UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 6

Diseño de herramienta software para la automatización de pruebas de Caja Blanca al departamento de Señales Digitales del centro GEySED.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN INFORMÁTICA.

AUTOR: David Rodríguez Sánchez

TUTOR: Ing. Yusdenys Pérez Mendoza

Ciudad Habana, Junio, 20 del 2011.

"Año 53 de la Revolución"

Dedicatoria:

A mi papá, a mi mamá, a Kenia, a Eglis y a mi familia en general.

Agradecimientos:

A mi papá, mi novia, mi familia, mi tutor, el tribunal, y a mis amigos.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 6 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____ _____

David Rodríguez Sánchez Ing. Yusdenys Pérez Mendoza

Autor Tutor

ÍNDICE

ÍNDICE DE TABLAS Y FIGURAS

RESUMEN

Con el objetivo de alcanzar mayor calidad en los productos elaborados en el centro GEySED de la Facultad 6 es que el Grupo de Calidad de dicha entidad asume la necesidad de mejorar el proceso de pruebas al software finalizado. Entre las diferentes categorías de pruebas se encuentran las técnicas de pruebas de Caja Blanca, que se caracterizan por incidir directamente sobre el código fuente de la aplicación a revisar, analizando su complejidad, eficiencia y calidad. Estas pruebas en particular tienden a ser muy trabajosas y complicadas, por lo cual surge la necesidad de automatizar este proceso con el fin de poder aplicarlas al proceso de pruebas de software, logrando un mejor aseguramiento de la calidad y eficiencia en dicho proceso. En el presente trabajo de diploma se realiza el análisis y diseño de una herramienta cuyo propósito es precisamente alcanzar este objetivo. Con este propósito se realizará entonces el proceso de modelación del Dominio, Análisis y Diseño del sistema propuesto, incluyendo en cada uno de estos los diagramas necesarios. Se añade también un breve resumen de las diferentes tecnologías a emplear durante estos procesos, y la implementación. Se llevará a cabo también un estudio del estado del arte de las aplicaciones de esta categoría y un análisis crítico del flujo actual de los procesos relacionados al tema así como de la factibilidad de la implementación de la herramienta.

INTRODUCCIÓN

Con el desarrollo de la producción de software a nivel mundial también ha surgido la necesidad, principalmente a partir de la década de 1970, de crear métodos que aseguren la calidad del producto que se realiza. Existen modelos que establecen normas y estándares para certificar la calidad del software que se produce a nivel mundial, ejemplo de esto es el Capability Maturity Model Integration (CMMI) o Modelo de Integración de Capacidad y Madurez. Este es un modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software.

El gobierno cubano en un esfuerzo por hacer de Cuba una sociedad informatizada crea la Universidad de Ciencias Informáticas (UCI), compuesta de 7 facultades centrales y 3 regionales ubicadas en Artemisa, Ciego de Ávila y Granma. En todas las facultades pertenecientes a la universidad se imparte la carrera de Ciencias Informáticas a la vez que se vincula el estudio con la producción de software. En este centro de estudios se crean aplicaciones con diferentes objetivos, entre ellos para la exportación. En este sentido se han obtenido resultados satisfactorios pero aún insuficientes.

Es entonces que con el objetivo de asegurar la calidad del software que se produce en la Universidad se crea el Departamento Central de Calidad que cuenta con grupos que asesoran a cada facultad. En el Centro de Geoinformática y Señales Digitales (GEySED) de la UCI, que se dedica principalmente al desarrollo de aplicaciones vinculadas a la geología y la captura y procesamiento de señales digitales de radio y televisión, existe un grupo de calidad que se encarga de velar por el cumplimiento del proceso de aseguramiento de la calidad del software que se produce para cerciorarse de que dicha aplicación se encuentra lista para ser liberada.

Entre los principales aspectos que validan un software, está el estado óptimo del código fuente con que se construyó. Para ello es preciso que todos los desarrolladores realicen la programación respetando estándares que se definen, documentan y comprueban. Para verificar la eficiencia de la aplicación en su etapa de desarrollo, se aplican las pruebas de Caja Blanca que hacen incidencia directa sobre el código fuente. Estas comprueban el funcionamiento y detectan errores en los bucles, condiciones, decisiones y todo lo relacionado a estructuras y variables. Es definitoria la

técnica del camino básico creada por Thomas J. McCabe en el año 1976 que se basa plenamente en la teoría de grafos. Este procedimiento tiene como resultado un parámetro llamado complejidad ciclomática que tiene varios significados:

- Determina la complejidad lógica del software.

- Determina el número máximo de casos de prueba que se deben diseñar para probar el código.

Del código se pueden extraer otros parámetros como la densidad de comentarios y cantidad de líneas. Los comprendidos en las métricas de Halstead y Chidamber & Kemerer son muy utilizados también. Son parámetros que si se logran mantener con valores adecuados, se obtiene un código útil y eficiente.

Para llevar a cabo estas actividades, es ventajoso utilizar herramientas informáticas que automaticen estos procesos. Dichas aplicaciones son capaces de realizar cálculos grandes en muy poco tiempo y con gran exactitud. Con ello se lograría analizar gran cantidad de código en instantes y con independencia del lenguaje y plataforma a que pertenezca. Esto brinda confianza y certeza en el grado de calidad.

Aplicaciones que realicen esta tarea existen muchas, e incluso en la UCI se ha trabajado en este sentido, sin embargo en el Departamento de Señales Digitales del centro GEySED precisan contar con una aplicación que sea capaz de agilizar el proceso de realización de las pruebas de Caja Blanca que cumpla con ciertas especificaciones de acuerdo al código que se genera en los proyectos del centro e incluso que dicha aplicación logre verificar el cumplimiento de los estándares de codificación establecidos.

Partiendo de la problemática anterior se plantea el siguiente **problema a resolver:** La necesidad de automatizar la aplicación de pruebas de Caja Blanca a los productos del Departamento de Señales Digitales del centro GEySED de la UCI.

Para orientar la investigación se cuenta entonces con el siguiente **objeto de estudio:** El proceso de pruebas aplicando técnicas de Caja Blanca. Y para delimitar el mismo se define como **campo de acción:** La automatización de los procesos de prueba

aplicando técnicas de Caja Blanca para los productos del Departamento de Señales Digitales del centro GEYSED de la UCI.

El **objetivo general** planteado es: Realizar el análisis y diseño de una herramienta software que permita la automatización del proceso de pruebas de Caja Blanca aplicadas a los productos del Departamento de Señales Digitales del centro GEySED de la UCI con las herramientas especificadas.

Para lo cual se plantea la siguiente **idea a defender:** El desarrollo de una herramienta para la automatización de técnicas de pruebas de Caja Blanca permitirá mejoras en el proceso de pruebas a los productos de software del centro GEYSED, lo que conllevará al aumento de la calidad de los mismos.

Con este trabajo de diploma se pretenden alcanzar los **posibles resultados**:

- Análisis y diseño de un sistema informático que permita la automatización del proceso de pruebas de Caja Blanca a los productos del Departamento de Señales Digitales.

- Documentación requerida de la investigación así como los artefactos vinculados con la metodología a utilizar.

Y es con este fin que se plantean las siguientes **tareas de investigación:**

1. Sintetizar los conocimientos relacionados con la calidad de software.
2. Caracterizar las diferentes técnicas utilizadas para las pruebas de Caja Blanca.
3. Definir el objeto de automatización.
4. Describir el estado actual sobre los lenguajes y plataformas de programación y herramientas CASE existentes que permitan el modelado e implementación de una herramienta para la realización del proceso de pruebas de Caja Blanca a los productos desarrollados en el Departamento de Señales Digitales.
5. Modelar los procesos de Negocio, Análisis y Diseño del sistema.

Los **métodos científicos** a utilizar en la investigación:

1. **Métodos Teóricos:**

    - Analítico-Sintético: Para la confección de la investigación se realizará el estudio de conceptos que serán analizados por separado, estudiando

minuciosamente cada uno de ellos y comparando el criterio de disímiles autores y las semejanzas entre ellos.

- Histórico-Lógico: Se realizará un estudio exhaustivo del surgimiento de los procedimientos para el aseguramiento de la calidad y en específico para la realización de pruebas de Caja Blanca con el interés de llegar a su esencia. Así como una búsqueda de las deficiencias que podría presentar dicho proceso.

2. **Métodos Empíricos:**

    - Observación: Se empleará para el aprendizaje del funcionamiento de las técnicas y métodos de pruebas de software que son objetivo de la presente investigación, profundizando mediante la experiencia en la práctica en el comportamiento y detalles de la realización de las pruebas de Caja Blanca.

    - Experimento: Los experimentos serán muy útiles en el transcurso de la investigación, en el aprendizaje de las técnicas de prueba de Caja Blanca.

En el **CAPÍTULO 1**: Fundamentación teórica. del presente trabajo de investigación se realizará el análisis de los conceptos vinculados al tema de la investigación para lograr un mejor entendimiento de lo planteado y se añade un estudio de las diferentes técnicas de pruebas de Caja Blanca y una caracterización de las mismas, así como de los diferentes niveles de prueba. Además se resumirán de forma breve las principales características de las tecnologías, herramientas y lenguajes a emplear en el desarrollo de la herramienta.

Luego en el **CAPÍTULO 2**: Características del sistema. se abordará la descripción del sistema. Como parte de dicha descripción se presentan los procesos de negocio relacionados al objeto de estudio, aunque se hace necesario definir un conjunto de conceptos agrupados en un Modelo de Dominio. Se enumeran además los requisitos funcionales y no funcionales, incluyendo una descripción de los actores y casos de uso.

Finalmente, en el **CAPÍTULO 3**: Análisis y Diseño del sistema. se lleva a cabo el flujo de Análisis y Diseño según las especificaciones de la metodología RUP, los cuales

conforman la vista lógica de la arquitectura de software en la construcción de aplicaciones dirigida por modelos. Se realizará el análisis del sistema, posteriormente se procederá a la modelación de la lógica del negocio mediante el uso de clases y se tratan los principios del diseño de la aplicación.

CAPÍTULO 1: Fundamentación teórica.

En este capítulo se analizan los conceptos vinculados al tema de la investigación para lograr un mejor entendimiento de lo planteado. Entre los temas a tratar se encuentran variadas definiciones de calidad, calidad de software y las diferentes formas de medir esta última. Se añade un estudio de las diferentes técnicas de pruebas de Caja Blanca y una caracterización de las mismas, así como de los diferentes niveles de prueba. Además se resumirán de forma breve las principales características de las tecnologías a emplear en el desarrollo de la herramienta.

1.1 Calidad de software.

1.1.1 Calidad de software.

La calidad son todas las propiedades y características de un producto, proceso o servicio que hacen de este la solución en mayor o menor grado a un conjunto de necesidades establecidas o implícitas. Esta definición ha evolucionado a través del tiempo, según G. Taguchi la calidad es: "Las pérdidas que un producto o servicio infringe a la sociedad desde su producción hasta su consumo o uso. A menores pérdidas sociales, mayor calidad del producto o servicio". Esta manera de ver este concepto contiene la ventaja de incluir tanto los problemas clásicos de calidad como son: pérdidas sociales debidas a la variabilidad, como los más recientes, como pérdidas sociales causadas por efectos secundarios nocivos y problemas del medio ambiente entre otros (1).

Si se toma como ejemplo uno de los conceptos emitidos por el Dr. Walter Shewart que la definía como: “Un problema de variación, el cual puede ser controlado y prevenido mediante la eliminación a tiempo de las causas que lo provocan” (2), la calidad es entonces una característica o propiedad ligada a las cosas que hace posible su comparación con otras de su mismo género. En realidad, definir la calidad de una forma precisa se hace muy difícil ya que se trata de una opinión subjetiva.

Luego Pressman define la calidad de software de forma muy completa como: “Concordancia del software producido con los requerimientos explícitamente establecidos, con los estándares de desarrollo prefijados y con los requerimientos implícitos no establecidos formalmente, que desea el usuario” (3).

Existen muchas definiciones de lo que es la calidad, pero en este trabajo se tiene en cuenta un pequeño resumen el cual se tomará como referencia. Definiendo la calidad entonces como: todo el proceso que se aplica a un producto con el objetivo de satisfacer las necesidades y expectativas del cliente. Con el fin de lograr un software que cumpla los requisitos deseados es que se somete el mismo, durante su elaboración, a un proceso de aseguramiento de la calidad.

1.1.2 Proceso de aseguramiento de la calidad.

El aseguramiento de la calidad, es comúnmente definido como el esfuerzo total para plantear, organizar, dirigir y controlar la calidad en un sistema de producción, con el objetivo de brindar al cliente mejores productos (4). Es un sistema y, como tal, es un conjunto organizado de procedimientos bien definidos y entrelazados armónicamente (5).

El aseguramiento de la calidad es el conjunto de acciones planificadas y sistemáticas, que son necesarias para proporcionar la confianza adecuada de que un producto o servicio satisface los requisitos de calidad, los cuales deben estar sustentados en la satisfacción de las expectativas de los clientes (6).

Teniendo en cuenta las definiciones anteriores se puede tomar como el más completo para esta investigación el tercer concepto ya que plantea de manera explícita los objetivos de este proceso.

Para llevar a cabo el aseguramiento de la calidad de software se debe escoger un modelo de calidad a seguir, lo cual no es más que un cúmulo de buenas prácticas para el proceso de desarrollo de software y una herramienta que guía a los grupos de desarrollo a la mejora continua y a la competitividad por obtener un producto de gran aceptación.

1.1.3 Proceso de prueba de software.

Una vez aplicado el modelo de calidad y finalizada la fase de implementación del software es entonces momento de someter el mismo a la fase de pruebas donde se medirá la calidad del producto. Es en este período donde se detectan posibles errores y/o fallas que podrían afectar el resultado final. Según Deutsch: “El desarrollo de sistemas de software implica una serie de actividades de producción en las que las posibilidades de que aparezca el fallo humano son enormes. Los errores pueden

empezar a darse desde el primer momento del proceso, en el que los objetivos… pueden estar especificados de forma errónea o imperfecta, así como en posteriores pasos de diseño y desarrollo… Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo de software ha de ir acompañado de una actividad que garantice la calidad" (7).

Los diversos tipos de prueba se resumen en el **¡Error! No se encuentra el origen de la referencia.**.

Las pruebas al software son aplicadas con variados objetivos y para esto se realizan en diferentes escenarios o niveles de trabajo, dichos niveles de prueba son:

- **Prueba de desarrollador:** Esta prueba es diseñada y realizada por el equipo de desarrollo. Tradicionalmente son realizadas solo para la prueba de unidad, aunque en la actualidad es posible ejecutar pruebas de integración. Lo cual se recomienda, ya que las pruebas de desarrollador deben abarcar más que solo la prueba de unidad.

- **Prueba independiente:** En este nivel la prueba es diseñada e implementada por alguien independiente del grupo de desarrolladores. Su objetivo es proporcionar una perspectiva diferente y un ambiente más rico que los desarrolladores. Una vista independiente de este tipo de prueba es la prueba independiente de los stakeholders, las cuales son basadas en las necesidades y preocupaciones de los mismos.

- **Prueba de unidad:** Es más enfocada a los elementos con posibilidades de ser probados más pequeños del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos y funcionen como es deseado. Siempre está orientada a pruebas Caja Blanca, que se analizarán más adelante.

- **Prueba de integración:** Es ejecutada para asegurar que los componentes en el modelo de implementación realicen su tarea correctamente cuando son mezclados para ejecutar un caso de uso. En su realización se revisa un paquete o un conjunto de paquetes del modelo de implementación. Estas pruebas sacan a la luz errores o incompletitud en las especificaciones de las interfaces de los paquetes.

Esta prueba es responsabilidad del equipo de desarrolladores y personal independiente a este.

- **Prueba de regresión:** Cada vez que se añade un nuevo módulo durante una prueba de integración, el software cambia. Estos cambios pueden causar problemas con funciones que antes trabajaban a la perfección. La prueba de regresión ayuda a asegurar que dichas modificaciones no introducen un comportamiento no deseado o errores adicionales a la aplicación.

- **Prueba basada en hilos:** Esta prueba integra el conjunto de clases necesarias para responder a una entrada de datos o un evento del sistema. Cada hilo se integra y chequea individualmente y luego se aplica la prueba de regresión para evitar resultados no deseados.

- **Prueba basada en uso:** En este caso se inicia construyendo las clases independientes, o sea aquellas que hacen poco uso de las clases servidor, luego se comprueban las clases dependientes, que usan las clases independientes, y esta secuencia continúa hasta completar la construcción del sistema.

- **Prueba de sistema:** Estas pruebas se realizan cuando el software está funcionando como un todo. Está dirigida a chequear el programa final, una vez que ya han sido integrados todos los componentes de software y hardware.

- **Prueba de aceptación:** Es la prueba final para dar paso al despliegue del sistema. Tiene como objetivo comprobar que el software está listo para ser usado por sus usuarios finales y ejecutar las funciones y tareas para las cuales fue construido.

Entre las muchas formas definidas para probar la calidad de un software se encuentran dentro de las más utilizadas las llamadas pruebas de Caja Negra, estas se enfocan en la interfaz del software, o sea, los casos de pruebas intentan demostrar que las funciones de la aplicación sean correctas, que la entrada y salida de datos sea la deseada, así como la integridad de la información externa, examinando, por tanto, algunos aspectos del modelo fundamental del sistema sin tener muy en cuenta la estructura lógica interna de este. Las pruebas de Caja Blanca, por su parte, se centran

en la optimización del código y la detección de errores en éste. A lo largo de la investigación se profundizará más en este tipo de pruebas.

