Universidad de las Ciencias Informáticas

Facultad 7

ANÁLISIS Y DISEÑO DE COMPONENTES PARA PRUEBAS DE CAJA BLANCA

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

Autora: Darling Darias Pérez

Tutora: Ing. Lourdes Escalona Peral

Ciudad de la Habana, julio de 2008

¨Año 50 de la Revolución¨

DECLARACIÓN DE AUTORÍA

Declaro que soy la única autora de este trabajo y autorizo a la Facultad 7 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los 8 días del mes de Julio del año 2008.

_____ _____

Autora Darling Darias Pérez **Tutor: Ing. Lourdes Escalona Peral**

DATOS DE CONTACTO:

Ing. Lourdes Escalona Peral:

Profesor graduado de Ingeniero Informático en el año 2004 en la Universidad de Holguín. Ha impartido las asignaturas Ingeniería de Software 1 y 2 y Seminario de Tesis. Fue líder del proyecto Atención Primaria de Salud durante dos años consecutivos y asesora de calidad de la facultad durante un año.

Ha tutorado tesis de perfil de análisis y diseño, implementación de sistema y atendiendo al área de calidad, Pruebas de software.

Posee la categoría docente de Instructor y cursa la maestría Gestión de Proyectos en la UCI. Se ha desempeñado como Jefe de Dpto. de la Especialidad de la Facultad 7, en los últimos 2 años.

Empresa: UCI Dirección: Carretera a San Antonio Km. 2 1/2 Reparto Torrens, Infraestructura productiva de la UCI, Ciudad Habana.

Teléfono: 835-8131. e-mail: lescalonap@uci.cu.

Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber.

Albert Einstein

AGRADECIMIENTOS

Son muchas las personas que de una forma u otra han propiciado el resultado de este trabajo y a las que deseo, sin perder la oportunidad, agradecer por brindarme energía positiva todo el tiempo, además del apoyo incondicional.

A Yurien, por estar dispuesto cada vez que lo necesitaba y aportarme sus granitos de conocimiento.

A Yenisel por aclarar mis dudas y sugerirme buenas ideas.

A Hugo por ayudarme a traducir contenidos que necesitaba para el desarrollo del trabajo.

Agradezco además, a esta Revolución y al siempre Comandante en Jefe, Fidel Castro Ruz, por darme la oportunidad de ser una de las protagonistas de este, su gran sueño.

A todos mis profesores, por trasmitirme los conocimientos y valores que permitieron mi formación como profesional.

A mi tutora Lourdes Escalona Peral por brindarme sus conocimientos, ser atenta y confiar en mí.

A la profesora Pura y a Duniesky por dedicar tiempo en la revisión de la documentación.

A todas mis amistades por su compañía durante todos estos años, por compartir los buenos y malos momentos.

Y de manera muy especial a mi familia por su dedicación, confianza y amor. Que son los principales intérpretes de mi vida, esforzándose por convertirme en un ser cada día mejor, por guiarme siempre por el camino correcto, por alentarme, darme seguridad ante todos los acontecimientos de mi vida y estar a mi lado en cada momento.

En fin, a todos los que han favorecido, al menos con un gesto o una palabra, muchísimas GRACIAS.

DEDICATORIA

A mis padres Lupe y Manuel, por el infinito amor y respeto que les tengo, por haberme dado la oportunidad de existir, además de regalarme en cada instante todo su cariño.

A ti mami por haberme educado con devoción y esmero, por ser ejemplo íntegro para mi formación como mujer, por cederme consejos y estar a mi lado en circunstancias difíciles en las que me alentabas. Por inculcarme valores, principios y la esencia del ser y más que todo lo anterior, por concederme todo tu amor.

A mi papi, por entregarme su ternura, por dedicarme tiempo y hacerme sentir la niña linda de sus ojos, por alentarme en cuanto a mi superación, por nunca dudar de la educación que recibía, por tenerme confianza y consentirme, por quererme excesivamente. Sin ustedes, que son la fuente de mi inspiración, solo fuera una tenue sombra errando sin destino.

