona el botón Cancelar.

Tabla 2: Caso de prueba de caja negra "Realizar análisis de código fuente".

Escenario de prueba	Descripción general	Respuesta del Sistema	Resultado de la prueba
EP 1.1	Realizar un análisis a él(los) módulo(s) seleccionado(s).	El sistema realiza el análisis y lo registra en la tabla de Análisis Realizados.	*Satisfactoria*
EP 1.2	Realizar un análisis a él(los) módulo(s) seleccionado(s) dejando el campo vacío.	El sistema muestra un mensaje informando: "No se puede dejar el nombre vacío."	*Satisfactoria*
EP 1.3	Realizar un análisis a él(los) módulo(s) seleccionado(s) introduciendo datos ya existentes.	El sistema muestra un mensaje informando: "El fichero "(nombre)" ya existe. ¿Desea reemplazarlo?"	*Satisfactoria*
EP 1.4	Cancelar	El sistema no realiza el análisis, cierra la ventana de analizar.	*Satisfactoria*

Caso de prueba de caja negra "Modificar análisis de código fuente"

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1:Modificar análisis de código fuente.	Se modifica un análisis, modificando el campo: Nombre	EP 1.1: Modificar un análisis introduciendo datos válidos.	Se selecciona el análisis que se desea modificar. Se presiona el botón Modificar. Se insertan los datos. Se presiona el botón Aceptar.
		EP 1.2: Modificar un	Se selecciona el análisis

	análisis dejando el campo requerido en blanco.	que se desea modificar. Se presiona el botón Modificar. No se introduce los datos dejando el campo requerido en blanco. Se presiona el botón Aceptar.
	EP 1.3: Modificar un análisis introduciendo nombres ya existentes	Se selecciona el análisis que se desea modificar. Se presiona el botón Modificar. Se introduce en el campo del nombre uno que ya se encuentre guardado. Se presiona el botón Aceptar.
	EP 1.4: Cancelar	Se selecciona el análisis que se desea modificar. Se presiona el botón Modificar. Se introducen, o no, datos. Se presiona el botón Cancelar.

Tabla 3: Caso de prueba de caja negra "Modificar análisis de código fuente".

Escenario de prueba	Descripción general	Respuesta del Sistema	Resultado de la prueba
EP 1.1	Modificar un análisis introduciendo datos válidos.	El sistema modifica el nombre del análisis.	*Satisfactoria*
EP 1.2	Modificar un análisis dejando el campo	El sistema muestra un mensaje informando:	*Satisfactoria*

	requerido en blanco.	"No se puede dejar el nombre vacío."	
EP 1.3	Modificar un análisis introduciendo nombres ya existentes	El sistema muestra un mensaje informando: "Este nombre ya está registrado, introduzca un nombre nuevo."	*Satisfactoria*
EP 1.4	Cancelar	El sistema no modifica el nombre del análisis, cierra la ventana de modificar.	*Satisfactoria*

Caso de prueba de caja negra "Eliminar análisis de código fuente"

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1: Eliminar análisis de código fuente.	Se elimina un análisis.	EP 1.1: Eliminar un análisis.	Se selecciona el análisis que se desea eliminar. Se presiona el botón Eliminar. Se presiona el botón Sí del mensaje de confirmación.
		EP 1.2: Cancelar	Se selecciona el análisis que se desea eliminar. Se presiona el botón Eliminar. Se presiona el botón No del mensaje de confirmación.

Tabla 4: Caso de prueba de caja negra "Eliminar análisis de código fuente".

Escenario de prueba	Descripción general	Respuesta del Sistema	Resultado de la prueba

EP 1.1	Eliminar un análisis.	El sistema muestra un mensaje de confirmación: "¿Está seguro que desea eliminar el análisis?", el sistema elimina el análisis.	*Satisfactoria*
EP 1.2	Cancelar	El sistema no elimina el análisis, cierra la ventana de eliminar.	*Satisfactoria*

Como resultado de las pruebas de caja negra que se le realizaron a la aplicación se detectaron 3 no conformidades en una primera iteración, de estas, 2 fueron de ortografía y una de interfaz, las cuales quedaron corregidas, impidiendo su detección en una segunda iteración, logrando que la herramienta estuviese libre de no conformidades y lista para ser utilizada.

3.5.2 Pruebas de caja blanca

Se basa en el minucioso examen de los detalles del procedimiento. Se comprueban los caminos lógicos del software, proponiendo casos de prueba que examinen si están correctas todas las condiciones y todos los bucles, para determinar si el estado real coincide con el esperado o afirmado. Esto genera gran cantidad de caminos posibles, por lo que hay que dedicar esfuerzos a la determinación de las condiciones de prueba que se van a verificar. (UCI, 2010)

Las principales técnicas de diseño de pruebas de caja blanca son:

- ✓ Pruebas de Flujo de Control.
- ✓ Pruebas de Flujo de Datos.
- ✓ Pruebas de Bifurcar Ramas (branch testing).
- ✓ Pruebas del Camino Básico.

Se tomó la decisión de utilizar como técnica de caja blanca la de Prueba del Camino Básico. Esta se basa en obtener una medida de la complejidad del diseño procedimental de un programa (o de la lógica del programa). Esta medida es la complejidad ciclomática de McCabe, y representa un límite superior para el número de casos de prueba que se deben realizar para asegurar que se ejecuta cada camino del programa. (UCI, 2010)

Los pasos a realizar para aplicar esta técnica son:

- ✓ Representar el programa en un grafo de flujo.
- ✓ Calcular la complejidad ciclomática.
- ✓ Determinar el conjunto básico de caminos independientes.
- ✓ Derivar los casos de prueba que fuerzan la ejecución de cada camino.

El grafo de flujo se utiliza para representar flujo de control lógico de un programa. Para ello se utilizan los tres elementos siguientes:

- ✓ Nodos: representan cero, una o varias sentencias en secuencia. Cada nodo comprende como máximo una sentencia de decisión (bifurcación).
- ✓ Aristas: líneas que unen dos nodos.
- ✓ Regiones: áreas delimitadas por aristas y nodos. Cuando se contabilizan las regiones de un programa debe incluirse el área externa como una región más.