1.2 Pruebas de Caja Blanca.

1.2.1 Pruebas de Caja Blanca.

Mediante el uso de pruebas de Caja Blanca al software se verifican los caminos lógicos del software, proponiendo casos de prueba en los cuales se compruebe el correcto funcionamiento de conjuntos específicos de condiciones y/o bucles. Es posible analizar el estado del programa en variados puntos para determinar si el estado real de la aplicación es semejante al esperado. Las pruebas de Caja Blanca a aplicar a un software son derivadas de las especificaciones internas del diseño del producto.

Haciendo uso de los métodos de prueba de Caja Blanca, que se especifican más adelante, el ingeniero de software puede realizar casos de prueba que garanticen que:

- Se ejerciten al menos una vez todos los caminos independientes para cada módulo.
- Se ejerciten todas las decisiones lógicas tanto en sus vertientes verdaderas como en las falsas.
- Se ejecuten todos los bucles en sus límites y con sus límites operacionales.
- Se ejerciten las estructuras internas de datos para asegurar su validez.

Por lo tanto para llevar a cabo este tipo de procedimientos es necesario el acceso al código fuente del programa.

1.2.2 Métodos de prueba de Caja Blanca.

Las pruebas de Caja Blanca se llevan a cabo de diferentes formas y métodos, algunos de ellos son:

- **Prueba del camino básico:** Propuesta inicialmente por Tom McCabe este método de prueba permite obtener una medida de la complejidad lógica de un diseño procedimental y luego utilizar esa medida como guía para la definición

de un conjunto básico de caminos de ejecución. Los casos de prueba derivados del conjunto básico deben garantizar que durante el transcurso de la prueba cada sentencia del programa sea ejecutada al menos una vez. Los pasos que se siguen para aplicar esta técnica son:

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

- **Prueba de condición:** Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de la aplicación, donde una condición simple se toma como una variable lógica o una expresión relacional. Pruebas de este género se han propuesto muchas pero probablemente la más sencilla es la prueba de las ramificaciones.

- **Prueba de flujo de datos:** Esta prueba selecciona caminos de prueba de un software de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.

- **Prueba de bucles:** Se centra exclusivamente en la validez de la implementación de los bucles, dados que estos son la piedra angular de la mayoría de las aplicaciones de software.

- **Prueba por mutaciones:** El objetivo de las pruebas utilizando mutación consiste en construir casos de prueba que descubran el error existente en cada mutante. Un mutante se define como una copia del programa original pero con un error de codificación que ha sido introducido a propósito, dicho error podría ser cambiar un operador de suma por uno de multiplicación.

1.3 Estándares de codificación.

1.3.1 Estándares de codificación.

Un estándar de codificación es un conjunto de reglas a seguir durante el proceso de escribir el código fuente de una aplicación y abarca todos los aspectos de la generación de código como son el formato de nomenclatura de los diferentes elementos que puede contener y la forma en que deben estar escritos y distribuidos los comentarios en el código. El estándar de programación a utilizar debe ser seleccionado muy cuidadosamente, pero este debe siempre debe ser pensado de forma que facilite la programación eficaz. Un estándar de codificación bien pensado y aplicado logra hacer parecer que el código entero fue escrito por un solo programador.

Un código fuente debe ser legible, este aspecto tiene influencia directa en la capacidad de un programador para entender el sistema de software. La mantenibilidad es una característica que determina la medida en la que un software puede ser modificado, corregido o mejorado. Si bien la legibilidad y la mantenibilidad son afectadas por diferentes factores, el programador influye mucho al aplicar determinada técnica de codificación y la mejor manera de asegurar que todo el equipo de desarrolladores se apegue a los requerimientos de calidad es establecer un estándar de codificación al que se le someterá a revisiones posteriores.

Cumpliendo condiciones anteriormente planteadas y haciendo uso de técnicas de codificación sólidas y buenas prácticas de programación es el modo de asegurar que un proyecto de software se convierta luego en un sistema fácil de comprender y mantener.

Las revisiones de código deben reforzar los estándares de codificación de manera uniforme, aunque su principal objetivo sea localizar defectos en el mismo. El estándar de codificación a utilizar en un proyecto de software debe especificarse al inicio de este, no tiene ningún sentido, ni es prudente imponer un estándar luego de iniciada la codificación.

1.3.2 Ventajas del uso de estándares de codificación.

El uso de estándares de codificación conlleva disímiles ventajas, algunas de ellas son:

- Asegurar la legibilidad del código, facilitando el trabajo en un equipo de programadores.
- Brindar una guía para el encargado de mantenimiento y/o actualización del sistema al contener este un código claro y bien documentado.
- Facilitar la portabilidad entre plataformas y aplicaciones.

Es por estos motivos que el código fuente del sistema debe cumplir con los requisitos establecidos en el proyecto de software antes de iniciar el proyecto.

En el centro GEySED, y no solo en este, sino en todos los centros y proyectos de la facultad 6, existe un estándar que debe ser usado en la codificación de las aplicaciones producidas por dichas entidades. A continuación se detallan los diferentes aspectos del mismo.

1.3.3 Aspectos del estándar de codificación.

Principios generales.

- Los nombres de cada uno de los elementos del programa deben ser significativos.
- No manejar en los programas más de una instrucción por línea.
- Declarar las variables en líneas separadas.
- Añadir comentarios descriptivos junto a cada declaración de variables, si es necesario.
- La mayoría de los elementos se deben nombrar usando sustantivos.
- Los atributos deben comenzar con letra minúsculas.
- Los métodos deben comenzar con letra mayúsculas.

Nombramiento de elementos.

- **Clases, interfaces y archivos fuente:**
    - Nombre sustantivo singular, con la primera letra en mayúscula y las demás en minúsculas.
    - Si el nombre de la clase está comprimido en más de una palabra la primera letra debe de ser mayúscula. Letras mayúsculas seguidas no son permitidas.
- **Clases:** Las clases deben comenzar con un prefijo alegórico al componente que correspondan, evitando nombres duplicados.
- **API:** Las API deben comenzar con un prefijo alegórico al componente que correspondan, evitando nombres duplicados y terminar en el sufijo _Component.
- **Constantes:** Nombre sustantivo en mayúsculas. Para separar palabras se usará el guión bajo (_).
- **Identificadores de variables:**

- Comenzarán siempre con la primera letra minúscula correspondiente a su tipo de dato.
- Para distinguir palabras dentro del nombre deberá emplearse un guión bajo (_).

Comentario de funciones.

- Objetivo de la función y no descripción del procedimiento.
- Explicación del uso de argumentos (parámetros).
- Explicación del uso de valores devueltos (de retorno).

1.4 Métricas.

La medición de la calidad de un producto es fundamental para obtener un software que cumpla los requerimientos establecidos. Las métricas aplicadas a la construcción de aplicaciones de software abarcan un gran número de medidas dentro del ámbito de la planificación del proyecto, las métricas de calidad pueden ser aplicadas tanto en organizaciones como en procesos o simplemente a productos y estas afectan directamente la estimación de los costos.

Una métrica es usada para caracterizar cierta propiedad de un objeto o una clase de objeto, en la ingeniería de software, esta caracterización es cuantitativa. Las métricas de software proveen una vía de estimación del costo de la realización del producto, evaluar la calidad y predecir el tiempo a invertir en el mantenimiento, mejora o actualización del sistema.

De ser necesario calcular el costo y esfuerzo invertido en un proyecto es necesario entonces conocer el tamaño de la aplicación, en el caso de realizar una estimación de la complejidad, existen diversas métricas que brindan métodos e información para realizar esta tarea, como lo son las métricas de McCabe y Haltstead.

Las métricas se miden en:

- **Directas:** Se obtienen a través de medidas realizadas directamente sin involucrar otros atributos.
- **Indirectas:** Se derivan de métricas directas.

La clasificación de las mismas tiene en cuenta los siguientes aspectos:

- **Del proceso:** Son los atributos de actividades relacionadas con el software.
- **Del producto:** Son los componentes, entregas o documentos resultantes de una actividad de proceso.
- **De los recursos:** Son entidades requeridas por una actividad de proceso.

En la actualidad existen numerosas métricas que sirven de apoyo en la construcción de software, como son las ya mencionadas de McCabe y Halstead y otras como la de Chidamber & Kemerer y Li – Henry; también CMMI, que es el sistema por el cual se rige la UCI, cuenta con diferentes métricas para medir la calidad de un software y actualmente en el centro GEySED de la Facultad 6 de la UCI se encuentra en desarrollo un conjunto de métricas a emplear en la construcción de software que se apegue a las necesidades del centro.

1.5 Antecedentes de sistemas automatizados basados en pruebas de Caja Blanca.

En la actualidad ha aumentado la tendencia a automatizar los diferentes procesos de prueba, en este aspecto también influye el aumento de las facilidades disponibles para esta tarea, aunque no por ello este tipo de aplicaciones han dejado de involucrar una alta complejidad tanto en su desarrollo, como en su aplicación práctica y mantenimiento. Debido a esto es que estas herramientas son escasamente utilizadas en entornos industriales.

Aun así han surgido a nivel mundial aplicaciones de esta gama con el objetivo de facilitar, perfeccionar y agilizar el proceso de pruebas de software.

En el caso de las aplicaciones basadas en pruebas de Caja Blanca se pueden mencionar algunas ya establecidas como son JTest, el cual fue el primer programa automático de búsqueda de bugs y fue ideado para Java, Docklight Scripting, una herramienta de testeo automatizado para protocolos seriales de comunicación vía COM, TCP y UDP4, entre otras como: AQtime, McCabe TQ y Resource Standard Metrics (RSM); este último provee análisis de los lenguajes ANSI C, ANSI C++, C# y Java, y puede ser usado en los sistemas operativos Windows y Unix.

En Cuba aun no existen aplicaciones de producción nacional que implementen funcionalidades variadas de prueba de Caja Blanca aunque si se están dando pasos en este sentido e incluso en nuestra universidad se cuenta ya con varias aplicaciones encaminadas en este sentido.

A pesar de la existencia de diferentes herramientas que brindan la capacidad de automatizar los variados procesos de prueba de Caja Blanca, en el centro GEySED de la Facultad 6 de la UCI, surge la necesidad de desarrollar una aplicación que se apegue a las exigencias de los productos generados en este centro.

1.6 Metodología de desarrollo de software.

Una metodología de desarrollo de software es la guía de apoyo que conduce a la elaboración de un software de calidad, es un conjunto de buenas prácticas que se deben aplicar durante el proceso de desarrollo.

La adecuada selección de la metodología a utilizar afectará directamente el proceso de confección de la aplicación. La metodología seleccionada define: ¿Quién debe hacer qué?, ¿Cuándo y cómo debe hacerlo? Las características de cada proyecto determinan la metodología a emplear en el mismo.

La metodología que regirá en este caso es: El Proceso Unificado de Desarrollo de Software o Rational Unified Process (RUP), en inglés, que es la usada en los proyectos de software del centro GEySED y la gran mayoría de los proyectos de la UCI. El empleo de esta metodología trae numerosas ventajas, dadas por su capacidad de abarcar ampliamente todos los aspectos del desarrollo de software, la experiencia acumulada en el uso de la misma en la universidad, la documentación que genera, que brinda amplio soporte al software y la compatibilidad con otros proyectos del centro.

1.6.1 El Proceso Unificado de Desarrollo de Software.

RUP es actualmente la metodología estándar más utilizada para el análisis, implementación y generación de documentación de sistemas basados en el paradigma orientado a objetos y soportados en el Lenguaje Unificado de Modelado (UML) para su metodología.

“RUP es un producto de Rational (IBM). Se caracteriza por ser:

- **Iterativo e incremental:** La alta complejidad de los sistemas actuales hace que sea factible dividir el proceso de desarrollo en varios mini-proyectos o versiones del producto donde a cada uno de estos se les denomina iteración y pueden o no representar un incremento en el grado de terminación del producto completo.

- **Centrado en la arquitectura:** La arquitectura representa la forma del sistema, la cual va madurando en su interacción con los casos de uso hasta llegar a un equilibrio entre funcionalidad y características técnicas.
- **Guiado por los casos de uso:** RUP utiliza los casos de uso tanto para especificar los requisitos funcionales del sistema, como para guiar todos los demás pasos de su desarrollo, dígase Diseño, Implementación y Prueba" (8).

RUP es muy general, ya que no está pensado para adaptarse solo a proyectos de software y este puede describirse en dos dimensiones a lo largo de dos ejes:

- **Eje horizontal:** Representa tiempo y muestra el aspecto dinámico del proceso, expresándolos en términos de ciclos, fases, iteraciones y metas.
- **Eje vertical:** Representa la parte estática del proceso: su descripción en términos de actividades, artefactos, trabajadores y flujos de trabajo.

Esta metodología define las fases:

- Inicio
- Elaboración
- Construcción
- Transición

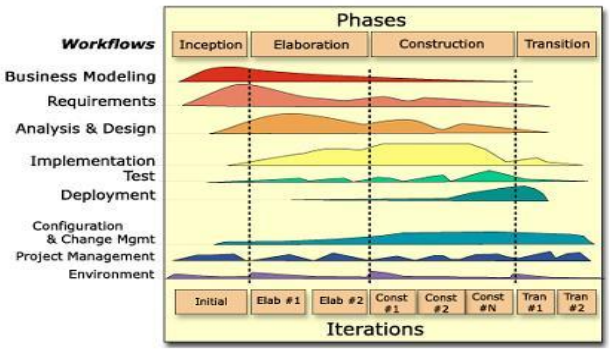

Ilustración 1: Fases y flujos de trabajo que componen a RUP.

También cuenta con nueve flujos de trabajo, seis de ellos de Ingeniería (Modelado del Negocio, Requerimientos, Análisis y Diseño, Implementación, Prueba y Despliegue) y tres de apoyo (Gestión de la Configuración, Gestión de Proyecto y Ambiente); y su objetivo es implementar las mejores prácticas actuales de la ingeniería de software.

1.7 Tecnologías, herramientas y lenguajes empleados en el desarrollo del sistema.

1.7.1 Herramienta CASE.

Las herramientas de Ingeniería de Software Asistida por Computadoras (CASE, por sus siglas en inglés), son definidas como el conjunto de aplicaciones y documentos de

ayuda que brindan asistencia al equipo de un proyecto de software a través de las diferentes etapas del mismo. Estos programas brindan ventajas como:

- Incremento en la velocidad de desarrollo del sistema o sistemas.
- Minimizan el tiempo de codificación y prueba, ofreciendo más tiempo al proceso de análisis y diseño.

Y durante el desarrollo del software permiten:

- Automatizar el dibujo de diagramas.
- Facilitar la generación de la documentación del sistema.
- Facilitar la creación de las relaciones en la base de datos.
- Generar estructuras de código.
- Aumentar la productividad.

Se ha definido que en el proceso de desarrollo del sistema propuesto se empleará como herramienta CASE el Visual Paradigm 3.4. Esta herramienta es considerada como una de las más completas en su campo, con gran facilidad de uso, soporte multiplataforma y extraordinaria capacidad de interoperabilidad con otras aplicaciones. Aunque la herramienta líder en este campo es el Rational Rose de IBM, pero contiene la desventaja de ser una aplicación con licencia de tipo privativa, a diferencia del Visual Paradigm, que no es libre, pero una vez que se ha pagado la licencia, la cual ya la UCI posee, se otorga al dueño de esta la posibilidad de visualizar y modificar el código fuente a su gusto.

El principal objetivo del Visual Paradigm es automatizar y acelerar el ciclo vital del desarrollo de software, proporcionando herramientas para la captura de requisitos, análisis, diseño e implementación de la aplicación, y ofreciendo funcionalidades de generación de código y de informes e ingeniería inversa.

En otro esquema de facilidades el Visual Paradigm permite crear esquemas de clases a partir de una base de datos y realizar también el proceso contrario. Posibilita invertir el código fuente de aplicaciones, archivos ejecutables y binarios en modelos UML e incluye toda la documentación necesaria.

Diseñado para el desarrollo de sistemas de software de gran escala basados en el uso del paradigma orientado a objetos, también soporta los más recientes estándares de

notación de Java y UML y permite el trabajo en equipo de varios desarrolladores en el mismo diagrama y con actualización de cambios en tiempo real.

1.7.2 Lenguaje Unificado de Modelado.

El Lenguaje Unificado de Modelado (UML), es un lenguaje gráfico para visualizar, especificar, construir y documentar sistemas de software. UML brinda un estándar para construir un esquema visual del sistema, llamado modelo, que incluye aspectos conceptuales como procesos de negocio y funcionalidades del software y aspectos concretos como sentencias del lenguaje de programación, esquemas de base de datos y componentes de software reutilizables.

"UML es un 'lenguaje' para especificar y no un método o un proceso. UML se puede usar en una gran variedad de formas para soportar una metodología de desarrollo de software (tal como el Proceso Unificado de Rational), pero no especifica en si mismo que metodología o proceso usar*"* (9)*.*

Los principales rasgos de UML son:

- Posibilita mejores tiempos totales de desarrollo.
- Permite modelar sistemas.
- Tecnología orientada a objetos.
- Posibilita la definición de conceptos y artefactos ejecutables.
- Encamina el desarrollo del escalamiento en sistemas complejos de misión crítica.
- Brinda un lenguaje de modelado empleado por humanos y máquinas.
- Mejor soporte a la planeación y al control de proyectos.
- Alta reutilización.
- Minimización de costos.