A mi hermanita Angélica por ser tan ocurrente y llevarme en su corazoncito, a Víctor Manuel mi hermano, a mis abuelas, especialmente Angelina, por su ternura, sabiduría y cordialidad, porque es mi otra madre, porque la quiero mucho.

A Juan por ser como un padre para mí. Quizás no nos unan lazos de sangre, pero nos une el cariño y el afecto.

A mis tíos y primos por su apoyo en muchos momentos de mi vida. Para ti tía Alina que me considera u tu hija.

A Hugo, mi pareja, por estar en las buenas y las malas, por ser mi apoyo y compañía durante muchos años, por alentarme y estar pendiente de mis obligaciones, por quererme y cuidarme.

A todas mis amistades de la universidad, que han estado junto a mí en muchas circunstancias, el recuerdo de ustedes permanecerá en mi alma. Especialmente a Naya, Diana, Dagmay, Yuliet, Mary, Yunesti, Yoisell, Franck, Arian, Karelys por permitirme ser parte de sus vidas y auxiliarme cuando lo necesito. A mis compañeros Yurien Ricardo y Ernesto, que aunque compartimos tesis diferentes siempre estuvieron pendientes y me ayudaron en varias ocasiones.

A todas las personas que han formado parte de mi vida en este último año.

A todos por su cariño y confianza ¡GRACIAS!

RESUMEN

El proceso de pruebas al software es unos de los aspectos fundamentales para medir el estado de calidad de un sistema informático e introducirlo satisfactoriamente en el mercado mundial. Sin embargo, en las revisiones de software del grupo de calidad de la facultad 7, existen disímiles dificultades, siendo una de las principales la carencia de mecanismos para hacer pruebas de Caja Blanca de forma automatizada.

El objetivo del presente trabajo de diploma es diseñar una herramienta informática para viabilizar los procesos de pruebas de Caja Blanca que se le realicen al software, favoreciendo el proceso de revisión de los mismos.

La herramienta está concebida bajo la metodología RUP, por ser la más utilizada a nivel mundial; basada a su vez, en UML (Lenguaje Unificado de Modelado). Para documentar el software, se utilizó la herramienta Rational Rose Enterprise Edition 2003.

El uso de este sistema para pruebas, facilitará el trabajo de los probadores. Además posibilitará el empleo de otro método de pruebas, poniendo al software en todas las situaciones posibles, lo que garantizará minimizar los errores y que cumpla con las funcionalidades requeridas por el cliente.

TABLA DE CONTENIDOS

INTRODUCCIÓN

El avance precipitado de la ciencia y las tecnologías de la información, ha traído como consecuencia que las empresas informáticas existentes, desafíen cada día un reto para brindar una respuesta rápida, eficaz y con calidad, a los clientes que cada vez se vuelven más exigentes no sólo en cuanto al precio, sino también en la confiabilidad y seguridad que deben poseer los productos de software.

A partir del surgimiento de los grandes sistemas informáticos en el mundo, se fue incluyendo al mismo tiempo y por necesidad, el proceso de pruebas, usado sistemáticamente junto al desarrollo de cada aplicación.

Las pruebas de software son un conjunto de actividades que se llevan a cabo metódicamente, pueden planificarse por adelantado y ejecutarse una vez construido el código para la revisión final de las especificaciones, del diseño y de la codificación del software. Las mismas, vistas desde el marco de un proceso de desarrollo de software, son los diferentes procesos que se ejecutan durante la realización de un producto, con el objetivo de asegurar efectividad, adecuada funcionalidad y validez del mismo.