A continuación se mostrarán algunos ejemplos de la prueba de camino básico que se le realizaron a algunas funcionalidades:

Caso de prueba de caja blanca "Salvar resultados"

```
public void salvarResultados() throws FileNotFoundException, IOException {
    File file = new File("analisis/" + nombre + ".ana");              (1)
  boolean aux = true;                                                 (1)
  if (file.exists()) {                                               (2)
    if (JOptionPane.showConfirmDialog(null, "El fichero \"" + nombre
       + "\".ana ya existe.\n¿Desea reemplazarlo?", "Analisis de Código Fuente",
       JOptionPane.YES_NO_OPTION) == JOptionPane.NO_OPTION) {   (3)
      aux = false;                                                   (4)
    }
  }
  if (aux) {                                                         (5)
    result.setFile(file);                                            (6)
    ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(file));
    out.writeObject(result);
    out.close();
  }
```

} (7)

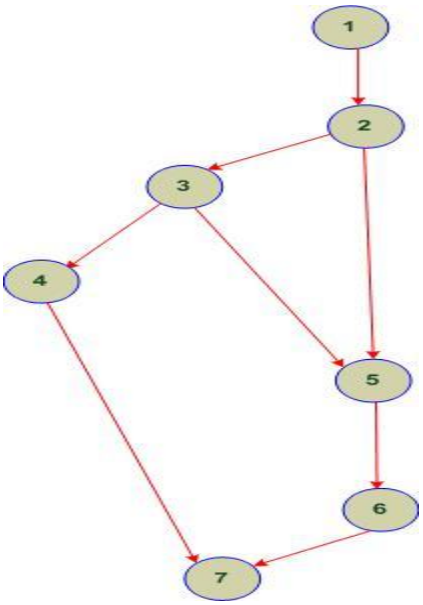

Ilustración 18: Grafo de flujo del caso de prueba "Salvar resultados".

Complejidad ciclomática => V(G)

Nodos => N

Aristas => A

V(G) = A – N + 2

V(G) = 8 – 7 + 2

V(G) = 3

Resultado de la prueba:

Existen 3 caminos linealmente independientes:

Camino 1: 1 2 5 6 7

Camino 2: 1 2 3 4 7

Camino 3: 1 2 3 5 6 7

Tabla 5: Caso de prueba de caja blanca "Salvar resultados".

No. de caminos	Caso de prueba	Objetivo	Resultado esperado
1	file.exists()==false aux == true	Verificar que el archivo no exista anteriormente para impedir que se repita con otro.	Al archivo no existir anteriormente, este es creado y salvado en la aplicación.
2	file.exists() == true aux = false	Verificar que si el archivo existe, y el	No se salva el archivo

		usuario decide no guardarlo, no se realice la operación.	
3	file.exists() == true aux = false	Verificar que si el archivo existe, y el usuario decide guardarlo, se realice la operación.	El archivo se salva en la aplicación reemplazando el que existía anteriormente.

Caso de prueba de caja blanca "Obtener resultados"

```
public List<Resultados> getResultados() throws FileNotFoundException, IOException{
   List<Resultados> results = new LinkedList<Resultados>();                (1)
   File raiz = new File("analisis");                                       (1)
   if(raiz.exists() && raiz.isDirectory() && raiz.listFiles().length > 0){  (2)
     for(int i = 0; i < raiz.listFiles().length; i++){                     (3)
       ObjectInputStream in = new ObjectInputStream(new FileInputStream(raiz.listFiles()[i]));  (4)
       try{                                                                (5)
         results.add((Resultados)in.readObject());                         (6)
       }catch(Exception e){}                                               (7)
     }                                                                     (8)
   }
   return results;                                                         (9)
 }
```

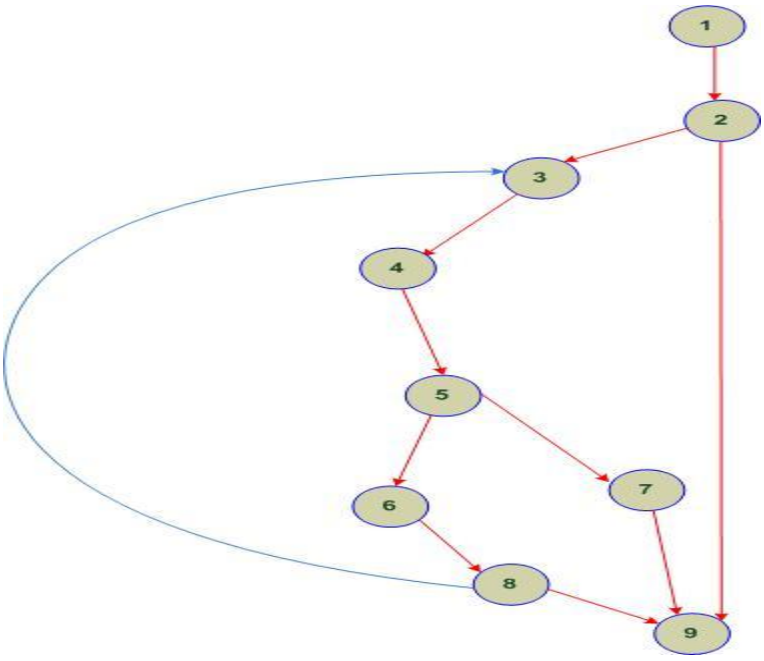

Ilustración 19: Grafo de flujo del caso de prueba "Obtener resultados".

Complejidad ciclomática => V(G)

Nodos => N

Aristas => A

V(G) = A – N + 2

V(G) = 11 – 9 + 2

V(G) = 4

Resultado de la Prueba:

Existen 4 caminos linealmente independientes:

Camino 1: 1 2 9

Camino 2: 1 2 3 4 5 6 8 9

Camino 3: 1 2 3 4 5 7 9

Camino 4: 1 2 3 4 5 6 8 3 4 5 7 9

Tabla 6: Caso de prueba de caja blanca "Obtener resultados".