Ventajas:

- Se puede conocer por adelantado lo que se debe obtener de un sistema de software que ha sido diseñado y documentado antes de ser codificado.
- Evita posibles comportamientos inesperados del software al brindar la posibilidad de detectar a tiempo 'agujeros' lógicos en el diseño.

- Minimiza el tiempo de desarrollo de software permitiendo tomar todas las decisiones del diseño antes de comenzar a escribir el código ya que la aplicación se desarrollará siguiendo el diseño previo.
- Realizar modificaciones en el sistema es mucho más sencillo sobre la documentación UML y hay que recurrir a este recurso menos frecuentemente.
- Los diagramas UML permiten un sencillo y rápido entendimiento del sistema.
- “La comunicación con los desarrolladores del proyecto y con desarrolladores externos, es mucho más eficiente” (10).

1.7.3 Lenguaje de programación.

El lenguaje de programación permite al programador de la aplicación comunicarse con el ordenador e indicarle a este de forma precisa que operaciones debe llevar a cabo para realizar las funciones deseadas. Estos lenguajes conforman un punto medio entre el lenguaje natural de los humanos y los ceros y unos de las computadoras; una de sus principales ventajas es que permite el entendimiento y trabajo en equipo de diferentes programadores.

El lenguaje de programación a utilizar será C++, el cual abarca tres paradigmas de programación: estructurada, genérica y orientada a objetos y es considerado entre los lenguajes más potentes y eficientes gracias a la posibilidad de usarlo para trabajar tanto a bajo como a alto nivel, aunque su aprendizaje es muy incómodo debido a que obliga a hacerlo casi todo manualmente.

1.7.4 Marco de trabajo.

Un marco de trabajo o framework, en el desarrollo de software, es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado, entre otras aplicaciones para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio y brinda una estructura y una metodología de trabajo, la cual, extiende o utiliza las aplicaciones del dominio.

El proceso de codificación de la aplicación en el lenguaje será auxiliado del marco de trabajo Qt. Qt es un framework para el desarrollo de aplicaciones multiplataforma desarrollado por la compañía Nokia.

Algunas de sus características son:

- Compatibilidad multiplataforma con un solo código fuente.
- Performance de C++.
- Disponibilidad del código fuente.
- Excelente documentación.
- Arquitectura lista para plugins.
- Múltiples librerías de C++ implementadas.

1.7.5 Entorno de desarrollo.

Un entorno de desarrollo integrado (IDE, por sus siglas en inglés) es una aplicación compuesta por un conjunto de herramientas de programación, entre ellas se pueden encontrar un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Puede dedicarse exclusivamente a un solo lenguaje de programación o abarcar varios de ellos. Los IDEs pueden constituir una aplicación por sí solos o pueden encontrarse integrados con otros sistemas de software.

El objetivo de un entorno de desarrollo es brindar un marco de trabajo amigable para la mayoría de los lenguajes de programación y en algunos de ellos puede constituir un sistema de tiempo de ejecución, en donde se permite trabajar con el lenguaje de forma interactiva.

El entorno de desarrollo que se empleará para facilitar el proceso de codificación de la aplicación en el lenguaje C++ con el uso del framework Qt es el Qt Creator, que no es más que un entorno de desarrollo integrado multiplataforma implementado específicamente para facilitar el proceso de codificación de aplicaciones haciendo uso del marco Qt.

El Qt Creator provee herramientas como:

- Editor de código C++ y JavaScript.
- Diseñador integrado de interfaz de usuario (UI).
- Herramientas de administración de proyecto y construcción de aplicaciones (build).

- Herramientas de asistencia a la búsqueda de errores en el código.
- Soporte para control de versiones.
- Simulador de UI para teléfonos móviles.

Conclusiones

A través del tiempo se ha ido perfeccionando el proceso de construcción de software y con él, el proceso de aseguramiento de la calidad derivando en métodos avanzados que realmente garantizan en gran medida la realización de un producto de mucha aceptación. Las pruebas al software son parte imprescindible de este proceso ya que permiten la detección de errores humanos y/o inconformidades con el resultado esperado. Es por ello que la automatización de este proceso, en particular del de Caja Blanca, en las que se enfoca este trabajo, es una tarea necesaria, así como la realización de una aplicación con las herramientas definidas y que realice esta tarea y se apegue a las necesidades del centro GEySED, para garantizar productos de alta calidad, evitar errores que pueden ser detectados a tiempo y ahorrar tiempo en el desarrollo de las aplicaciones.

CAPÍTULO 2: Características del sistema.

En este capítulo se abordará la descripción del sistema objetivo de este trabajo de diploma, como parte de dicha descripción se presentan los procesos de negocio relacionados al objeto de estudio, aunque se hace necesario definir un conjunto de conceptos agrupados en un Modelo de Dominio, debido a la escasa estructuración de los procesos del negocio, con el fin de realizar una apropiada captura de requisitos y crear las condiciones para la posterior implementación del sistema.

Se enumeran además los requisitos funcionales y no funcionales, incluyendo una descripción de los actores y casos de uso resultantes del flujo de trabajo de Requisitos.

2.1 Objeto de automatización.

Con el objetivo de asegurar la calidad del software, comúnmente se realizan revisiones al código fuente mediante el uso de pruebas de Caja Blanca. El carácter engorroso de este proceso, el tiempo necesario para realizarlo y la insuficiencia de personal especializado para llevarlo a cabo justifican la necesidad de automatizar este tipo de pruebas en el centro GEySED de la Facultad 6 haciendo uso de las siguientes técnicas ya mencionadas:

- Prueba del camino básico.
- Prueba de condición.
- Prueba de flujo de datos.
- Prueba de bucles.
- Generación automatizada de casos de prueba.

Sin embargo, debido a la alta complejidad técnica que requiere la implementación de una aplicación que realice dichas operaciones, se decide realizar un sistema informático que facilite la realización de las pruebas e incluya entre sus funcionalidades las siguientes tareas:

- Generar el grafo de flujo a partir del código.
- Calcular la complejidad ciclomática.
- Determinar el conjunto básico de caminos independientes.
- Revisar los estándares de codificación.

Con esta herramienta se persigue la meta de facilitar la realización de los procesos de pruebas, perfeccionarlos e incrementarlos y de esta forma alcanzar mayor seguridad y confianza en la calidad del software producido en el centro.

2.1.1 Flujo actual de los procesos involucrados en el campo de acción.

En el Grupo de Calidad del centro GEySED de la Facultad 6, que trabaja en conjunto al centro de Calidad UCI, se llevan a cabo diversas tareas vinculadas a la entrega de productos de alta calidad como resultado de un proceso de desarrollo de software. Entre las mencionadas tareas se destaca el proceso de pruebas a las aplicaciones terminadas, siempre teniendo en cuenta la planificación o las estrategias de pruebas a seguir en el momento que son realizadas con el fin de garantizar así una mejor práctica de dicho proceso y lográndose que el sistema obtenga la calidad esperada por el cliente.

En la actualidad los integrantes de dicho proyecto no cuentan con la experiencia necesaria para llevar a cabo con profundidad y efectividad el proceso de control de la calidad, a pesar de esto, los esfuerzos en este sentido han producido resultados y ya los productos listos para ello son sujetos a una serie de pruebas de Caja Negra a partir de un grupo de casos de pruebas aceptados y revisados previamente por el personal del Grupo de Calidad del centro GEySED en conjunto con el proyecto objeto de revisión que garantizan la correcta ejecución de las funcionalidades solicitadas por el cliente.

Sin embargo, aunque se realizan pruebas, se hace necesario realizar pruebas al código fuente de las aplicaciones tanto para verificar el nivel de optimización y rendimiento de este y que cumpla con los estándares y parámetros de codificación establecidos. Con el fin de alcanzar este objetivo es que se determina la meta del presente trabajo de diploma como el primer paso en este sentido.

2.1.2 Análisis crítico de la ejecución de los procesos.

En la actualidad el procedimiento de pruebas de Caja Blanca, utilizando cualquiera de las técnicas citadas, no se aplica en el Grupo de Calidad del centro GEySED, pero no por esto deja de ser una etapa importante y necesaria en el proceso de aseguramiento de la calidad y una forma más de medir la funcionalidad del software implementado y lograr minimizar las posibilidades de ocurrencia de errores tras su liberación.

En esto influye el factor de que el personal del proyecto no cuenta con la preparación requerida para llevar a cabo este tipo de pruebas, además de que de realizarlas, sería de forma manual, y, de este modo, su realización puede llegar a resultar bastante engorrosa y agotadora y no sería eficiente su ejecución pues solo provocaría un derroche de tiempo y dejaría exhausto al probador al realizar, por ejemplo, los cálculos de la complejidad ciclomática o la representación del grafo de flujo en la técnica del camino básico, que es fundamental y casi obligatoria en términos de pruebas de Caja Blanca, de forma tan poco factible y obsoleta. El resultado final no sería del todo satisfactorio, debido a la posible pérdida de información, y solo resultaría en el atraso prácticamente injustificado de la fecha de liberación del producto de software.

Todos estos aspectos pueden repercutir de forma notoria en la efectividad y duración del período de pruebas, disminuyendo así la validez del mismo, ya que no existen herramientas automatizadas que provean los mecanismos necesarios en el centro GEySED para darle solución a estas problemáticas, siendo esta la única posibilidad viable para lograr aplicar en todos los escenarios permisibles las técnicas de prueba de Caja Blanca.

2.2 Modelo de Dominio.

El Modelo de Dominio muestra las clases conceptuales significativas en el dominio del problema en cuestión, abarcando los ejemplares más importantes de objetos que existen o los eventos que suceden en el entorno donde se desempeñará la herramienta. Es considerado un subconjunto del Modelo de Objetos del Negocio.

En el transcurso del desarrollo de la herramienta no se logró relacionar procesos bien definidos en el entorno del negocio. Se hizo difícil especificar los elementos más importantes del sistema y sus interconexiones, así como no se logró establecer las reglas su funcionamiento. A pesar de ello es posible identificar personas, eventos, transacciones y objetos involucrados en dicho entorno escasamente perfilado, por lo cual se determinó necesario realizar un modelado del dominio concerniente a la solución.

2.2.1 Descripción de los conceptos principales.

- **Usuario:** Se considera como cualquier persona que trabaje con la aplicación, sin tener en cuenta su categoría.

- **Parámetro de código:** Son las características o propiedades de una métrica, u orden o valor que se le pasa a una función con el fin de modificar su comportamiento.
- **Estándar de codificación:** Norma técnica que se puede utilizar como punto de comparación para evaluar la calidad del servicio o alertar sobre las deficiencias del mismo. En este caso dichas normas son aplicadas a la tarea de la generación del código de un software.
- **Fragmento de código:** Se refiere a la porción de texto escrito, generalmente, por una persona y que luego será traducido a otro lenguaje para que sea interpretado y ejecutado por una computadora. Referente a la programación de software. Podrá ser cargado desde un archivo en el que se haya guardado.
- **Nodo:** Representación de una o varias sentencias en secuencia o procedimentales. Cada nodo comprende como máximo una sentencia de decisión (Bifurcación). Un solo nodo es capaz de abarcar una secuencia de procesos o una sentencia de decisión. Puede ocurrir también que existan nodos que no se asocien, se usan principalmente al inicio y final del grafo.
- **Arista:** Línea que conecta dos nodos y muestra el flujo de control, son análogas a las representadas en un diagrama de flujo. Una arista debe terminar en un nodo, aunque dicho nodo no represente sentencia procedimental alguna.
- **Grafo de flujo:** Representación del flujo de control lógico de la aplicación mediante una notación. Para ello se tienen en cuenta los nodos, aristas y regiones.
- **Conjunto básico de caminos independientes:** Grupo de caminos del software en el cual cada camino es único e incluye nuevas instrucciones de un proceso o una nueva condición.
- **Complejidad ciclomática:** Medida de la complejidad lógica de un módulo determinado y el esfuerzo mínimo necesario para calificarlo. Es también el número de rutas lineales independientes de dicho módulo y por lo tanto el número mínimo de rutas que deben probarse.
- **Revisión de estándares:** Es el conjunto de acciones realizadas por el sistema para comprobar el correcto cumplimiento de los estándares de codificación establecidos.
- **Reporte:** Es la representación de un conjunto de datos en un formato establecidos, en este contexto dichos datos exponen los resultados de las pruebas realizadas al código.

- **Proyecto:** Se refiere al trabajo realizado durante el análisis de un código. Incluye la configuración especificada para dicho código fuente, el código en sí y los resultados obtenidos de las pruebas realizadas.

2.2.2 Diagrama del Modelo de Dominio.

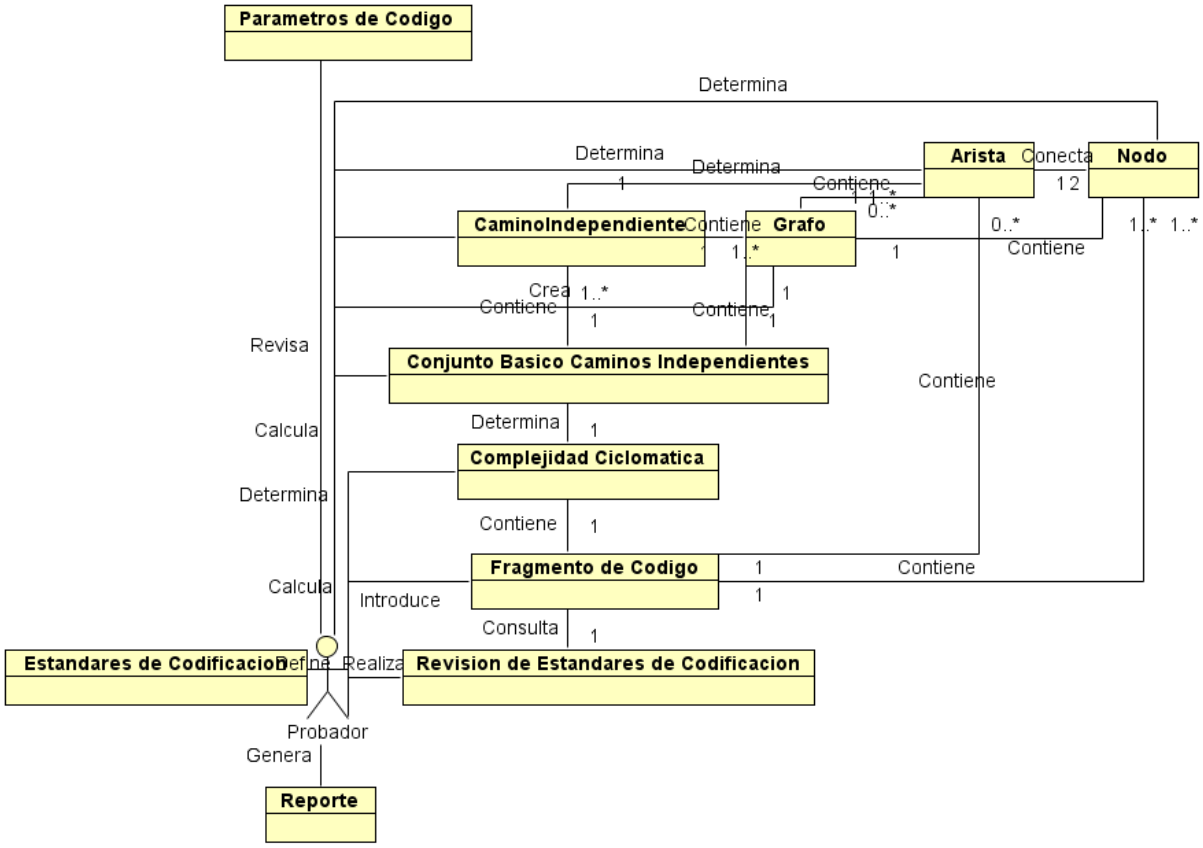

Ilustración 2: Diagrama del Modelo de Dominio.

2.2.3 Diagrama de Clases del Modelo de Objetos.

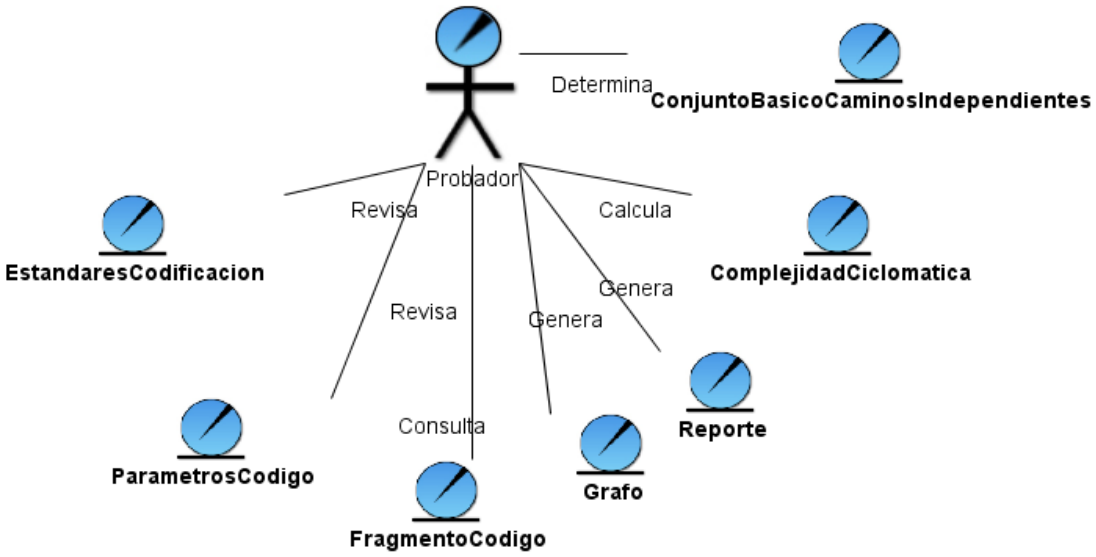

Ilustración 3: Diagrama de Clases del Modelo de Dominio.