Al contrario de lo que muchas personas creen, el proceso de pruebas, según algunos cálculos internacionales representa más de la mitad del coste de un programa, al requerir un tiempo similar al de la programación. Obviamente esto conduce a un alto costo económico cuando no involucra vidas humanas, puesto que en este último caso el costo suele superar el 80%, según algunas informaciones, siendo esta etapa más costosa que el propio desarrollo y diseño de los distintos programas que conforman el sistema. Es por ello la necesidad de una verdadera metodología, la cual exige herramientas y conocimientos destinados a dicha tarea y la insistencia de comenzar a realizar las pruebas desde la misma etapa de análisis y levantamiento de requerimientos, ya que desde un principio se puede incurrir en erróneas interpretaciones de las reglas del negocio, lo que finalmente tendrá como consecuencia discrepancia entre lo que el cliente quiere y lo que se ha desarrollado.

Realizar las pruebas inequívocas de un programa implicaría ponerlo en todas las situaciones posibles, de esta forma se aseguraría que el mismo se encuentre libre de errores. Para ello se considera importante la búsqueda de formas y métodos que garanticen dicho resultado. En la actualidad, un intento de mejoras, lo constituye la automatización del proceso de pruebas. Entre sus dificultades se encuentran la falta de herramientas, debido fundamentalmente a su elevado precio o a que las existentes no se ajusten al propósito y/o entorno para el que se necesitan. A esto se le une la deficiencia de compatibilidad e interoperabilidad entre sistemas, la ausencia de un proceso básico de

pruebas y el desconocimiento acerca de qué es lo que se debe probar, sin menos preciar, la privación del uso de las herramientas de prueba que ya se poseen, bien por su dificultad de uso o por falta de tiempo para aprender a manejarla, por falta de soporte técnico, obsolescencia, por la formación inadecuada en el uso de las mismas, entre otros aspectos.

Cuba, desafortunadamente no ha trascendido en la producción de software acreditado por sistemas de calidad, al igual que muchos países. Aunque en los últimos años ha ido aumentando el interés de usar el capital intelectual como principal fuente económica; lo que ha propiciado el surgimiento de diferentes estrategias con el fin de elevar la producción de software cubano, solo que en muchas de las empresas se trabaja en una fase artesanal donde aún no se definen procedimientos para medir los estándares de calidad, provocando que no se alcancen los avances que se aspiran.

"La Universidad de las Ciencias Informáticas (UCI), institución surgida al calor de la Batalla de Ideas, tiene incluido en sus principales objetivos, convertirse en una Industria de Software de alto prestigio a nivel nacional e internacional. Actualmente cuenta con disímiles proyectos productivos, que en su mayoría aportan considerables ingresos a la economía del país. De forma general, se contribuye al cumplimiento de una de las líneas directrices de la Batalla de Ideas, la informatización de la sociedad." [1]

Para confirmar que los proyectos entreguen productos de alta calidad, como resultado de un proceso de desarrollo de software, la Universidad se trazó como estrategia la formación de un Grupo de Calidad en cada facultad, asesorados por el Laboratorio Central de Calidad de la UCI. Estos grupos de apoyo a la calidad desempeñan diversas tareas relacionadas con este tema, entre ellas se encuentran las pruebas a los productos de software terminados, utilizando una metodología o un plan de pruebas a seguir en el momento que son efectuadas, además algunas técnicas de pruebas de Caja Negra. Otra de las actividades asignadas es controlar, de cierta forma, todo el proceso de calidad dentro de la facultad correspondiente.

En el grupo de la Facultad 7 el proceso de control de la calidad aún no cuenta con la profundidad y eficacia que se requiere, debido en cierta medida por falta de experiencia. Actualmente solo se ha desarrollado el proceso de pruebas específicamente utilizando métodos de Caja Negra. Los métodos de Caja Blanca solo se han realizado de forma manual, incluso, muchas veces no se cometen, solo el anteriormente mencionado, dejándose de revisar el código de los software y así desconocer totalmente qué tan óptimo este pueda ser, sin tenerse en cuenta que ambos métodos son complementarios. Existe un trabajo de diploma que propone un procedimiento para llevar acabo el método de prueba de

Caja Blanca, sin embargo no es aplicado por los integrantes del grupo de Calidad. Además, no se cuenta con una herramienta automatizada con determinadas funcionalidades, que sea adaptable a la situación interna o a problemas específicos y que realice pruebas utilizando el conocimiento del funcionamiento interno del código.