No. de caminos	Caso de prueba	Objetivo	Resultado esperado
1	!raiz.exists() || !raiz.isDirectory() && raiz.listFiles().length== 0	Verificar que la carpeta no exista, o que no sea un directorio y que cuente con al menos 1 archivo.	Devuelve una lista vacía.
2	raiz.exists() || raiz.isDirectory() && raiz.listFiles().length > 0	Verificar que si cumple con los criterios se puedan guardar los resultados.	Devuelve una lista con 1 resultado.
3	catch(Exception e){}	Verificar que si ocurre algún error sea capturado y lanzado.	Lanza la excepción capturada.
4	raiz.exists() || raiz.isDirectory() && raiz.listFiles().length > 0	Verificar que si cumple con los criterios se puedan guardar los resultados.	Devuelve una lista con todos los resultados que cumplieron con los criterios.

Caso de Prueba de Caja Blanca "Obtener Tree Model"

public TreeModel getTreeModel(){

```
  TreeModel result;                                    (1)
  TreeNode root = new DefaultMutableTreeNode(nombre);                (1)
  MutableTreeNode  [] mods = new MutableTreeNode[modulos.size()];          (1)
  for(int i = 0; i < modulos.size(); i++){                          (2)
    mods[i] = new DefaultMutableTreeNode(modulos.get(i));              (3)
    MutableTreeNode  apps = modulos.get(i).getApps();                  (3)
    MutableTreeNode  web = modulos.get(i).getWeb();                    (3)
    ((DefaultMutableTreeNode)mods[i]).add(apps);                       (3)
    ((DefaultMutableTreeNode)mods[i]).add(web);                        (3)
    ((DefaultMutableTreeNode)root).add(mods[i]);                       (3)
  }                                                    (4)
  result = new DefaultTreeModel(root);                              (5)
  return result;
}
```

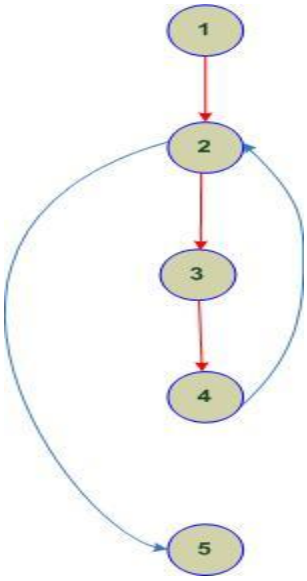

Ilustración 20: Grafo de flujo del caso de prueba "Obtener Tree Model".

Complejidad ciclomática => V(G)

Nodos => N

Aristas => A

V(G) = A – N + 2

V(G) = 5 – 5 + 2

V(G) = 2

Resultado de la Prueba:

Existen 2 caminos linealmente independientes:

Camino 1: 1 2 5

Camino 2: 1 2 3 4 2 5

Tabla 7: Caso de prueba de caja blanca "Obtener Tree Model".

No. de caminos	Caso de prueba	Objetivo	Resultado esperado
1	i = 0 i == modulos.size()	Verificar que si no existen módulos para adicionarlos al resultado no se produzca la adición.	Devuelve un TreeModel vacío.
2	i < modulos.size()	Verificar que si existen módulos para adicionarlos al resultado se produzca la adición.	Devuelve un TreeModel con los módulos adicionados.

Caso de Prueba de Caja Blanca "Abrir"

```
public void abrir(String nombre, String ruta) throws Exception {
   LinkedList<Modulo>  modulos = new LinkedList<Modulo>();                    (1)
   File root = new File(ruta);                                                (1)
   File apps = new File(ruta + System.getProperty("file.separator") + "apps");   (1)
   File web = new File(ruta + System.getProperty("file.separator") + "web");     (1)
   File[] rutasWeb = web.listFiles();                                         (1)
   File[] rutasApps = apps.listFiles();                                       (1)
   for (int iApps = 0; iApps < rutasApps.length; iApps++) {                   (2)
    for (int iWeb = 0; iWeb < rutasWeb.length; iWeb++) {                      (3)
      if (rutasApps[iApps].isDirectory()                                      (4)
         && rutasWeb[iWeb].isDirectory()
         && rutasApps[iApps].getName().equals(rutasWeb[iWeb].getName())
         && !rutasApps[iApps].getName().equals(".svn")) {
       MutableTreeNode  appsTreeRoot = new DefaultMutableTreeNode("apps");     (5)
       MutableTreeNode  webTreeRoot = new DefaultMutableTreeNode("web");       (5)
       MutableTreeNode  appsTreeNode = new ArchivosTreeNode(rutasApps[iApps]);  (5)
       MutableTreeNode  webTreeNode = new ArchivosTreeNode(rutasWeb[iWeb]);     (5)
       webTreeRoot.insert(webTreeNode, 0);                                    (5)
       appsTreeRoot.insert(appsTreeNode, 0);                                  (5)
```

```
        Modulo modulo = new Modulo(appsTreeRoot, webTreeRoot, rutasApps[iApps].getName()); (5)
        modulos.add(modulo);                                                                (5)
      }
    }                                                                                       (6)
  }                                                                                         (7)
  proyecto = new Proyecto(nombre, ruta, modulos);                                           (8)
}
```