2.2.4 Trabajadores del Negocio.

Un trabajador del negocio es la representación de personas o sistemas dentro del negocio, estos efectúan las actividades comprendidas dentro de un caso de uso. Dichos trabajadores se encuentran incluidos en la frontera del negocio y posteriormente serán los usuarios del sistema que se desea implementar.

Tabla 1: Trabajadores del Negocio.

Trabajadores del negocio	Función
Probador	La tarea de este trabajador es introducir el fragmento de código a comprobar y realizarle las pruebas para posteriormente emitir el reporte de resultados de cada prueba. Entre sus funciones también se encuentran establecer los parámetros del código de las pruebas que se llevarán a cabo y los estándares de codificación cuyo cumplimiento será objeto de comprobación.

2.3 Especificación de requisitos de software.

La captura de requerimientos constituye un proceso crítico de la ingeniería de software porque entre los puntos más importantes a tener en cuenta durante el desarrollo de un sistema de software se encuentran los requisitos que debe cumplir la aplicación, estos constituyen los cimientos de la solución propuesta y la evidencia para desarrollar la modelación del sistema.

2.3.1 Requisitos funcionales.

Se denominan requerimientos funcionales a aquellas capacidades o condiciones con las que la aplicación debe cumplir, los mismos especifican funcionalidades que el sistema debe estar apto para efectuar. Estos deben ser redactados de forma comprensible para garantizar una única interpretación por parte de clientes, usuarios y desarrolladores y deben ser medibles y verificables.

Los requerimientos incluyen las acciones que podrán ser realizadas por el usuario, las acciones ocultas que el software debe ejecutar y las condiciones extremas a determinar por el sistema.

A continuación enumeran los requisitos funcionales que no son más que las tareas que el sistema deberá ser capaz de efectuar y las cuales determinan el cumplimiento de los objetivos planteados para esta investigación:

RF-1: Gestionar proyecto.
- **RF-1.1:** Generar proyecto.
- **RF-1.2:** Cargar proyecto.
- **RF-1.3:** Guardar proyecto.

RF-2: Cargar archivo de código.
- **RF-2.1:** Cargar código Java.
- **RF-2.2:** Cargar código C++.
- **RF-2.3:** Cargar código PHP.

RF-3: Calcular parámetros del código.
- **RF-3.1:** Determinar cantidad de líneas de código.
- **RF-3.2:** Calcular densidad de comentarios.

RF-4: Mostrar código.
- **RF-4.1:** Vista por archivos.

RF-4.2: Vista por clases.

RF-4.3: Mostrar atributos de línea de código.

RF-5: Generar grafo.

RF-5.1: Generar grafo.

RF-5.2: Resolver método ausente.

RF-6: Calcular complejidad ciclomática.

RF-6.1: Calcular complejidad ciclomática de métodos por grafo.

RF-6.2: Calcular complejidad ciclomática de métodos por código.

RF-7: Calcular caminos independientes.

RF-7.1: Calcular caminos independientes.

RF-7.2: Calcular conjunto básico de caminos independientes.

RF-8: Mostrar caminos independientes.

RF-8.1: Mostrar caminos independientes.

RF-8.2: Mostrar conjunto básico de caminos independientes.

RF-9: Recorrer camino independiente.

RF-10: Configurar camino independiente.

RF-11: Gestionar señalamientos.

RF-11.1: Agregar señalamiento.

RF-11.2: Eliminar señalamiento.

RF-11.3: Mostrar señalamientos.

RF-12: Revisar estándares de codificación.

RF-13: Gestionar reporte de clase.

RF-13.1: Generar reporte de clase.

RF-13.2: Mostrar reporte de clase.

RF-13.3: Guardar reporte de clase.

RF-13.4: Imprimir reporte de clase.

2.3.2 Requisitos no funcionales.

A continuación se describen un conjunto de cualidades o propiedades que la aplicación debe poseer con el propósito de que cuente con la calidad requerida y de satisfacer las necesidades del centro GEySED, y en particular del Grupo de Calidad de dicho centro:

- **Funcionalidad:**

- El proceso de reescribir el archivo de salva del proyecto debe ocurrir de forma rápida y en el último momento posible con el fin de evitar pérdidas de datos en caso de error.
- La aplicación debe guardar, además de la salva normal, archivos de respaldo.
- La funcionalidad ‘cargar archivo de código’ debe permitir seleccionar y cargar múltiples documentos de código fuente a la vez.

- **Utilización:**
  - La aplicación será sencilla, fácil de usar, amigable e intuitiva para el usuario. La misma no estará cargada de imágenes o colores, solo mostrará referencias a funcionalidades específicas debido a su naturaleza de sistema pensado para trabajar.
  - La interfaz gráfica presentará un flujo sencillo e intuitivo.
  - El lenguaje empleado en la interfaz gráfica no debe contener palabras en otros idiomas.
  - El diseño de la interfaz gráfica debe alinearse a los estándares establecidos en la UCI.
  - El producto final debe contar con la documentación apropiada. En la misma incluirá descripciones de todas las funcionalidades del sistema y una guía para su uso o ayuda para su aprendizaje.
  - La herramienta será construida pensando en su utilización por parte del personal con la responsabilidad de efectuar el rol de probador. El mismo debe contar con los conocimientos fundamentales sobre los métodos de Caja Blanca y estándares de codificación.
  - El software solo será usado dentro de la UCI.
- **Fiabilidad:**
  - La aplicación debe garantizar la integridad de los datos que se manejan.
  - No debe presentar fallos y en un caso extremo debe ser capaz de reducir las pérdidas de información al mínimo evitando reescribir el archivo de salva previo hasta el último momento, lo cual evita la corrupción de los datos guardados.
  - La aplicación debe contar con salvas de respaldo para facilitar la recuperación de datos en caso de fallos.
- **Rendimiento:**

- La aplicación contará con la capacidad de analizar grandes cantidades de líneas de código en pocos segundos.

- **Restricciones de diseño:**
  - Para realizar el diseño de la aplicación se empleará la metodología de desarrollo de software RUP, como herramienta CASE se usará Visual Paradigm 3.4 haciendo uso del lenguaje UML 2.0 o superior.
- **Implementación:**
  - Para la implementación de la aplicación se deberá usar el entorno de desarrollo Qt Creator 2.0.1 o superior y el marco de trabajo Qt 4.7.0 (32 bit) o superior con el lenguaje C++.
  - El código fuente de la aplicación debe cumplir los estándares de codificación establecidos en la UCI.
  - El sistema debe ser multiplataforma y contar con la capacidad de ser instalado en sistemas operativos variados, principalmente en varias versiones de Windows (Ejemplo: Windows 2000, Windows XP) y varias distribuciones de Linux (Ejemplo: Nova 3.0, Ubuntu 10.10).
- **Requisitos físicos:**
  - Procesador de 600 MHz o superior.
  - 128 MB de memoria RAM.
  - Monitor VGA o superior.
  - Ratón de Microsoft o compatible.
  - Espacio en disco duro máximo de 30 MB.

2.4 Modelo de Sistema. Definición de casos de uso.

El Modelo de Casos de Uso del Sistema muestra las funcionalidades que se deben implementar en la aplicación y el entorno en el cual funcionará la aplicación mediante el uso de actores y casos de uso. El mismo crea los cimientos necesarios para el desarrollo del proceso de Análisis y el Diseño del software. La identificación de los casos de uso constituye la guía a seguir por el ingeniero de software al frente del desarrollo de un sistema de software.

2.4.1 Definición de actores.

Los actores del sistema son la representación de aquellas personas o sistemas que fueron trabajadores del negocio y que interactúan de alguna forma con la aplicación y

están asociados al cumplimiento de los requerimientos funcionales o procesos que responden a las funcionalidades definidas para el mismo.

Tabla 2: Actores del sistema.

Actores	Descripción
Usuario	Es el encargado de introducir el código en la herramienta y realizarle las pruebas necesarias, haciendo uso para ello de las funcionalidades que le brinda el sistema con el fin de facilitar su tarea.

2.4.2 Listado de casos de uso.

A continuación se muestra un resumen de los casos de uso que se tienen en cuenta para la correcta comprensión de las funcionalidades del sistema, incluyendo los requerimientos que lo conforman:

Tabla 3: CU_#1: Gestionar proyecto.

Caso de uso #1:	Gestionar proyecto.
Actores:	Usuario.
Resumen:	El caso de uso inicia cuando el usuario desea cargar o guardar un proyecto. El caso de uso, brinda la posibilidad de cargar un proyecto previamente guardado o guardar un proyecto en el que se encuentra trabajando. El caso de uso finaliza, cuando el usuario carga o guarda un proyecto.
Referencias:	RF-1 (RF-1.1, RF-1.2, RF-1.3).
Prioridad:	Crítico.

Tabla 4: CU_#2: Cargar archivo de código.

Caso de uso #2:	Cargar archivo de código.
Actores:	Usuario.
Resumen:	El caso de uso inicia cuando el usuario desea cargar uno o varios archivos de código, permitiendo al usuario cargar archivos de código pertenecientes a cualquiera de los lenguajes soportados para su posterior revisión. El caso de uso termina cuando se han cargado él o los archivos de código.
Referencias:	RF-2(RF-2.1, RF-2.2, RF-2.3)
Prioridad:	Crítico.

Tabla 5: CU_#3: Calcular parámetros del código.

Caso de uso #3:	Calcular parámetros del código.
Actores:	Usuario.
Resumen:	Este caso de uso se inicializa luego de que el usuario selecciona los archivos de código que va a cargar para la aplicación y en él se realizan los cálculos de varias características del código que pueden resultar útiles. El caso de uso termina cuando se han calculado todos los parámetros de todos los archivos de código y se muestran en la interfaz principal.
Referencias:	RF-3 (RF-3.1, RF-3.2).
Prioridad:	Medio.

Tabla 6: CU_#4: Mostrar código.

Caso de uso #4:	Mostrar código.
Actores:	Usuario.
Resumen:	Este caso de uso se inicia cuando el usuario desea cambiar la forma en la que se organiza el código con el que trabaja y brinda al usuario la posibilidad de poder seleccionar la manera en la que se visualiza el código en la interfaz de la aplicación y permite visualizar la información referente a cada línea de código. El caso de uso termina una vez que se muestra el código en la interfaz principal de la manera solicitada por el usuario.
Referencias:	RF-4 (RF-4.1, RF-4.2, RF-4.3).
Prioridad:	Medio.

Tabla 7: CU_#5: Generar grafo.

Caso de uso #5:	Generar grafo.
Actores:	Usuario.
Resumen:	Este caso de uso se inicializa cuando el usuario desea generar el grafo de flujo a partir del código fuente cargado y brinda al usuario la posibilidad de procesar el código que ha cargado en la aplicación y generar el grafo correspondiente al mismo, para su posterior análisis. El caso de uso termina cuando se ha generado el grafo de flujo a partir del código.

Referencias:	RF-5 (RF-5.1, RF-5.2).
Prioridad:	Crítico.

Tabla 8: CU_#6: Calcular complejidad ciclomática.

Caso de uso #6:	Calcular complejidad ciclomática.
Actores:	Usuario.
Resumen:	Este caso de uso se inicia cuando el usuario desea calcular el valor de la complejidad de uno o varios métodos declarados en el código, y brinda la posibilidad de calcular la complejidad ciclomática de los métodos del código mediante varias técnicas. El caso de uso finaliza una vez que se muestran en un cuadro de diálogo los valores calculados.
Referencias:	RF-6 (RF-6.1, RF-6.2).
Prioridad:	Crítico.

Tabla 9: CU_#7: Calcular caminos independientes.

Caso de uso #7:	Calcular caminos independientes.
Actores:	Usuario.
Resumen:	Este caso de uso se inicializa una vez que el usuario desea conocer los caminos de ejecución del código brinda la capacidad de determinar los disímiles caminos independientes contenidos en el grafo y cuáles de ellos

	pertenecen al conjunto básico. El caso de uso finaliza una vez que se han calculado los caminos de ejecución.
Referencias:	RF-7 (RF-7.1, RF-7.2).
Prioridad:	Crítico.

Tabla 10: CU_#8: Mostrar caminos independientes.

Caso de uso #8:	Mostrar caminos independientes.
Actores:	Usuario.
Resumen:	Este caso de uso se inicia cuando el usuario desea ver los caminos de ejecución del grafo de flujo y brinda la posibilidad de visualizar un listado de los caminos independientes calculados, o solo de aquellos contenidos en el conjunto básico. El caso de uso termina cuando se muestran en la interfaz los caminos calculados.
Referencias:	RF-8 (RF-8.1, RF-8.2).
Prioridad:	Crítico.

Tabla 11: CU_#9: Recorrer camino independiente.

Caso de uso #9:	Recorrer camino independiente.
Actores:	Usuario.
Resumen:	Este caso de uso inicia cuando el usuario desea seguir la secuencia de un camino independiente con el fin de revisarlo y brinda la posibilidad de recorrer línea a línea el

	camino de ejecución seleccionado con el objetivo de revisarlo. El caso de uso termina una vez que el usuario cierra la ventana de navegación de caminos.
Referencias:	RF-9.
Prioridad:	Crítico.

Tabla 12: CU_#10: Configurar camino independiente.

Caso de uso #10:	Configurar camino independiente.
Actores:	Usuario.
Resumen:	Este caso de uso se inicia cuando el usuario desea personalizar los atributos de un camino independiente y brinda la posibilidad de marcar el camino de ejecución seleccionado como “Revisado” o “No revisado” y configurar algunos de sus atributos. El caso de uso termina cuando el usuario desea guardar los cambios realizados.
Referencias:	RF-10.
Prioridad:	Medio.

Tabla 13: CU_#11: Gestionar señalamientos.

Caso de uso #11:	Gestionar señalamientos.
Actores:	Usuario.
Resumen:	Este caso de uso se inicia cuando el usuario desea visualizar los señalamientos ligados a una línea de código

	específica y brinda la posibilidad de visualizar, agregar y eliminar comentarios referentes a las líneas que componen el código cargado. El caso de uso termina una vez que el usuario cierra la ventana de señalamientos.
Referencias:	RF-11(RF-11.1, RF-11.2, RF-11.3).
Prioridad:	Crítico.

Tabla 14: CU_#12: Revisar estándares de codificación.

Caso de uso #12:	Revisar estándares de codificación.
Actores:	Usuario.
Resumen:	Este caso de uso inicia cuando el usuario desea que se realice una revisión del cumplimiento de los estándares de codificación establecidos y brinda la posibilidad de verificar los mismos. El caso de uso termina cuando se muestra un listado de los errores encontrados.
Referencias:	RF-12.
Prioridad:	Crítico.

Tabla 15: CU_#13: Gestionar reporte de clase.

Caso de uso #13:	Gestionar reporte de clase.
Actores:	Usuario.
Resumen:	Este caso de uso se inicia cuando el usuario desea resumir visualizar un resumen de los resultados de las pruebas

	realizadas y brinda la posibilidad de ver el reporte de una clase, guardarlo en el ordenador o imprimirlo. El caso de uso termina una vez que el usuario ha cerrado la ventana de reportes.
Referencias:	RF-13 (RF-13.1, RF-13.2, RF-13.3, RF-13.4).
Prioridad:	Medio.

2.4.3 Diagrama de Casos de Uso del sistema.

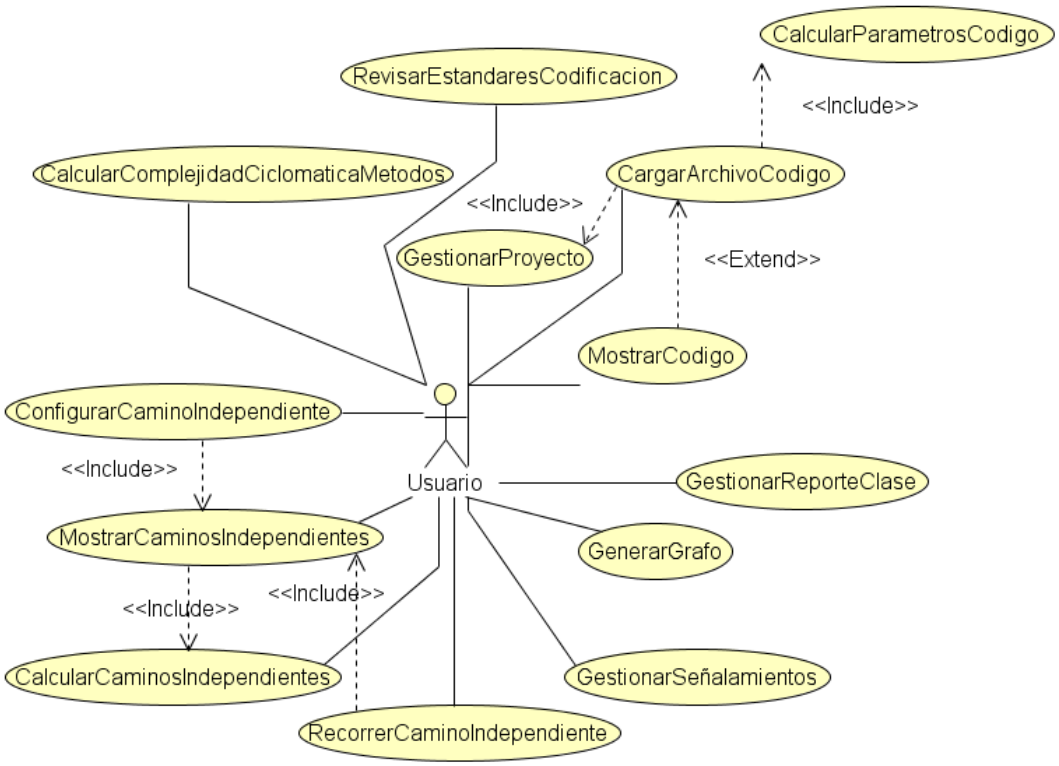

Ilustración 4: Diagrama de Casos de Uso del sistema.

2.4.4 Casos de uso expandidos.

La descripción de casos de uso expandida es una vía para puntualizar las principales características de los mismos, ya que no basta con su representación gráfica. Dichas descripciones se mostrarán en el **¡Error! No se encuentra el origen de la referencia.**, aunque, a continuación, en modo de ejemplo se presentarán dos de estas

descripciones correspondientes a los casos de uso #1: Gestionar proyecto y #2: Cargar archivo de código.

Tabla 16: CUA_#1: Gestionar proyecto.

Caso de Uso:	Gestionar proyecto
Actores:	Usuario
Resumen:	El caso de uso inicia cuando el usuario desea cargar o guardar un proyecto. El caso de uso, brinda la posibilidad de cargar un proyecto previamente guardado o guardar un proyecto en el que se encuentra trabajando. El caso de uso finaliza, cuando el usuario carga o guarda un proyecto.
Precondiciones:	
Referencias	RF-1 (RF-1.1, RF-1.2, RF-1.3).
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el usuario selecciona en el menú la opción “Proyecto”.	1.1. El sistema despliega un menú con las opciones “Nuevo proyecto”, “Cargar proyecto” y “Guardar proyecto”.
2. El usuario selecciona una de las siguientes opciones: • Cargar proyecto. • Guardar proyecto.	
Sección “Cargar proyecto”	
	2.1 El sistema muestra una ventana con las opciones para cargar un

	proyecto.
3. El usuario selecciona el archivo a cargar, ajusta las opciones y presiona el botón “Aceptar”.	3.1. Se carga el proyecto seleccionado, y finaliza el caso de uso.
Flujo Alterno.	
Acción del Actor	**Respuesta del Sistema**
	3.1. El sistema muestra un cuadro de diálogo indicando que no se logró cargar el proyecto seleccionado y los motivos del error.
Sección “Guardar proyecto”	
	2.1 El sistema muestra una ventana con las opciones para guardar un archivo de proyecto.
3. El usuario selecciona la ubicación para guardar, ajusta las opciones y presiona el botón “Aceptar”.	3.1. Se guarda el proyecto seleccionado y finaliza el caso de uso.
Flujo Alterno.	
Acción del Actor	**Respuesta del Sistema**
	3.1. Se muestra un cuadro de diálogo indicando que no se logró guardar el proyecto seleccionado y los motivos del error.
Poscondiciones:	Se carga el proyecto generándose a partir de los archivos de código seleccionados o se guarda un proyecto en el cual se estaba trabajando.

Tabla 17: CUA_#2: Cargar archivo de código.

Caso de Uso:	Cargar archivo de código.
Actores:	Usuario
Resumen:	El caso de uso inicia cuando el usuario desea cargar uno o varios archivos de código, permitiendo al usuario cargar archivos de código pertenecientes a cualquiera de los lenguajes soportados para su posterior revisión. El caso de uso termina cuando se han cargado él o los archivos de código.
Precondiciones:	El código en el archivo debe haber sido procesado por algún compilador previamente.
Referencias	RF-2(RF-2.1, RF-2.2, RF-2.3)
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el usuario selecciona la opción “Cargar archivo de código” en el menú y selecciona una de las siguientes opciones: • Cargar archivo de código Java. • Cargar archivo de código C++. • Cargar archivo de código PHP.	
Sección “Cargar archivo de código Java”	
	1.1. El sistema muestra una ventana con las opciones para cargar uno o varios archivos de código Java.
2. El usuario selecciona el o los archivos a	2.1. El caso de uso finaliza cuando se

cargar, ajusta las opciones y presiona “Aceptar”.	cargan los archivos de código seleccionados y se procesan.
Sección “Cargar archivo de código C++”	
	1.1. El sistema muestra una ventana con las opciones para cargar uno o varios archivos de código C++.
2. El usuario selecciona el o los archivos a cargar, ajusta las opciones y presiona “Aceptar”.	2.1. El caso de uso finaliza cuando se cargan los archivos de código seleccionados y se procesan.
Sección “Cargar archivo de código PHP”	
	1.1. El sistema muestra una ventana con las opciones para cargar uno o varios archivos de código PHP
2. El usuario selecciona el o los archivos a cargar, ajusta las opciones y presiona “Aceptar”.	2.1. El caso de uso finaliza cuando se cargan los archivos de código seleccionados y se procesan.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	2.1. Se muestra un cuadro de diálogo indicando que no se logró cargar el o los archivos de código seleccionados y los motivos del error.
Prototipo de Interfaz	
Poscondiciones:	Quedan cargados los archivos de códigos generando un nuevo proyecto de ser necesario.

2.5 Prototipos de interfaz de usuario.

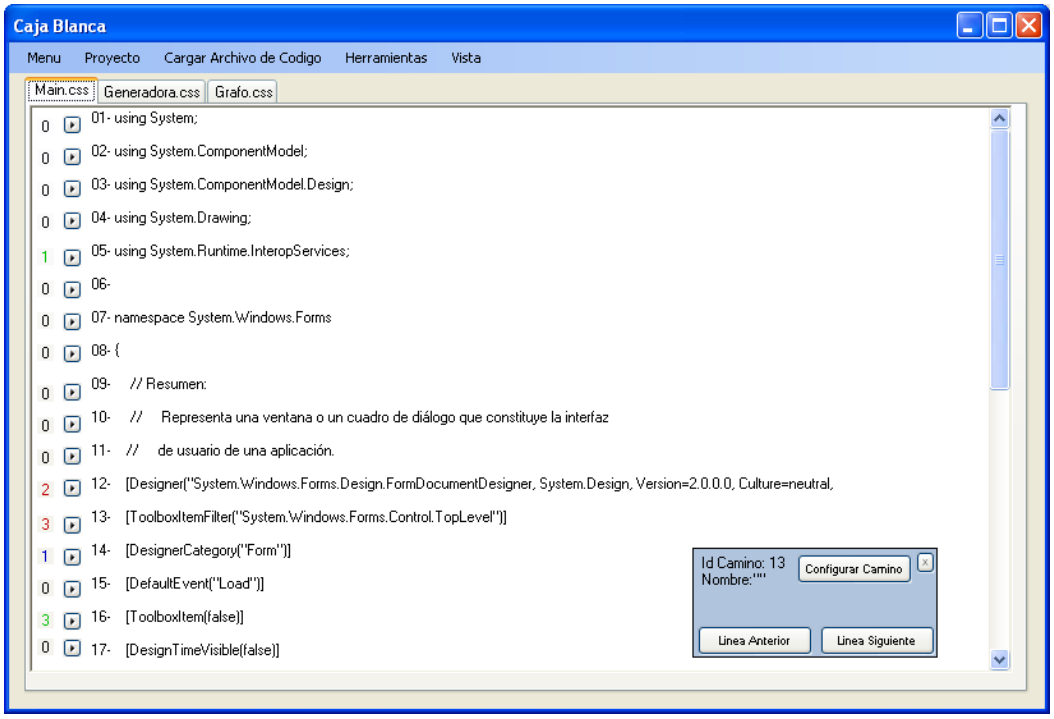

Ilustración 5: Prototipo de interfaz: Principal.

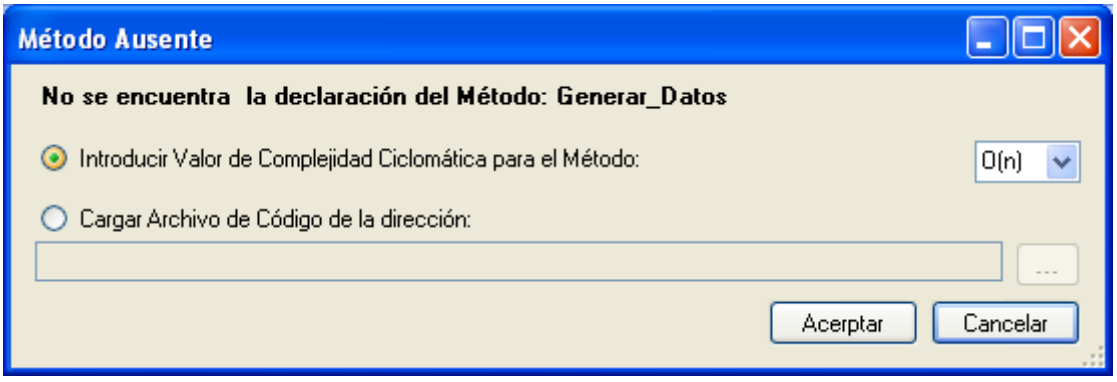

Ilustración 6: Prototipo de interfaz: Resolver método ausente.

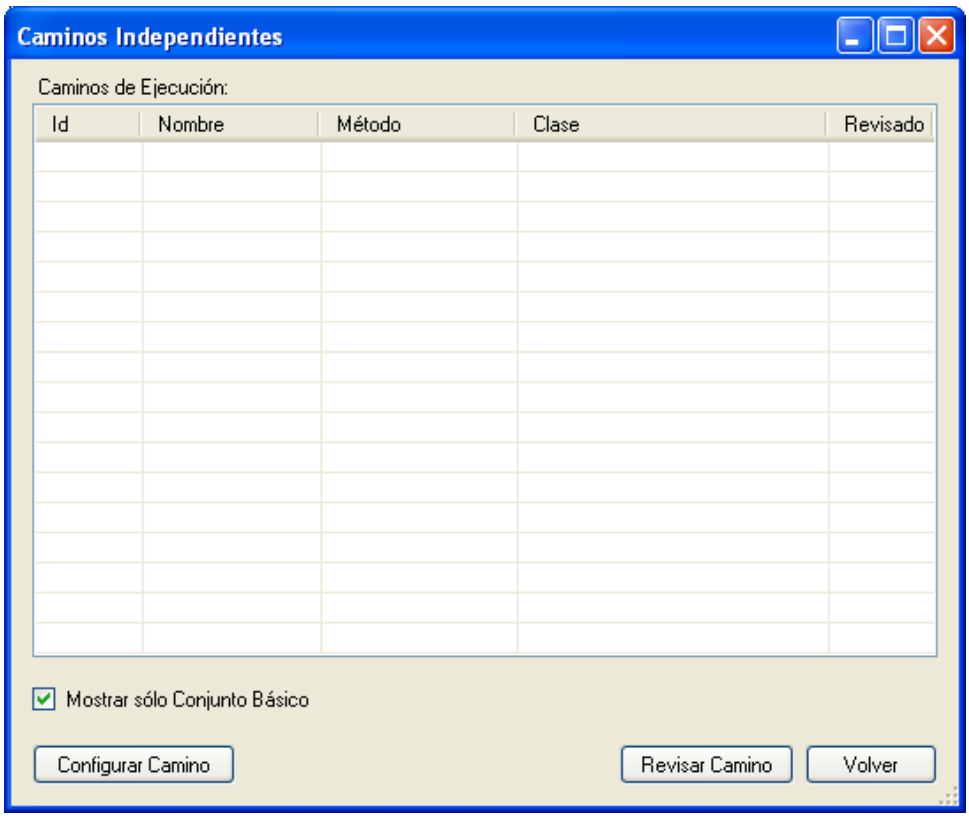

Ilustración 7: Prototipo de interfaz: Mostrar caminos independientes.

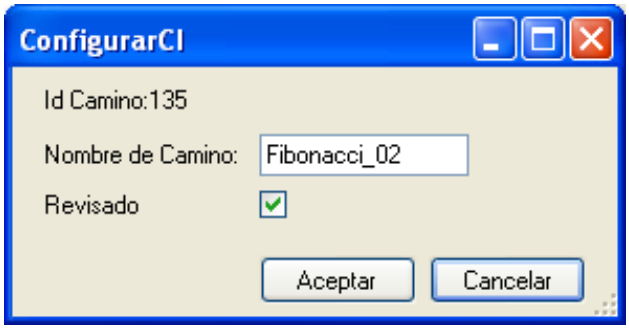

Ilustración 8: Prototipo de interfaz: Configurar camino independiente.

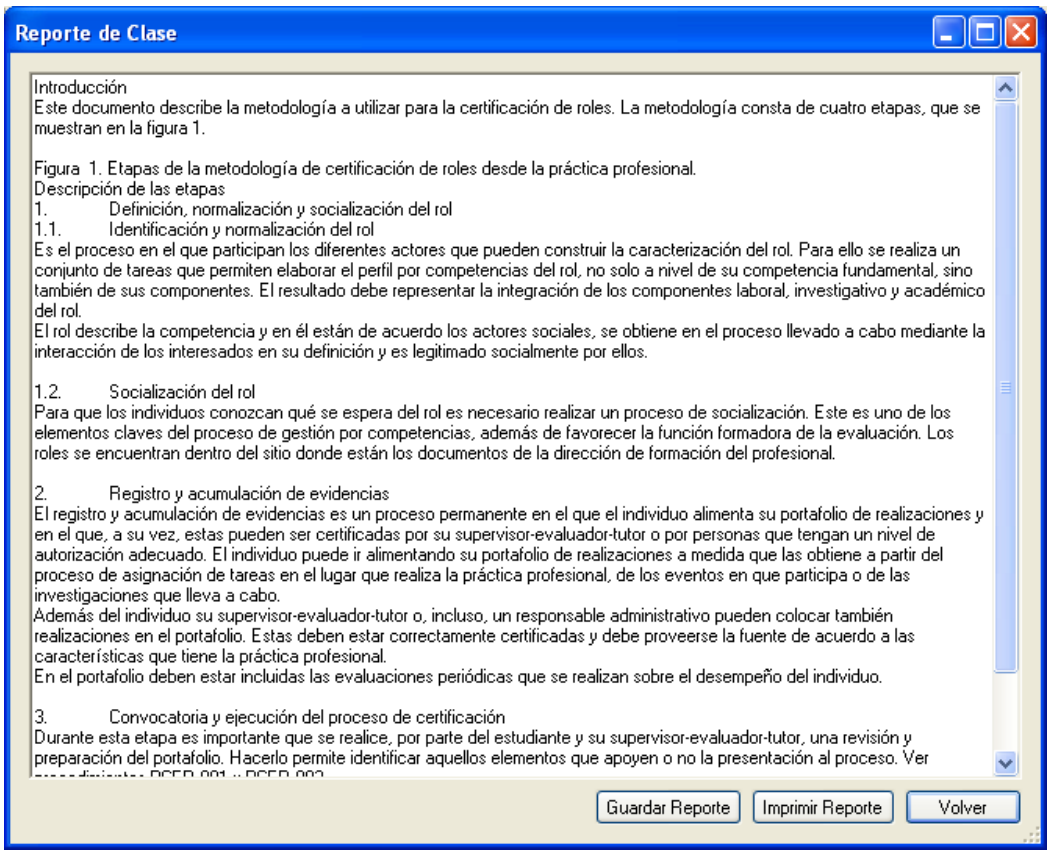

Ilustración 9: Prototipo de interfaz: Mostrar reporte.

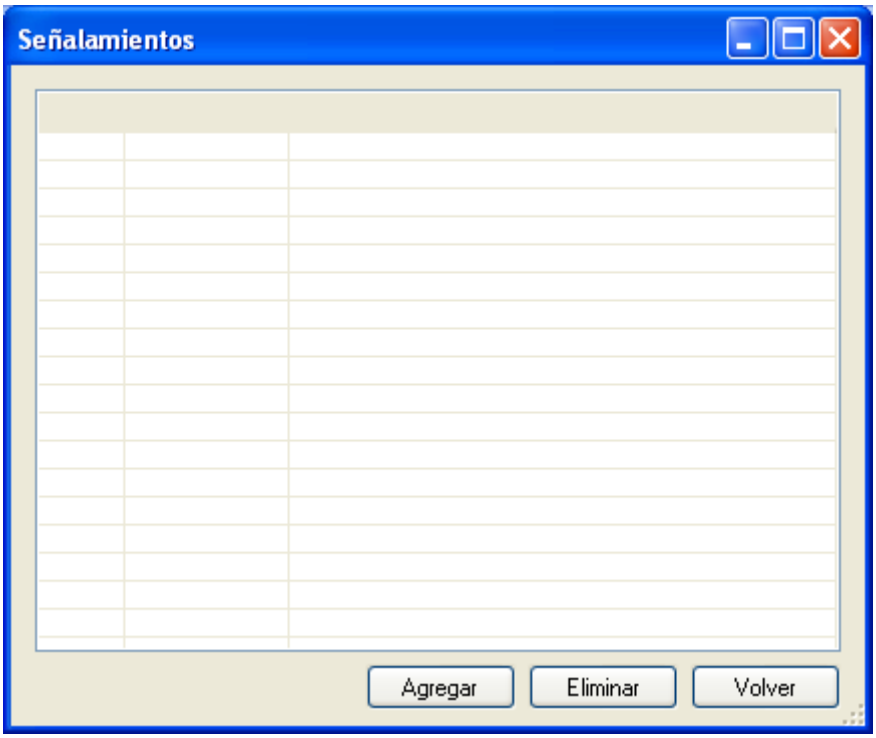

Ilustración 10: Prototipo de interfaz: Señalamientos.