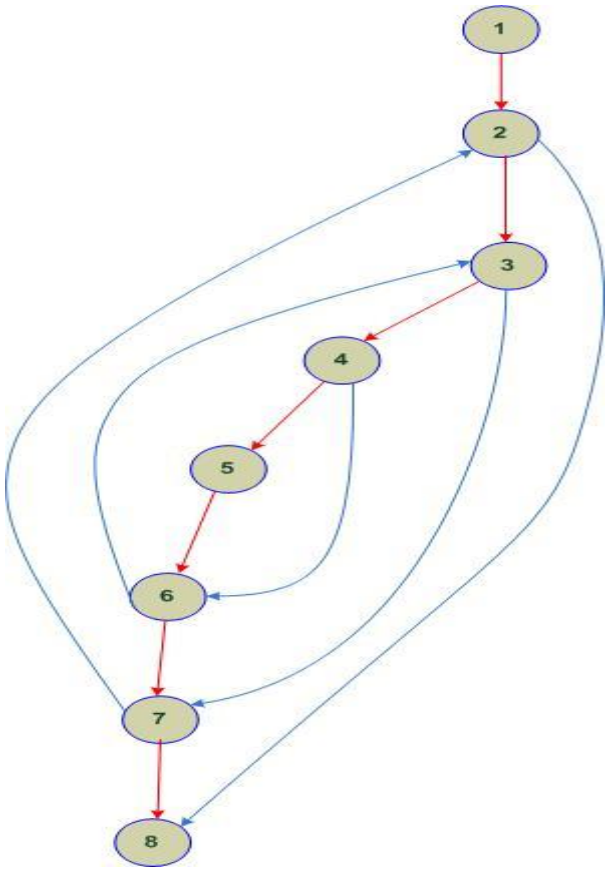

Ilustración 21: Grafo de flujo del caso de prueba "Abrir".

Complejidad ciclomática => V(G)

Nodos => N

Aristas => A

V(G) = A – N + 2

V(G) = 12 – 8 + 2

V(G) = 6

Resultado de la Prueba:

Existen 6 caminos linealmente independientes:

Camino 1: 1 2 3 4 5 6 7 8

Camino 2: 1 2 8

Camino 3: 1 2 3 7 8

Camino 4: 1 2 3 4 6 7 8

Camino 5: 1 2 3 4 5 6 3 4 5 6 7 8

Camino 6: 1 2 3 4 5 6 7 2 3 4 5 6 7 8

Tabla 8: Caso de prueba de caja blanca "Abrir".

No. de caminos	Caso de prueba	Objetivo	Resultado esperado
1	iApps <rutasApps.length iWeb < rutasWeb.length rutasApps[iApps].isDirectory() rutasWeb[iWeb].isDirectory()	Verificar que al menos cuente con 1 ruta web y una apps, y que las dos sean un directorio.	Crea un módulo y lo adiciona a la lista de módulos.
2	iApps == rutasApps.length	Verificar que si no existe ninguna ruta apps no se cree el módulo, ni se adicione.	Se devuelve una lista vacía.
3	iWeb < rutasWeb.length	Verificar que si no existe ninguna ruta web no se cree el módulo, ni se adicione.	Se devuelve una lista vacía.
4	!rutasApps[iApps].isDirectory() !rutasWeb[iWeb].isDirectory() !rutasApps[iApps].getName(). equals(rutasWeb[iWeb].getName()) rutasApps[iApps].getName().equals(".svn"))	Verificar que si el iweb y el iapps no son directorios o no tiene el mismo nombre, o el nombre que tiene es ".svn" no se permita crear el módulo y	Devuelve una lista vacía

		adicionarlo.	
5	iApps <rutasApps.length iWeb < rutasWeb.length	Verifica que mientras se cumpla esta condición, se pueda crear una serie de módulos y adicionarlos	Crea varios módulos y los adiciona a la lista de módulos.
6	iApps <rutasApps.length iWeb < rutasWeb.length	Verifica que mientras se cumpla esta condición se pueda crear una serie de módulos y adicionarlos	Crea varios módulos y los adiciona a la lista de módulos.

Al aplicarle la técnica de caja blanca del camino básico a 4 de las funcionalidades de la aplicación se pudo comprobar que se obtuvieron los resultados deseados debido a que se pudieron recorrer todos los caminos básicos evitando la pérdida de algún camino por ejecutar.

3.6 Conclusiones parciales

Una vez concluido el proceso de realización del presente capítulo se obtuvieron las siguientes conclusiones:

- ✓ Se obtuvo el modelo de implementación del sistema, a través del diagrama de componentes y de despliegue lo que permitió dar paso a la fase de implementación.
- ✓ Durante la fase de implementación, se logró un correcto tratamiento de errores garantizando una mejor calidad del sistema.
- ✓ Se realizaron las pruebas de caja blanca y caja negra para validar la aplicación, las cuales mostraron que la misma contaba con un adecuado funcionamiento.

Conclusiones Generales

Una vez concluido el desarrollo de la “Herramienta para el análisis de código fuente sobre el marco de trabajo Sauxe”, se puede plantear que se cumplieron todos los objetivos trazados al inicio de la investigación, razón por la cual se arriba a las siguientes conclusiones generales:

- ✓ El estudio del arte de las herramientas informáticas que pudieran resultar útiles para el análisis de código fuente arrojó que ninguna de estas cumple con las necesidades que se requieren, por lo que se hizo necesario la realización de esta herramienta.

- ✓ Se decidió la utilización de herramientas como el Visual Paradigm 8.0 y el NetBeans 7.0 y lenguajes como Java y UML, elementos fundamentales para la realización de la aplicación.

- ✓ Se realizó el análisis y el diseño de la aplicación, permitiendo recoger todos los requisitos necesarios, además de lograr un mejor entendimiento del proceso informatizado, garantizando una comprensión exacta sobre las funcionalidades a informatizar.

- ✓ Se realizó la implementación de la solución propuesta lo que permitió dar respuestas a los requisitos identificados.

- ✓ La herramienta fue expuesta a diferentes pruebas, que arrojaron resultados satisfactorios lo que permitió validar el correcto funcionamiento del software.

Recomendaciones

Con el desarrollo de la herramienta y el resultado de la investigación se proponen las siguientes recomendaciones para mejorar la calidad de la misma:

- ✓ Incluir nuevos tipos de análisis a la herramienta con vista a aumentar la cantidad de servicios a brindar.
- ✓ Que la herramienta sea puesta en explotación en los proyectos productivos de la universidad que utilicen Sauxe como marco de trabajo.

Bibliografía

1. **Expósito, Raúl. 2011.** ¿Qué es el Análisis Estático del Código ? *minipaint-mconde.* [En línea] 2011. http://minipaint-mconde.googlecode.com/files/AnalisisEstaticoCodigo.pdf.