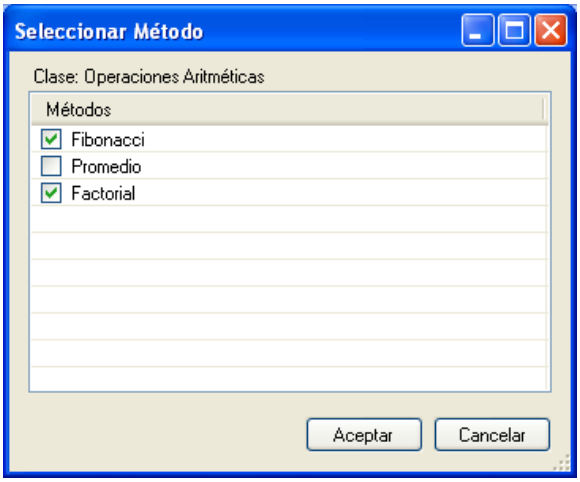

Ilustración 11: Prototipo de interfaz: Seleccionar métodos.

Conclusiones

En este capítulo se abordaron temas relacionados con la propuesta de herramienta para la automatización de diferentes métodos de prueba de Caja Blanca, con el objetivo de optimizar el necesario proceso de revisiones del software producido en el centro GEySED de la Facultad 6 para garantizar la calidad de los mismos.

Se incluye la modelación del dominio, para mostrar de una forma clara y sencilla los diferentes procesos que ocurren en el mismo para que los usuarios finales y los desarrolladores obtengan un entendimiento común. Se abordaron, en este tema, los conceptos fundamentales a tener en cuenta, se identificaron y describieron los trabajadores del Negocio y se logró el modelado de los objetos del Negocio.

En este capítulo se identificaron también los requerimientos del sistema, funcionales y no funcionales, a tener en cuenta para la implementación de la aplicación. Se concretaron los casos de uso del sistema, los cuales se representaron en un diagrama el cual contiene, además, la interacción que se desarrolla entre ellos. Posteriormente se realizó la descripción de cada uno de estos casos de uso y de los actores del sistema, con el fin de brindar una noción general de la concepción y el funcionamiento del sistema.

CAPÍTULO 3: Análisis y Diseño del sistema.

En el presente capítulo se lleva a cabo el flujo de Análisis y Diseño que es uno de los flujos de trabajo de la metodología utilizada en la modelación de la herramienta: RUP. En dicha metodología se compactan los procesos de Análisis y Diseño en uno solo, pero cada uno de ellos cumple funcionalidades diferentes, las cuales conforman la vista lógica de la arquitectura de software en la construcción de aplicaciones dirigida por modelos.

Se realizará el Análisis del sistema, modelando importantes diagramas como son: el Diagrama de Análisis y el Diagrama de Clases del Diseño y posteriormente se procederá a la modelación de la lógica del Negocio mediante el uso de clases, se tratan los principios del Diseño de la aplicación, donde se incluyen los Diagramas de Clases del Diseño y los Diagramas de Secuencia.

3.1 Flujos de trabajo de Análisis y Diseño.

El principal objetivo del flujo de Análisis y Diseño es transformar los requisitos a una especificación que describa cómo implementar el sistema. El Análisis es, fundamentalmente, obtener una apreciación de las funcionalidades que poseerá la herramienta a desarrollar, para lo cual se apega a los requerimientos funcionales. El Diseño, por su parte, logra un mayor refinamiento al tener en cuenta los requisitos no funcionales, y se concentra más en 'cómo' el sistema cumple sus funciones.

Los objetivos de este flujo de trabajo son:

- Transformar los requerimientos al diseño de la futura aplicación.
- Desarrollar una arquitectura para el software.
- Adaptar el diseño para que sea consistente con el entorno de implementación.

3.2 Modelo de Análisis.

El Modelo de Análisis se ocupa del procesamiento de los requisitos funcionales, creando un bosquejo de cómo llevar a cabo el funcionamiento dentro del sistema, incluidas las funciones significativas para la arquitectura. Este modelo sirve como un primer acercamiento al Diseño y es la derivación del análisis de los casos de uso.

El Modelo de Análisis abarca las clases que describen la realización de los casos de uso, sus atributos y las relaciones entre ellas, los cuales se identifican durante la

construcción de dicho modelo. Con esta información se edifican los Diagramas de Clases del Análisis, que constituyen los conceptos en un dominio del problema, sin llegar al nivel de detalle que se logra en el Diseño.

3.2.1 Diagramas de Clases del Análisis.

Los Diagramas de Clases del Análisis muestran la relación entre las clases, centradas en los requisitos funcionales, las mismas poseen atributos y establecen relaciones de asociación, agregación/composición, generalización/especialización y tipos asociativos con otras clases. En la metodología RUP se clasifican las clases en:

- Clases Interfaz: Modelan la interacción entre la aplicación y los actores.
- Clases Controladoras: Coordinan la realización de uno o unos pocos casos de uso, regulando las actividades de los objetos que implementan la funcionalidad del caso de uso.
- Clases Entidad: Manejan la información que posee larga vida y que es, frecuentemente, persistente.

A continuación se presentarán los Diagramas de Clases del Análisis de los casos de uso: ‘Generar grafo’ y ‘Calcular caminos independientes’, otros Diagramas de Clases del Análisis referentes a los demás casos de uso se expondrán en el **¡Error! No se encuentra el origen de la referencia..**

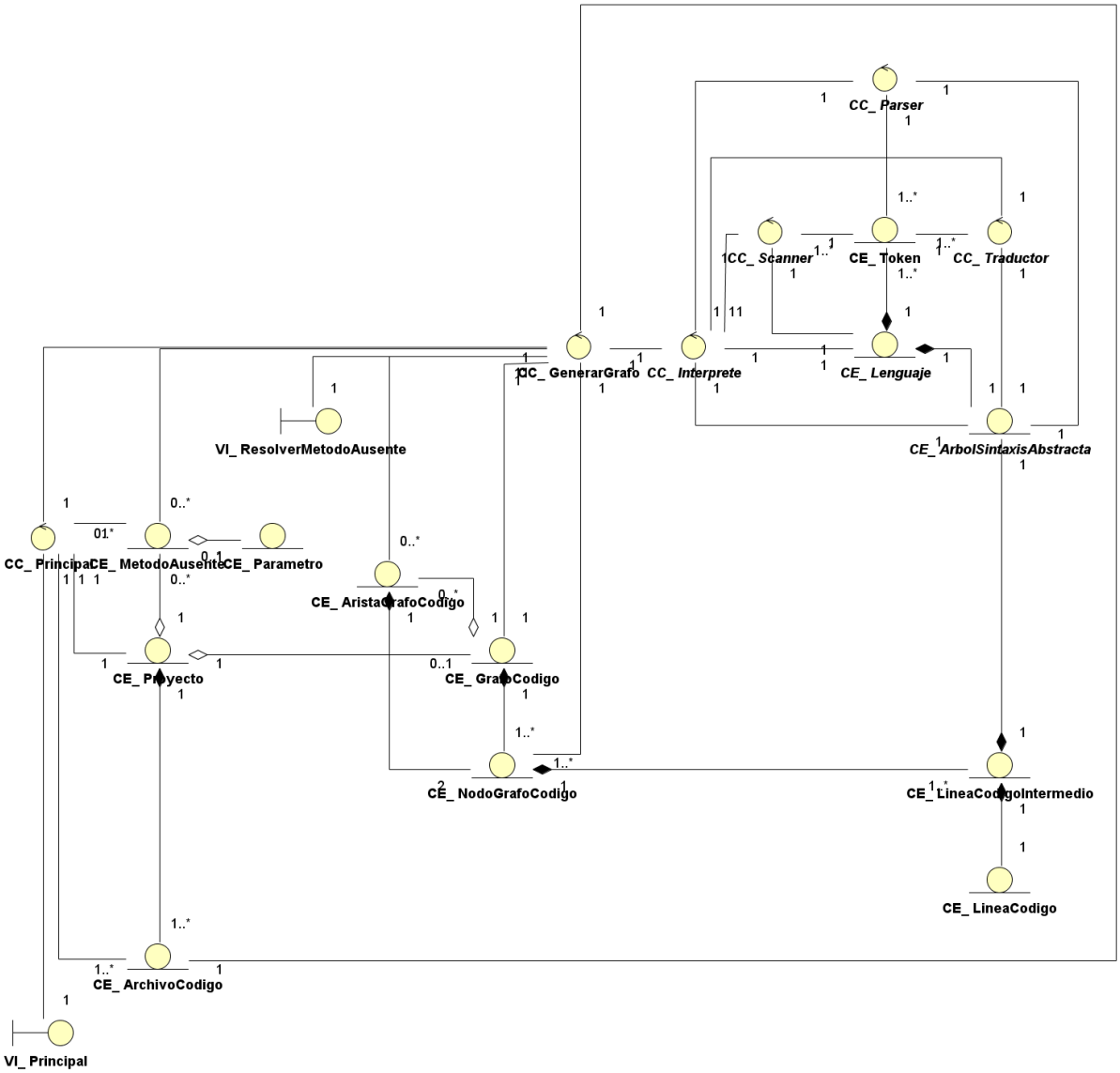

Ilustración 12: DCA_CU: Generar grafo.

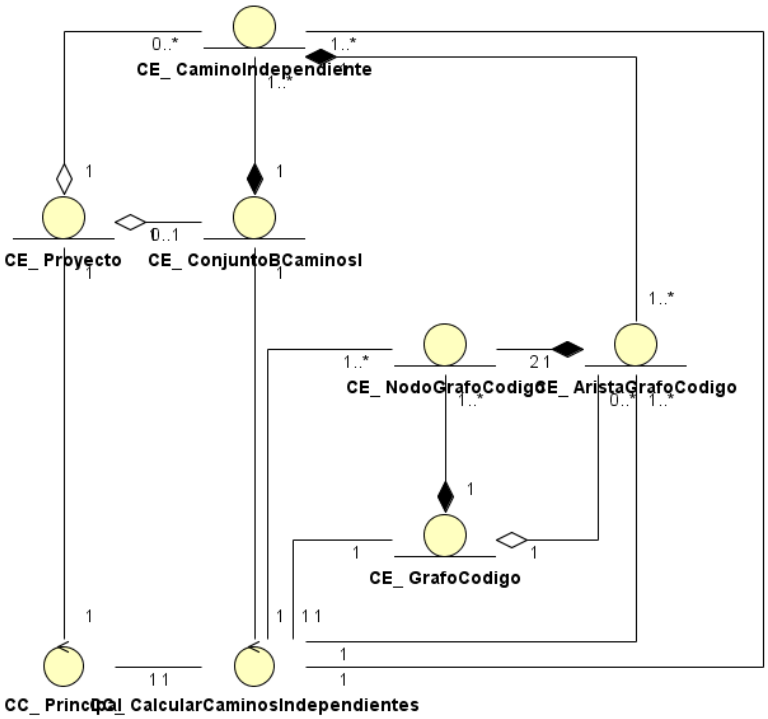

Ilustración 13: DCA_CU: Calcular caminos independientes.

3.2.2 Diagramas de Comunicación.

Los Diagramas de Comunicación de la mayor parte de los casos de uso se pueden encontrar en el **¡Error! No se encuentra el origen de la referencia.**, sin embargo se incluirán en este epígrafe los de los dos primeros casos de uso: 'Calcular caminos independientes' y 'Calcular complejidad ciclomática'.

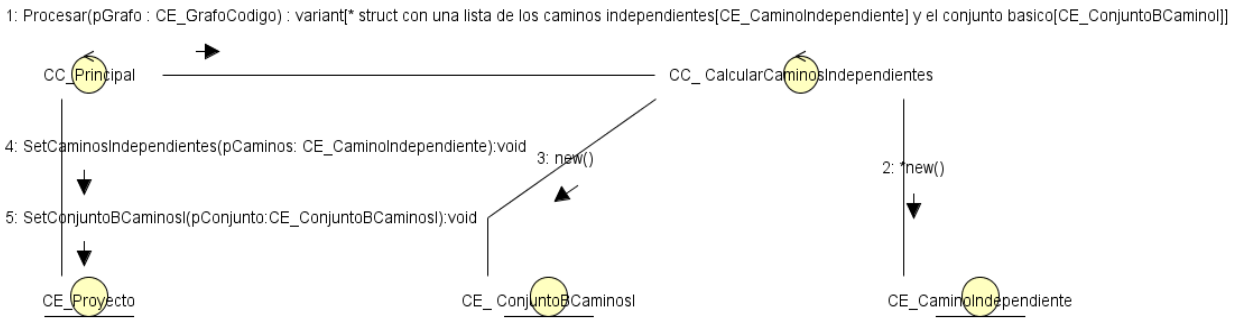

Ilustración 14: DC_CU: Calcular caminos independientes.

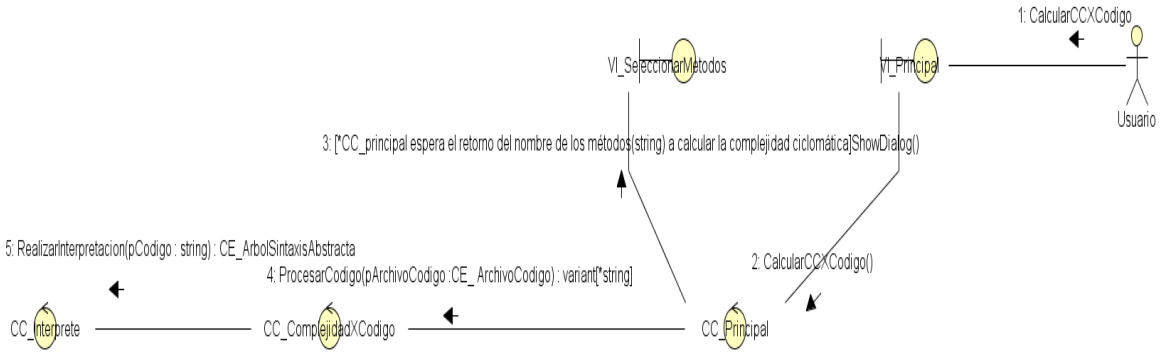

Ilustración 15: DC_CU: Calcular complejidad ciclomática (Escenario: Complejidad ciclomática por código).

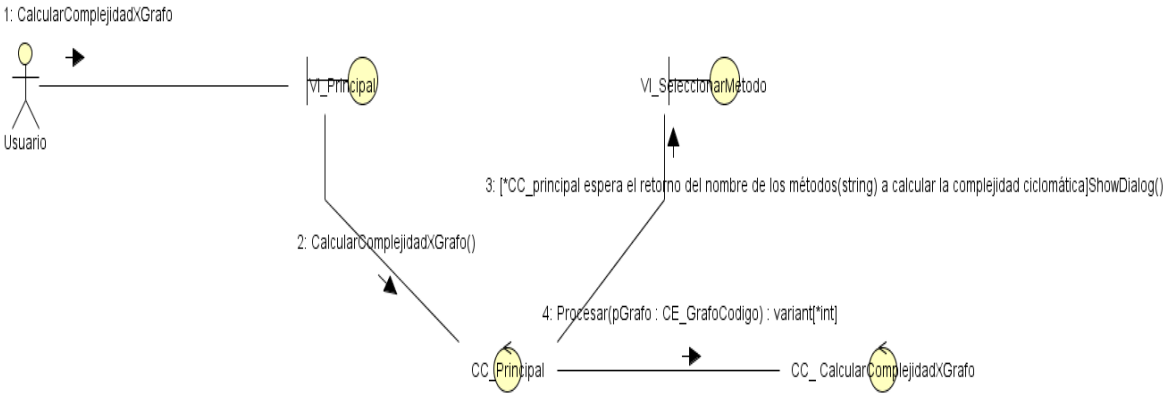

Ilustración 16: DC_CU: Calcular complejidad ciclomática (Escenario: Complejidad ciclomática por grafo).

3.3 Modelo de Diseño.

El Modelo de Diseño describe la realización física de los casos de uso y se centra en el impacto que tienen los requisitos, funcionales y no funcionales, en el sistema a implementar. Entre sus objetivos se encuentran:

- Crear una entrada y un punto de partida para la implementación.
- Descomponer las tareas de la implementación en secciones más manejables que puedan ser realizadas por diferentes equipos de desarrolladores.
- Lograr una clara comprensión de los aspectos referentes a los requisitos funcionales y no funcionales, las limitaciones ligadas a las tecnologías y herramientas a emplear.

La realización de casos de uso del Diseño incluye una descripción de los flujos de eventos textual, Diagramas de Clases y Diagramas de Interacción. Los Diagramas de

Clases constituyen el artefacto más frecuentemente empleado en el modelado de aplicaciones orientadas a objetos. Estos representan las clases que serán usadas dentro del sistema, las cuales se especifican utilizando la sintaxis del lenguaje de programación elegido, y las relaciones que se establecen entre las mismas.

3.3.1 Principios del Diseño.

- **Interfaz de usuario:** La interfaz ideada para la herramienta se encuentra basada en el estándar de ventanas. El formato de la fuente será Microsoft Sans Serif con estilo regular y tamaño 8.25 para el texto de las etiquetas, los mensajes de los formularios y menús y el diseño de la interfaz debe ser apropiado para el tipo de usuario (Que, generalmente, encarnará el rol de probador) con la meta de lograr una fácil comprensión del mismo en el lenguaje empleado y las opciones que este brinda. La aplicación debe mostrar una barra de menú en la parte superior donde se localizarán las opciones de trabajo.
- **Ayuda:** El vínculo para acceder a este importante segmento de cualquier herramienta de software se situará en la barra de menú de la aplicación y a la misma también se podrá ingresar presionando la tecla ‘F1’. La ‘Ayuda’ incluirá temas relacionados a la utilización y manejo del sistema tratando de englobar respuestas para cualquier duda del usuario.
- **Tratamiento de errores:** Este se debe tener en cuenta en el diseño de la interfaz de usuario y se debe lograr que los mensajes de error que el sistema emita sean comprensibles para el usuario, describan claramente la situación y alerten de los posibles riesgos de las acciones que realice.
- **Extensibilidad:** El Diseño de la aplicación debe tener en cuenta la posibilidad de adicionar nuevas funcionalidades al sistema de la manera más sencilla posible y se deben crear las condiciones necesarias para esta tarea.

3.3.2 Patrones de Diseño.

Un patrón es una pareja problema/solución con un identificador, que estandariza buenos principios y sugerencias. Los patrones pueden ser aplicados a otros contextos, por supuesto teniendo en cuenta las especificaciones sobre la forma de emplearlo en esas situaciones. El objetivo de crear patrones es crear un idioma común a una comunidad de desarrolladores para comunicar experiencia relacionada a los problemas y sus soluciones. Pueden referirse a disímiles niveles de abstracción (11).

3.3.3 Fundamentación del empleo de patrones.

Evitan la creación de sistemas y componentes frágiles, difíciles de mantener, entender, reutilizar y/o extender, disminuyendo las posibilidades de que ocurran decisiones poco atinadas, lo cual se denota en la gran variación que ocurre en la calidad de la interacción entre los objetos y la asignación de responsabilidades. Es propio de diseñadores expertos en orientación a crear objetos ir conformando un vasto repertorio de principios generales y de expresiones que los guían al crear aplicaciones de software, a algunos de estos los definen como patrones y los codifican en un formato ordenado que detalla el inconveniente y su solución.

En la actualidad el uso de patrones se ha convertido en una práctica común, ya que son empleados y se encuentran en la mayoría de los mejores sistemas informáticos a nivel mundial.

En particular un patrón de diseño es una abstracción de una solución en un nivel alto que soluciona problemas que existen en muchos niveles de abstracción.

“Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable” (12).

3.3.4 Patrones.

- **Experto:** Plantea que el experto en información siempre debe ser el encargado de la información, es decir, la creación de un objeto recae sobre la clase que contiene toda la información necesaria para crearlo.
- **Creador:** Como consecuencia de la frecuencia con la que se crean objetos en sistemas orientados a estos, es entonces conveniente seguir un principio general para asignar las responsabilidades referentes a ella. Identifica quien debe ser el responsable de la creación de nuevos objetos o clases.
- **Alta cohesión:** *“*Reduce la dependencia, la información que almacena una clase debe de ser coherente y está en la mayor medida de lo posible relacionada con la clase. La responsabilidad se reparte en varias clases y existe una fuerte colaboración entre ellas, es muy eficiente” (13).
- **Bajo acoplamiento:** Se refiere a asignar la menor cantidad de responsabilidades a cada clase con el fin de evitar que esta recurra a muchas otras. Crear clases más independientes reduce el impacto al cambio y son más reutilizables.

- **Controlador: "**Evita el acceso directo a las clases entidades, evita que la capa de presentación no maneje los eventos del sistema" (14).

Con el propósito de distribuir las responsabilidades de las clases y establecer las relaciones entre ellas, para evitar sobrecargas de funcionalidades así como que exista mucha dependencia con otras clases, es que se han aplicado los patrones mencionados a los diagramas de clases elaborados. Estos permiten la reutilización del conocimiento, su uso como guía para el desarrollo y garantiza un producto robusto y acabado.

3.3.5 Diagrama de Clases del Diseño.

En los Diagramas de Clases del Diseño se representan, de una manera simple y de fácil comprensión, los atributos y métodos principales de cada una de las clases que componen el sistema, sus relaciones y responsabilidades.

Estos diagramas se utilizan para modelar la Vista del Diseño Estática de un sistema, lo que incluye modelar el vocabulario del sistema y las colaboraciones o esquemas. Constituyen también la base para los Diagramas de Componentes y de Despliegue. Los Diagramas de Clases para visualizar, especificar y documentar modelos estructurales, pero también para la implementación de sistemas ejecutables, mediante el uso de ingeniería directa o inversa.

A continuación se presentarán algunos fragmentos del Diagrama de Clases del Diseño. Los diagramas restantes se incluirán en el **¡Error! No se encuentra el origen de la referencia..**

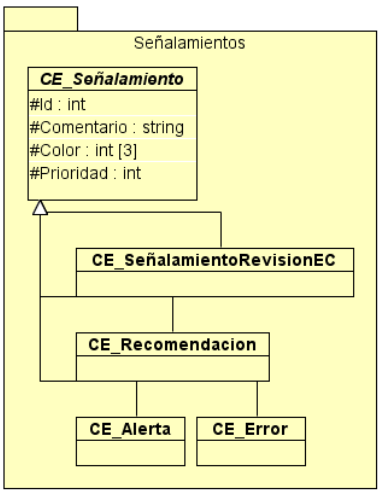

Ilustración 17: DCD: Señalamientos.

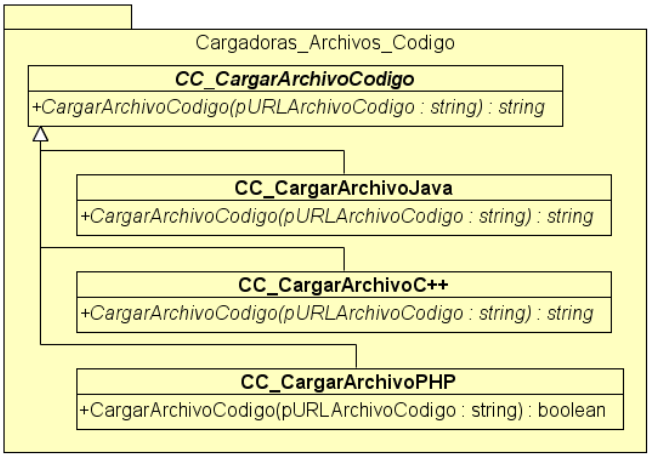

Ilustración 18: DCD: Cargadoras archivos código.

3.3.6 Diagramas de Secuencia.

Los Diagramas de Interacción se emplean en la modelación de los aspectos dinámicos de una aplicación, lo cual abarca modelar instancias concretas o prototípicas de clases, interfaces, componentes y nodos, así como los mensajes enviados entre ellos, en el contexto de un escenario que muestre su comportamiento.

“Los Diagramas de Interacción pueden utilizarse para visualizar, especificar, construir y documentar la dinámica de una sociedad particular de objetos, o se pueden utilizar para modelar un flujo de control particular de un caso de uso” (15).

Un Diagrama de Secuencia representa las interacciones entre objetos organizadas temporalmente atendiendo al orden en que estas ocurren, de esta forma se hace de fácil comprensión para los clientes. Incluyen detalles de implementación del escenario, abarcando los objetos y clases que se utilizan en la realización del escenario y los mensajes enviados entre los objetos.

Se presentarán los Diagramas de Secuencia de los casos de uso: ‘Calcular complejidad ciclomática’ y ‘Calcular caminos independientes’. En el **¡Error! No se encuentra el origen de la referencia.** podrán ser consultados los diagramas de los demás casos de uso.

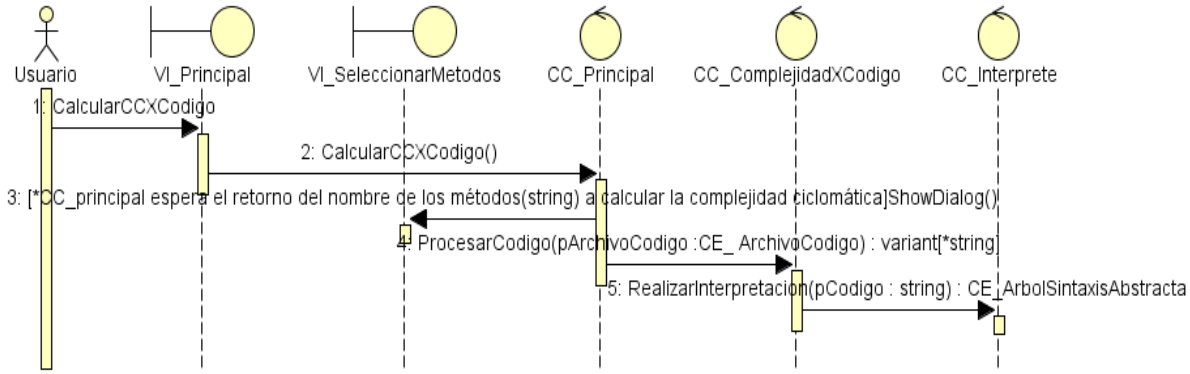

Ilustración 19: DS_CU: Calcular complejidad ciclomática (Escenario: Complejidad ciclomática por código).

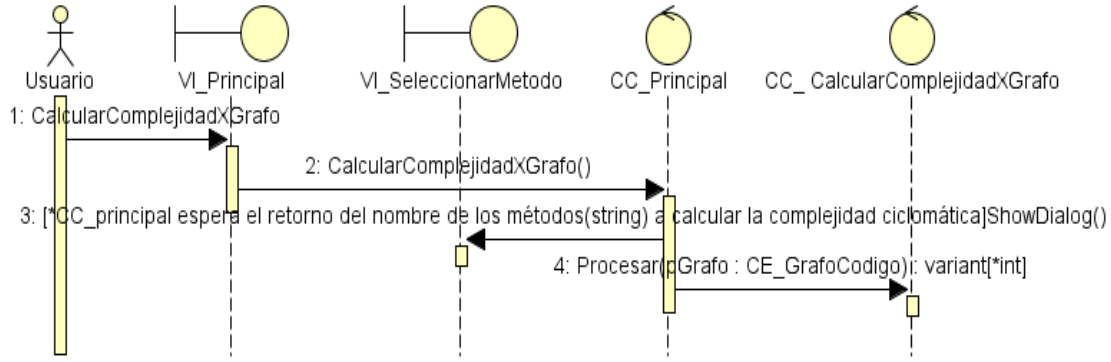

Ilustración 20: Calcular complejidad ciclomática (Escenario: Complejidad ciclomática por grafo).

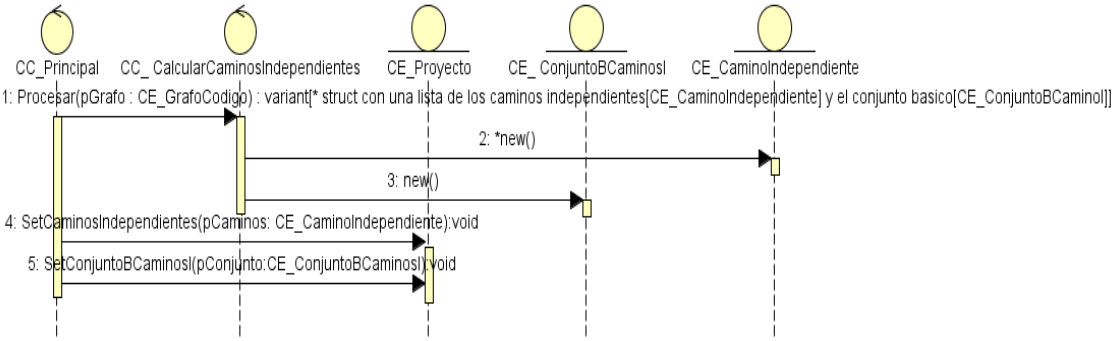

Ilustración 21: DS_CU: Calcular caminos independientes.

3.4 Breve estudio de factibilidad.

Durante el proceso de desarrollo de software una de las fases más importantes es la Planificación del Proyecto. Esta abarca tareas como son estimación de resultados, tiempo de desarrollo y consumo de recursos.

Es importante, por tanto, para la realización de un proyecto tasar el esfuerzo humano, el tiempo de desarrollo del sistema y su costo. A continuación se llevará a cabo un estudio de factibilidad, haciendo uso del método: 'Análisis de puntos de casos de uso', de la aplicación a implementar, el cual posibilita la documentación de los requisitos del sistema en términos de actores y casos de uso

3.5 Planificación mediante puntos de casos de uso.

Este método de planificación, inicialmente, fue propuesto por Gustav Karner y luego muchos otros autores contribuyeron a su refinamiento. El mismo trata de la estimación del tiempo de desarrollo mediante la asignación de "pesos" a un cierto número de factores que lo afectan, para, luego, registrar el tiempo total estimado para el proyecto a partir de dichos factores.

La planificación mediante puntos de casos de uso consiste en la realización de una secuencia de pasos que se desarrolla a continuación:

Paso #1: Consiste en el cálculo de los puntos de casos de uso sin ajustar. Este valor se determina según la siguiente ecuación:

UUCP = **UAW** + **UUCW**

Donde:

UUCP: Puntos de casos de uso sin ajustar.

UAW: Factor de peso de los actores sin ajustar.

UUCW: Factor de peso de los casos de uso sin ajustar.

Factor de peso de los actores sin ajustar (UAW):

Este valor se obtiene mediante un análisis de la cantidad de actores presentes en el sistema y su complejidad. La complejidad de los actores se define teniendo en cuenta, en primer lugar, si se trata de una persona u otro sistema, y luego, la forma en que interactúa con la aplicación. En este caso el usuario es el único actor presente, es de tipo complejo y tiene un peso de 3, debido a que se trata de una persona usando la herramienta a través de una interfaz.

Tabla 18: Factor de peso de los actores sin ajustar.

Tipo de actor	Descripción	Factor de peso	Actores	Total
Simple	Sistemas ajenos que interactúan con la aplicación haciendo uso de una interfaz de programación.	**1**	**0**	**0**
Medio	Sistemas ajenos que interactúan con la aplicación mediante un protocolo o una interfaz basada en texto.	**2**	**0**	**0**
Complejo	Persona que interactúa con la aplicación mediante una interfaz gráfica.	**3**	**1**	**3**

Por tanto el valor de la UAW es **3.**

Factor de peso de los casos de uso sin ajustar (UUCW):

El factor de peso de los casos de uso sin ajustar se define analizando la cantidad de casos de uso presente y su complejidad, la misma se establece teniendo en cuenta la cantidad de transacciones efectuadas en el caso de uso. Se considera una transacción a una secuencia de actividades atómica, o sea, se efectúa la secuencia de actividades completa, o no se efectúa ninguna. En la aplicación existen 15 casos de uso de tipo simple que tienen valor 5.

Tabla 19: Pesos de los casos de uso sin ajustar.

Tipo de CU	Descripción	Peso	Cantidad de CU	Total

Simple	Casos de uso que contienen de 1 a 3 transacciones.	**5**	**13**	**65**
Medio	Casos de uso que contienen de 4 a 7 transacciones.	**10**	**0**	**0**
Complejo	Casos de uso que contienen 8 o más transacciones.	**15**	**0**	**0**

Por tanto el factor de peso de los casos de uso es:

UUCW = **9** x **5** = **65**

Luego los casos de uso sin ajustar dan como resultado:

UUCP = **UAW** + **UUCW** = **3** + **65** = **68**

Paso #2: Una vez que se obtienen los puntos de casos de uso sin ajustar, se debe ajustar este valor haciendo uso de la ecuación

UCP = **UUCP** x **TCF** x **EF**

Donde:

UCP: Puntos de casos de uso ajustados.

UUCP: Puntos de casos de uso sin ajustar.

TCF: Factor de complejidad técnica.

EF: Factor de ambiente.

Factor de complejidad técnica (TCF).

Este coeficiente se obtiene al cuantificar un grupo de factores que definen la complejidad técnica del sistema. Estos factores pueden tener un valor entre 0 y 5, donde 0 significa un aporte irrelevante y 5 uno muy importante.

El cálculo de la TCF se realiza con la siguiente fórmula:

TCF = 0.6 + 0.01 x ∑ (**Pesoi** x **Valori**), donde '**Valor**' es un número del 0 al 5.

Los pesos se obtienen empleando la siguiente tabla:

Tabla 20: Pesos de los factores de complejidad técnica.