2. **SDTeam. 2009.** Security Database. *RATS v2.3 - Rough Auditing Tool for Security.* [En línea] 10 de Noviembre de 2009. http://www.security-database.com/toolswatch/RATS-v2-3-Rough-Auditing-Tool-for.html.

3. **Miguel Angel Sicilia. 2009.** Métricas de Mantenibilidad Orientadas al Producto. *Connexions.* [En línea] 2009. http://cnx.org/content/m17466/latest/.

4. **Atom. 2010.** trainedchimpanzees. [En línea] 2010. http://trainedchimpanzees.blogspot.com/2010/03/mantenibilidad-vs-reusabilidad.html.

5. **Camps, Jose. 2011.** ethek. [En línea] 22 de Noviembre de 2011. http://www.ethek.com/definicion-del-codigo-fuente/.

Referencias Bibliográficas

1. **Expósito, Raúl. 2011.** ¿Qué es el Análisis Estático del Código ? *minipaint-mconde.* [Online] 2011. http://minipaint-mconde.googlecode.com/files/AnalisisEstaticoCodigo.pdf.

2. **BUGUROO OFFENSIVE SECURITY S.L. 2011.** buguroo . *bugScout.* [En línea] 21 de Febrero de 2011. https://buguroo.com/productos/bugscout/.

3. **Mitchell, Scott. 2008.** MSDN Magazine. *Herramientas de análisis estático para. NET, Matt Berseth 's blog.* [En línea] Diciembre de 2008. http://msdn.microsoft.com/es-es/magazine/dd263071.aspx.

4. **Rudder, Don. 2011.** CASE building + technology. *Software Development.* [En línea] 2011. http://www.case-inc.com/content/software-development.

5. **Visual Paradigm International. 2010.** Visual Paradigm. *Visual Paradigm for UML.* [En línea] 2010. http://www.visual-paradigm.com/product/vpuml/.

6. **Terreros, Julio Casal. 2011.** msdn. *Desarrollo de Software basado en Componentes .* [En línea] 2011. http://msdn.microsoft.com/es-es/library/bb972268.aspx.

7. **Cornejo, José Enrique González. 2010.** docIRS. *¿Qué es UML?* [En línea] 2010. http://www.docirs.cl/uml.htm.

8. **Cervantes, Humberto. 2011.** SG. *Arquitectura de Software.* [En línea] Diciembre de 2011. http://sg.com.mx/content/view/922.

9. **Pelaez, Juan Carlos. 2009.** Blog de Juan Peláez en Geeks.ms. *Arquitectura basada en capas.* [En línea] 29 de Mayo de 2009. http://geeks.ms/blogs/jkpelaez/archive/2009/05/29/arquitectura-basada-en-capas.aspx.

10. **Mercado, Ing. William. 2007.** El Rincón del Vago . *Lenguajes de programación.* [En línea] 31 de marzo de 2007. http://html.rincondelvago.com/lenguajes-de-programacion_22.html.

11. **Filipiak, Joanna María. 2009.** *Herramienta de análisis estático de código para encontrar vulnerabilidades de seguridad en las aplicaciones Web.* Madrid : s.n., 2009.

12. **Oracle Corporation. 2011.** NetBeans IDE. *NetBeans IDE Features.* [En línea] 2011. http://netbeans.org/features/index.html.

13. **Oracle Corporation. 2011.** JAVA. *Que es Java, Características de Java como Lenguaje de programación.* [En línea] 2011. http://www.infor.uva.es/~jmrr/TAD2003/Sesiones/TADONJava/JAVA.html.

14. **Thames, Juan Pablo Bustos. 2011.** slideshare. *Diagramas de clases del Diseño.* [En línea] 9 de Agosto de 2011. www.slideshare.net/jpbthames/diagramas-de-clases.

15. **López, Dailyn Sosa. 2011.** eumed.net. *SISTEMA PARA EL CONTROL DEL USO DE LOS SOFTWARES EDUCATIVOS.* [En línea] 2011. http://www.eumed.net/libros-gratis/2009c/585/Descripcion%20del%20modelo%20del%20dominio.htm.

16. **Martínez, Juan Francisco Javier. 2007.** Entorno Virtual de Aprendizaje. *Guía de construcción de software en Java con patrones de diseño.* [En línea] 2007. http://eva.uci.cu/file.php/158/Documentos/Recursos_bibliograficos/Libros_y_articulos_UD_1/Diseno_de_software/Patrones.

17. **Rojas, M.C. Juan Carlos Olivares. 2006.** Entorno Virtual de Aprendizaje . *Patrones de Diseño.* [En línea] 2006. http://eva.uci.cu/file.php/158/Documentos/Recursos_bibliograficos/Libros_y_articulos_UD_1/Diseno_de_software/Patrones_de_Diseno_Art._2.pdf.

18. **PRUEBASDESOFTWARE . 2005.** pruebasdesoftware. *Gestión de Calidad y Pruebas de Software.* [En línea] 2005. http://pruebasdesoftware.com/laspruebasdesoftware.htm.

19. **UCI. 2010.** Entorno Virtual de Aprendizaje. *Tipos de prueba de Caja Blanca y Negra.* [En línea] 2010. http://eva.uci.cu/file.php/158/Documentos/Recursos_bibliograficos/Libros_y_articulos_UD_2/Comun/Material_de_caja_b_y_caja_n.pdf.

20. **Alva, Eduardo Rivera. 2008.** Scribd. *Diagramas de Componentes.* [En línea] 11 de Noviembre de 2008. http://es.scribd.com/doc/7884665/Arquitectura-de-Software-II-Diagrama-de-Componentes-y-Despliegue.

21. **Sparx Systems. 2007.** Sparx Systems. *Diagrama de Despliegue .* [En línea] 2007. http://www.sparxsystems.com.ar/resources/tutorial/uml2_deploymentdiagram.html.

22. **Toro, Amador Durán. 2000.** Metodología para la Elicitación. *Entorno Virtual de Aprendizaje.* [En línea] Octubre de 2000. http://eva.uci.cu/file.php/161/Documentos/Materiales_basicos/Materiales_basicos_de_la_Unid

ad_2/Otros_materiales/2_Obtencion_de_Requisitos/Metodologia_para_la_Elicitacion_de_Requisitos_de_Sistemas_Software.pdf.

23. **MasterMagazine. 2010.** MASTERMAGAZINE. *Definición de Código Fuente.* [En línea] 2010. http://www.mastermagazine.info/termino/4328.php.

24. **Adrián Hernández Yeja, Yosbany Tejas de la Cruz. 2010.** *Sistema de Análisis Estático de Vulnerabilidades. Módulo PHP.* La Habana : s.n., 2010.

25. **Teudis Naranjo Ortiz, William Sánchez Espinosa. 2010.** *Sistema de Análisis Estático de Vulnerabilidades en Python.* La Habana : s.n., 2010.

26. **Yaritza Montero Vaillant, Ernesto Miguel Rodríguez Rodríguez. 2011.** *Plataforma para el Análisis Estático de Vulnerabilidades del Código.* La Habana : s.n., 2011.

27. **EelcoVisser. 2006.** Program-Transformation.Org. *PHP-SAT.org.* [En línea] 2006. http://www.program-transformation.org/PHP/.

28. **Pylint. 2006.** pylint.org. *Pylint User Manual.* [En línea] 2006. http://www.pylint.org/.

Anexos

1- **Descripción del RF5**

Precondiciones	Se ha registrado al menos un análisis de código fuente en el sistema	
Flujo de eventos		
Flujo básico		
1	El usuario selecciona el análisis a eliminar de la lista de análisis guardados en el sistema.	
2	Presiona el botón eliminar.	
3	El sistema solicita confirmación para eliminar el análisis.	
4	El usuario confirma eliminar el análisis seleccionado.	
5	El sistema confirma la eliminación.	
6	Concluye el requisito.	
Pos-condiciones		
1	Se eliminó el análisis previamente seleccionado.	
Flujos alternativos		
Flujo alternativo *. El usuario cancela la acción		
1	Concluye el requisito.	
Pos-condiciones		
1	No se elimina el análisis.	
Validaciones		
1	Se validan los datos según lo establecido en el Modelo conceptual.	
Relaciones	**Requisitos Incluidos**	N/A
	Extensiones	N/A
Conceptos	N/A	N/A
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

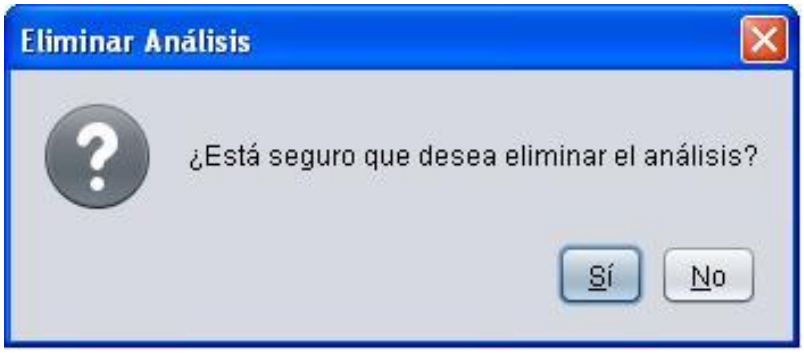

Ilustración 22: Prototipo de interfaz de usuario "Eliminar análisis de código fuente".

2- **Descripción del RF6**

Precondiciones	Se ha registrado al menos un análisis de código fuente en el sistema
Flujo de eventos	
Flujo básico	
1	Se selecciona el análisis a modificar de la lista de análisis guardados en el sistema.
2	Se presiona el botón modificar.
3	El sistema muestra una ventana que permite editar el nombre del análisis seleccionado.
4	El usuario introduce los datos nuevos.
5	Se presiona el botón aceptar.
6	El sistema valida los datos introducidos.
7	Si los datos son correctos el sistema los registra.
8	El sistema confirma el registro de los datos.
9	Concluye el requisito.
Pos-condiciones	
1	Se modificó el análisis previamente seleccionado.
Flujos alternativos	
Flujo alternativo *.a El usuario cancela la acción	
1	Concluye el requisito.
Pos-condiciones	
1	No se registran los datos.
Flujo alternativo 7.a Datos incompletos	
1	El sistema señala los datos vacíos y permite corregirlos.
2	El usuario corrige los datos.

3	Volver al paso 6 del flujo básico.
Flujo alternativo 7.a Datos repetidos	
1	El sistema informa la existencia de datos repetidos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 6 del flujo básico.
Pos-condiciones	
	N/A
Validaciones	
	Se validan los datos según lo establecido en el Modelo conceptual.

Relaciones	**Requisitos Incluidos**	N/A
	Extensiones	N/A
Conceptos	N/A	N/A
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

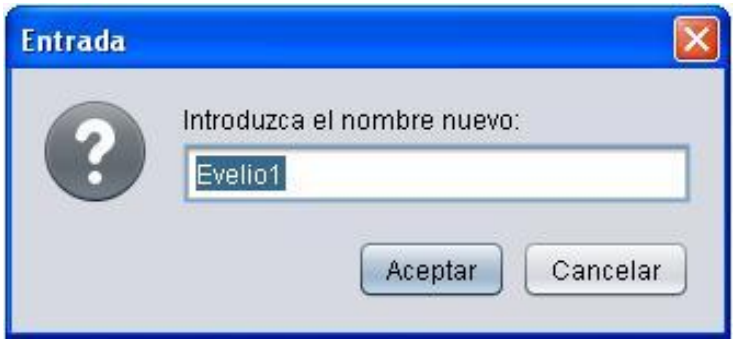

Ilustración 23: Prototipo de interfaz de usuario "Modificar análisis de código fuente".