Factor	Descripción	Peso	Valor asignado	Total	Comentario
T1	Sistema distribuido.	**2**	**0**	**0**	El sistema es centralizado.
T2	Tiempo de respuesta.	**1**	**4**	**4**	La velocidad es rápida.
T3	Eficiencia del usuario final.	**1**	**5**	**5**	Escasas restricciones de eficiencia.
T4	Funcionamiento interno complejo.	**1**	**3**	**3**	Algunos cálculos de cierta complejidad.
T5	El código debe ser reutilizable.	**1**	**4**	**4**	Se requiere que el código sea reutilizable.
T6	Facilidad de instalación.	**0.5**	**5**	**2.5**	Algunos requerimientos de facilidad de instalación.
T7	Facilidad de uso.	**0.5**	**5**	**2.5**	Fácil de usar.
T8	Portabilidad.	**2**	**3**	**6**	Se requiere que el

					sistema sea portable.
T9	Facilidad de cambio.	**1**	**4**	**4**	Si hay un cambio en el sistema se incurre en algunos gastos.
T10	Concurrencia.	**1**	**0**	**0**	No hay concurrencia.
T11	Incluye objetivos especiales de seguridad.	**1**	**0**	**0**	Seguridad normal.
T12	Provee acceso directo a terceras partes.	**1**	**0**	**0**	No tiene accesos directos a terceras partes.
T13	Se requieren facilidades especiales de entrenamiento de usuarios.	**1**	**1**	**1**	Sistema fácil de usar.

Sumatoria = **32**

TCF = 0.6 + 0.01 x **32** = **0.92**

Factor de ambiente (EF).

Estos agentes se avistan en el cálculo del factor de ambiente. La resolución de estos es similar al del factor de complejidad técnica, o sea, se trata de un conjunto de factores que se les asigna un valor entre el 0 y el 5.

El procedimiento es el siguiente:

EF = 1.4 – 0.03 x ∑ (**Pesoi** x **Valori**), donde '**Valor**' es una cifra del 0 al 5.

A continuación la tabla para obtener los pesos:

Tabla 21: Pesos de los factores del ambiente.

Factor	Descripción	Peso	Valor asignado	Total	Comentario
E1	Familiaridad con el modelo de proyecto utilizado.	**1.5**	**2**	**3**	Se está algo familiarizado con el tema del modelo.
E2	Experiencia en la aplicación	**0.5**	**1**	**0.5**	Primera vez que se trabaja en la aplicación.
E3	Experiencia en la orientación a objetos.	**1**	**4**	**4**	Se ha programado orientado a objetos.
E4	Capacidad del analista líder.	**0.5**	**4**	**2**	Tiene una adecuada capacidad.
E5	Motivación.	**1**	**5**	**5**	Altamente motivado.
E6	Estabilidad de requerimientos	**2**	**4**	**8**	Se esperan cambios.
E7	Personal part–time	**-1**	**0**	**0**	El personal es full-time.
E8	Dificultad del lenguaje de	**-1**	**4**	**-4**	El lenguaje que se utiliza es C++.

	programación				

Sumatoria = **18.5**

EF = 1.4 – 0.03 x **18.5** = **0,845**

Quedando, entonces, como puntos de casos de uso ajustados:

UCP = **68** x **0.92** x **0.845** = **52.8632**

Paso #3: Conversión de puntos de casos de uso a la estimación del esfuerzo.

Gustav Karner originalmente sugirió que cada punto de casos de uso requiere 20 horas – hombre, planteamiento que luego fue refinado con el objetivo de obtener una respuesta más precisa:

El factor de conversión (**CF**) resulta 20 horas/hombre.

Luego se calcula el esfuerzo estimado horas – hombre.

E = **UCP** x **CF** = **52.8632** x **20** = **1057.264** horas/hombre.

Teniendo en cuenta que este resultado no representa más que una mera estimación del esfuerzo horas – hombre, contemplando solo el desarrollo de la funcionalidad especificada en los casos de uso.

Finalmente, para lograr una evaluación más completa de la duración total del producto, hay que añadir al cálculo del esfuerzo obtenido por los puntos de casos de uso, las estimaciones de esfuerzo de las restantes actividades relacionadas al desarrollo de la herramienta. Además se considera que este esfuerzo representa un porcentaje del esfuerzo total del proyecto, de acuerdo a los siguientes valores porcentuales:

Tabla 22: Horas - hombre en cada fase del desarrollo del software.

Actividad	Porcentaje	Horas – hombre
Análisis	10	**264.316**
Diseño	20	**528.632**
Implementación	40	**1057.264**

Pruebas	15	**396.474**
Sobrecargas	15	**396.474**
Total	100	**2643.16**

Paso #4: Conversión a hombres – mes.

Dado que la jornada laboral consta de 8 horas y un mes de trabajo contiene aproximadamente 24 días, una persona trabaja 192 horas en un mes, entonces:

Et = **E** (Horas - hombres) / 192 horas – mes.

El resultado sería **Et** = **2643.16** (Horas - hombres) / 192 horas – mes = **13.766** mes – hombres aproximadamente.

Y, si en el proyecto trabajan tres hombres, el tiempo de desarrollo es:

Tiempo de desarrollo = **Et** / **Cantidad de hombres.**

Tiempo de desarrollo = **4** meses aproximadamente.

El tiempo a emplear para el desarrollo del sistema es aproximadamente 4 meses.

Paso #5: Salario.

El cálculo del salario mensual se realiza teniendo en cuenta que los desarrolladores de la herramienta pueden ser ingenieros recién graduados, entonces se toma como salario mensual: **$432.25**, que es el salario actual de los ingenieros recién graduados de la UCI.

Paso #6: Costo.

Anteriormente se definió que en el proyecto trabajarían 3 personas y que su salario sería de **$432.25 m.n.**, entonces el costo total sería:

Ct = **Salario mensual** x **Cantidad de hombres** x **Tiempo de desarrollo.**

Ct = **$5187.00 m.n.** ó **$207.48 c.u.c.**

El desarrollo de la aplicación planteada presenta un costo aproximado de $5187.00 m.n. ó $207.48 c.u.c.

3.6 Beneficios tangibles e intangibles.

- **Beneficios tangibles:**

Como es parte de los propósitos de la UCI insertar los productos desarrollados por su personal en el mercado internacional de aplicaciones de software, en el cual la competencia va, cada vez más, en ascenso con el paso del tiempo, es que es tal la importancia de garantizar que cada software que se libera, cuente con la calidad esperada, a pesar de esto no existen en la UCI medidores automatizados y abarcadores que respondan a este objetivo. La herramienta que se propone abarca funcionalidades que facilitan la realización de diferentes métodos de prueba que son escasamente utilizadas y que resultarían en beneficio para el desarrollo de los períodos de prueba de los proyectos.

- **Beneficios intangibles:**

Entre los principales planes económicos del país y en particular de la UCI se encuentra la exportación de software de alta calidad, por lo que la automatización del proceso de pruebas es vital para el desarrollo de sistemas con dicho propósito, lo cual resulta muy económico a la vez.

La herramienta propuesta facilita el proceso de efectuar pruebas al código de una aplicación de software, para determinar su complejidad y revelar las posibilidades de error que pueda contener, además de exponer su validez y efectividad. También revisa el cumplimiento de los estándares de codificación trazados.

Esta categoría de pruebas no se realiza en el Grupo de Calidad del centro GEySED de la Facultad 6 en la actualidad y la misma constituye una forma más de garantizar la calidad del software producido en dicho centro, de una manera sencilla y rápida al automatizarse las técnicas de caja blanca propuestas. Esto conlleva a un aumento de la capacidad organizativa en el proyecto durante el desarrollo del proceso de pruebas y al perfeccionamiento de dicho proceso.

3.7 Análisis de costos y beneficios.

El costo de un producto debe encontrarse justificado por los beneficios que el mismo reporta. Según lo expresado en el epígrafe anterior, entre los beneficios que aporta la herramienta propuesta se encuentran: un mejor aseguramiento de la calidad de los productos de software con destino a la exportación realizados en la UCI, al

automatizar parte del proceso de aplicación de tan complicadas técnicas de pruebas. Además, dicha herramienta, desarrollada por el estudiantado de la universidad, reporta el ahorro de dinero en el empleo de otros sistemas con el fin de hacer pruebas y se incluye que cuenta con características específicas para su utilización por parte del centro GEySED.

No obstante, según los beneficios intangibles mencionados, el producto es viable y justificado, pues reporta grandes beneficios al Grupo de Calidad aportando una gran ayuda a la informatización del proceso de pruebas, de gran importancia en la actualidad, debido a la necesidad de la realización de productos de muy alta calidad.

Conclusiones

En este capítulo se expusieron los diferentes componentes de la solución propuesta haciendo uso de disímiles diagramas con el fin de modelar los elementos vinculados a la aplicación. Fueron determinadas las clases del Análisis y las clases del Diseño, incluyendo sus relaciones en los Diagramas de Clases del Diseño.

Se ha logrado, de esta forma, modelar todos los procesos involucrados en el desarrollo de la herramienta propuesta y proporcionar una visión lo más completa posible de lo que esta trata. Se plantean, también, los principios del Diseño de la aplicación, la concepción del tratamiento de errores, de la ayuda y el sistema en general, y sus propiedades de extensibilidad.

Los diferentes diagramas incluidos en este capítulo son una guía de sencilla comprensión para los futuros desarrolladores, probadores y encargados del mantenimiento de la herramienta. Además se realizó una estimación del costo y el tiempo necesario para el desarrollo de la misma mediante un pequeño estudio de factibilidad.

El sistema propuesto ofrece un conjunto de beneficios para los probadores de los diferentes proyectos productivos que componen el centro GEySED de la Facultad 6, pues contribuye a la realización de las pruebas de Caja Blanca, convirtiéndolas en un proceso más fácil y rápido de llevar a cabo.

CONCLUSIONES

- Con el fin de realizar la modelación de la herramienta se analizaron algunas soluciones existentes a nivel internacional con propósitos similares.
- Se llevó a cabo un estudio de las características de varias técnicas de pruebas de Caja Blanca.
- La aplicación se diseñó según la pauta de la metodología RUP y se emplearon diagramas para la modelación de todas las fases del proyecto que se realizaron.
- Se acometió la modelación del Dominio y del sistema propuesto compendiado en el alcance del trabajo. Posteriormente se realizó el análisis de la aplicación para facilitar la comprensión y construcción del diseño propuesto, garantizando que la herramienta resultante sea fácil de entender y cumpla los estándares del Diseño.
- Se han alcanzado los objetivos definidos para este trabajo, modelándose un sistema de software que satisface las necesidades y requerimientos del Grupo de Calidad del centro GEySED de la Facultad 6.

RECOMENDACIONES

Tras finalizar la presentación del estudio realizado que culmina con el diseño de la herramienta software para la automatización del proceso de pruebas de Caja Blanca al departamento de Señales Digitales del centro GEySED, se brindan, a continuación, recomendaciones para la ampliación, modificación, mejora e implementación de esta aplicación y/o nuevas versiones de la misma:

- Llevar a cabo la realización de la herramienta con el propósito de dar solución a las necesidades existentes en el Grupo de Calidad del centro GEySED de la Facultad 6.
- Continuar el estudio del tema examinando las posibilidades de ampliar las funcionalidades de la aplicación, añadiéndole nuevas técnicas de pruebas de Caja Blanca y diferentes lenguajes de programación que puedan ser revisados así como nuevas funcionalidades en general.
- Incluir la herramienta en el proceso de pruebas a los productos realizados en el centro GEySED y estudiar la posibilidad de extenderlo a otros centros.
- Enriquecer la interpretación de los estándares de codificación y parámetros de código incluidos y añadir nuevos.
- Emplear, durante el proceso de implementación de la herramienta, módulos existentes, ya sea plugins o componentes, para la realización y facilitación de la tarea de interpretar el código y su procesamiento.

BIBLIOGRAFÍA Y REFERENCIAS BIBLIOGRÁFICAS

1. mgar.net. *Calidad.* [En línea] Mgar.net, 2011. [Citado el: 03 de 12 de 2010.] http://mgar.net/soc/isointro.htm.

2. **Copyright © 2001-2002 Estrucplan Consultora S.A. Argentina.** estrucplan on line. [En línea] Estrucplan Consultora S.A, 2011. [Citado el: 03 de 12 de 2010.] http://www.estrucplan.com.ar/Producciones/Entrega.asp?identrega=2624.

3. **© 2011 SlideShare Inc. All rights reserved.** slideshare. *Calidad Del Software.* [En línea] SlideShare Inc, 2010. [Citado el: 03 de 12 de 2010.] http://www.slideshare.net/lcahuich/calidad-del-software-presentation.

4. Aseguramiento de la calidad. *wiki.* [En línea] Wikipedia.org, 2010. [Citado el: 04 de 12 de 2010.] http://es.wikipedia.org/wiki/Calidad#Aseguramiento_de_la_Calidad.

5. **Mgar.net.** mgar.net . *Aseguramiento de la calidad.* [En línea] Mgar.net, 2010. [Citado el: 04 de 12 de 2010.] http: //mgar.net/soc/isointro.htm.

6. www.mitecnologico.com. *Aseguramiento de la calidad.* [En línea] Tecnológico, 2010. [Citado el: 04 de 12 de 2010.] http: //www.mitecnologico.com/Main/SistemaDeAseguramientoDeCalidad.

7. **Pressman, Roger S.** *Ingeniería del Software, Un enfoque práctico, 5ta Edición.* Madrid : McGraw - Hill, 2002.

8. [En línea] 2011. [Citado el: 04 de 02 de 2011.] https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Introducción%20a%20RUP.doc.

9. [En línea] 2011. [Citado el: 04 de 02 de 2011.] http://deigote.blogspot.com/2006/03/extreme-programming.html.

10. [En línea] 2011. [Citado el: 04 de 02 de 2011.] http://cfrela.en.eresmas.com/uml/umlanalisis.htm.

11. *UML en 24 horas.*

12. **Saavedra, J.** Patrones GRASP (Patrones de Software para la asignación General de Responsabilidad).Parte II. *Patrones de Software para la asignación General de.* [En

línea] 2007. [Citado el: 10 de 02 de 2011.] http://jorgesaavedra.wordpress.com/2007/05/08/patrones-grasp-patrones-de-software-para-laasignacion-.

13. *Conferencia 1. Introducción a la Ingeniería de Software.* **Colectivo de la Universidad de las Ciencias Informáticas.** 2005 - 2006.

14. **Copyright IBM Corp.** Rational Enterprise Edition. *Rational Enterprise Edition, Ayuda extendida.* 2006.

15. **Jacobson, I., Rumbaugh, J. y Booch, G.** *El Proceso Unificado de Desarrollo de Software, Volumen I y II.* La Habana : s.n., 2008.

16. **Acosta Rodríguez, Darianne y Barzaga Llano, Eglis.** *Propuesta de proceso de aseguramiento de la calidad de la distribución GNU/Linux Nova y adaptación de la metodología SXP.* La Habana : s.n., 2010.

17. **Darias Pérez, Darling.** *Análisis y Diseño de componentes para pruebas de Caja Blanca.* La Habana : s.n., 2007.

18. **Fuentes Guerra, Yurién Ricardo y Jordán Borjas, Ernesto.** *Implementación de una herramienta para viabilizar el proceso de pruebas de caja blanca.* La Habana : s.n., 2008.

19. [En línea] 2011. http://www.cafeconf.org/2007/slides/lisandro_perez_meyer_introduccion_%20a_qt4.pdf .

20. [En línea] 2010. http://gridtics.frm.utn.edu.ar/docs/Introduccion%20a%20la%20Calidad%20de%20Software%20Vazquez.pdf.

21. [En línea] 2011. http://personales.unican.es/ruizfr/is1/doc/lab/03/is1-p03-NegocioDominio.pdf.

22. [En línea] 2011. http://es.scribd.com/doc/18066982/MODELO-DE-OBJETOS.

23. [En línea] 2011. http://www.15dejuniomnr.com.ar/blog/apunteca/Ciclo%20Basico/Informatica%20II/Informatica%20II%20(ECA)/02.%20Modelo%20de%20objetos.ppt.

24. **Nokia Corporation and/or its subsidiaries.** Qt. *Qt Creator IDE and tools .* [En línea] Nokia, 2008 - 2011. [Citado el: 01 de 02 de 2011.] http://qt.nokia.com/products/developer-tools/.

25. CodeBox. [En línea] 2011. [Citado el: 03 de 02 de 2011.] http://www.codebox.es/glosario.

26. [En línea] 2007. http://migueljaque.com/index.php/tecnicas/tecnicasmodnegocio/37-modelado_negocio/46-modelo-de-dominio?tmpl=component&print=1&page=.

27. **UCI.** EcuRed. *Flujo de Trabajo Modelo del Negocio.* [En línea] E.V.A. UCI, 2011. [Citado el: 02 de 22 de 2011.] http://www.ecured.cu/index.php/Flujo_de_Trabajo_Modelo_del_Negocio#Diagrama_de_Clases_del_Modelo_de_Objetos.

28. **Visual Paradigm.** Visual Paradigm. *UML, BPMN and Database Tool for Software Development.* [En línea] Visual Paradigm, 2011. [Citado el: 12 de 02 de 2011.] http://www.visual-paradigm.com/.

29. **Rojas, Johanna y Barrios, Emilio.** Métodos de prueba de caja blanca. *Métodos de prueba de caja blanca.* [En línea] Grupo ARQUISOFT, 2007. [Citado el: 07 de 02 de 2011.] http://gemini.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node26.html.

30. Clase 8: Modelos de objetos e invariantes. [En línea] http://mit.ocw.universia.net/6.170/6.170/f01/pdf/lecture-08.pdf.