3-

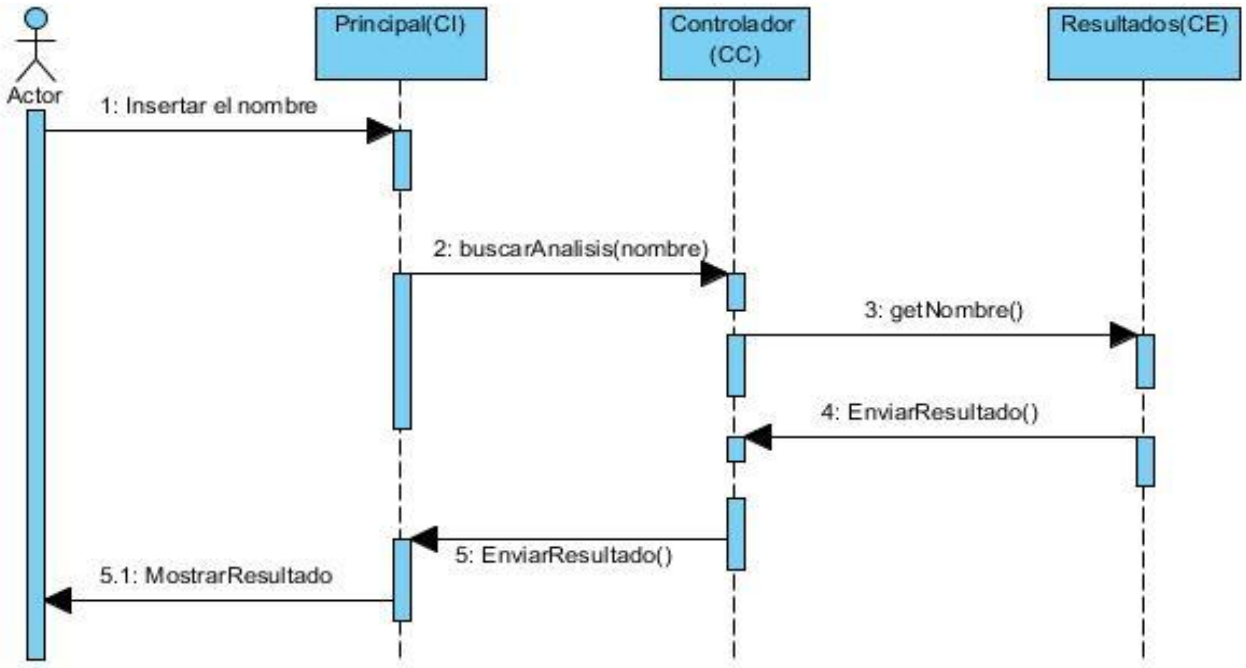

Ilustración 24: Diagrama de secuencia "Buscar análisis de código fuente".

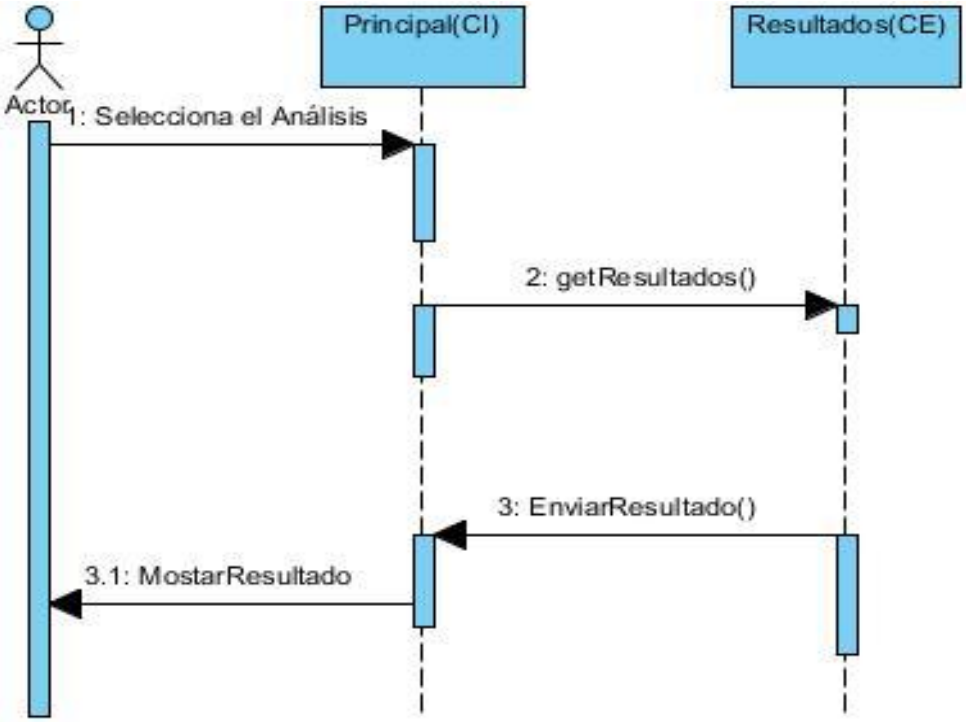

Ilustración 25: Diagrama de secuencia "Vista previa".

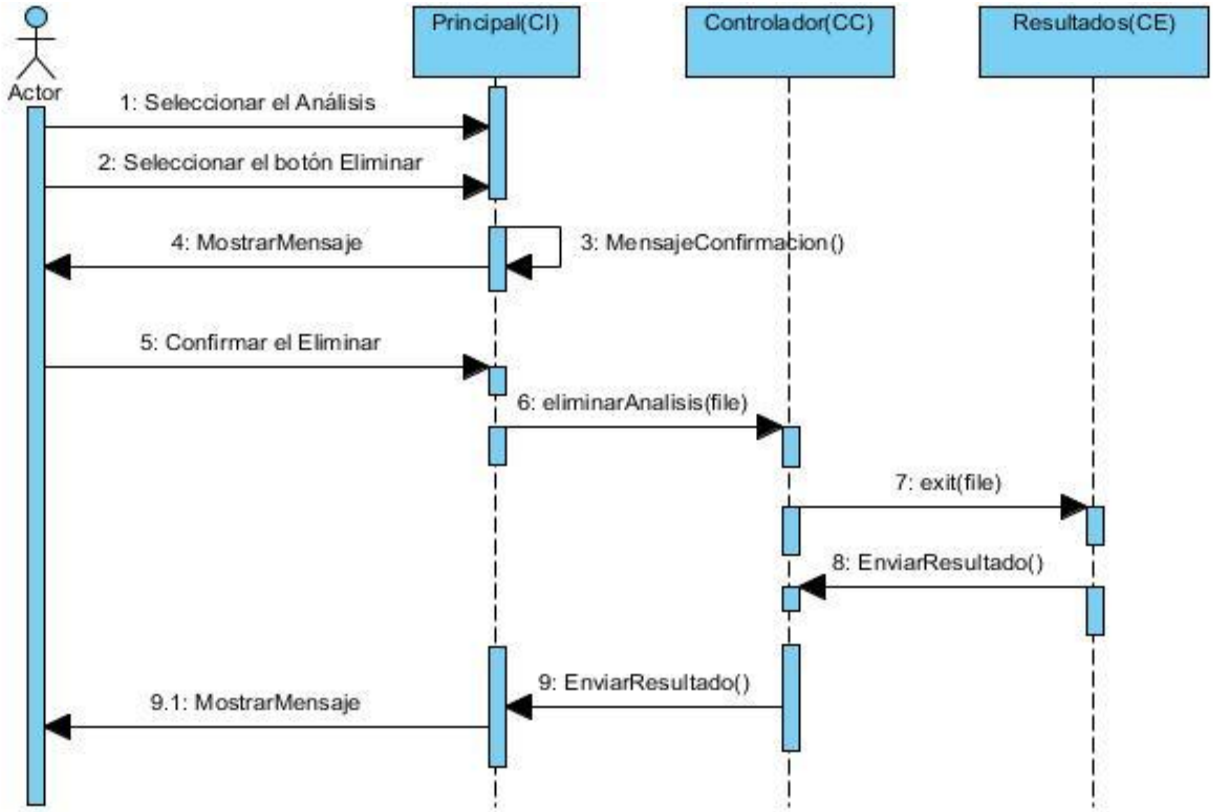

Ilustración 26: Diagrama de secuencia "Eliminar análisis de código fuente".

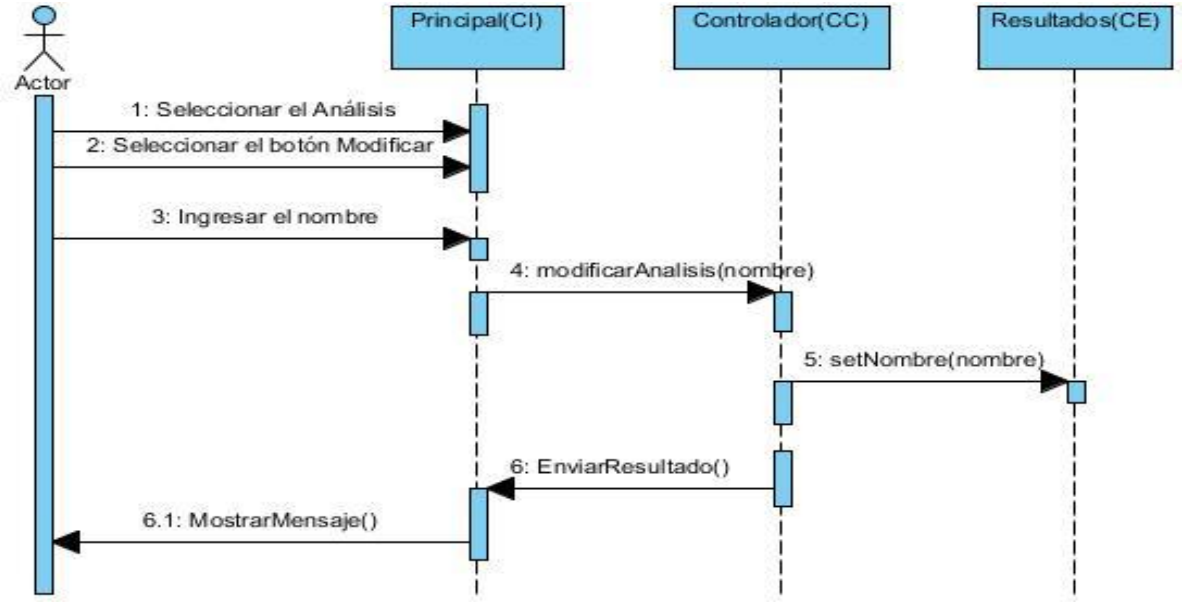

Ilustración 27: Diagrama de secuencia "Modificar análisis de código fuente".

4- **Caso de prueba de caja negra "Eliminar análisis de código fuente"**

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1:Buscar análisis de código fuente.	Señala el análisis que cumple con el criterio de búsqueda insertado.	EP 1.1: Buscar análisis insertando datos válidos en el criterios de búsqueda.	Se introduce los datos de búsqueda para la acción. Se presiona el botón Buscar.
		EP 1.2: Buscar análisis sin insertar datos en los criterios de búsqueda.	No se introduce ningún dato de búsqueda para la acción. Se presiona el botón Buscar.
		EP 1.2: Buscar análisis insertando datos que no concuerden con el criterio de búsqueda	Se introduce los datos de búsqueda para la acción. Se presiona el botón Buscar.

Caso de prueba de caja negra "Listar análisis de código fuente"

Nombre del requisito	**Descripción general**	**Escenarios de pruebas**	**Flujo del escenario**
1: Listar análisis de código fuente.	Se listan los análisis que estén guardados en el sistema, se muestran los datos: Nombre. Tipo. Fecha. Módulos.	EP 1.1: Listar análisis.	Se muestran los análisis.

Caso de prueba de caja negra "Vista previa"

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1: Vista Previa.	Se obtiene el resultado del análisis que se señale.	EP 1.1: Vista Previa.	Se selecciona el análisis del que se desea obtener el resultado.