Luego del análisis de la situación existente en el grupo de Calidad de la Facultad 7, se identificó el siguiente **problema científico**: ¿Cómo facilitar el proceso de revisiones de software de los proyectos productivos de la Facultad 7?

Este problema se enmarca en el **objeto de estudio**: Proceso de revisiones de Software.

Siendo abarcado como el **campo de acción**: Procesos de pruebas de Caja Blanca.

El **Objetivo general** es:

- Realizar el análisis y diseño de una herramienta informática para viabilizar los procesos de pruebas de Caja Blanca que se realizan a los software en los proyectos productivos de la Facultad 7.

Las **Tareas de la investigación** concebidas:

- Determinar las características de las pruebas de Caja Blanca usando la técnica del camino básico para las revisiones de software en la Facultad 7.
- Definir posibles estándares de codificación en los lenguajes más utilizados en la facultad.
- Determinar posibles parámetros de eficiencia en el código (cantidad de operandos diferentes, longitud estimada, etc.)
- Definir posibles parámetros del código fuente en funciones o archivos con código.
- Realizar estado del arte de las pruebas automatizadas de Caja Blanca.
- Seleccionar la metodología y herramientas a utilizar para el desarrollo de la aplicación para pruebas de Caja Blanca.
- Modelar los conceptos fundamentales del dominio.
- Realizar el análisis y diseño de una aplicación para la ejecución de pruebas de Caja Blanca automatizadas.

El presente trabajo consta de tres Capítulos distribuidos de la siguiente forma:

En el Capitulo 1 **Fundamentación Teórica**, se hace alusión de los conceptos fundamentales afines con la temática, también se manifiesta un estudio detallado sobre las características de las pruebas de Caja Blanca usando la técnica del Camino Básico, de los estándares de codificación y los parámetros del código fuente que se tendrán en cuenta para insertar en la herramienta. Se incluye la situación de los sistemas automatizados utilizados en la actualidad a nivel mundial, así como la situación de las tecnologías actuales, lo que permite decidir sobre qué funcionalidades se conciben para el diseño e implementación del sistema.

En el Capitulo 2 **Características del Sistema** se abordan los aspectos relacionados con el objeto de estudio, haciendo énfasis en los procesos involucrados en el campo de acción, un análisis de cómo se ejecutan los diferentes procesos, la información que se maneja, así como el objeto de automatización. También se llevan a cabo los flujos de trabajo del Modelado del Dominio y Especificación de los Requisitos de Software. En el primero se exponen los principales artefactos o necesidades de los clientes resultantes del Modelo del dominio que finalmente conlleva a que se realice la propuesta del sistema. En el segundo se muestran los requisitos funcionales y no funcionales, además de la descripción de los actores y casos de uso resultantes del flujo de trabajo de Requisitos.

En el Capítulo 3 **Análisis y Diseño del sistema** se hace referencia al Modelo de Análisis, al Modelo de Diseño que incluye, los diagramas de clases de diseño, diagramas de interacción y la descripción de las clases de diseño. Además de una breve investigación analizando qué tan beneficioso resultaría la herramienta a desarrollar en vez de comprar varias aplicaciones que cumplan con las características propuestas.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se abordan los conceptos fundamentales afines con la fundamentación teórica, las características principales de las pruebas de Caja Blanca, así como sus diferentes técnicas. Especificándose, la del camino básico, además del estudio de los estándares y parámetros de codificación. Se hace referencia al estado del arte de los sistemas automatizados existentes para la realización de pruebas de software, utilizando el método pruebas de Caja Blanca, detallándose los antecedentes prácticos a nivel internacional y nacional. Se realiza un análisis descriptivo de las tendencias, tecnologías y metodologías más usadas en la actualidad y se concluye con la selección de las herramientas a utilizar durante el desarrollo de la aplicación.

1.1 Conceptos Fundamentales a tener en cuenta.

Pruebas de Caja Blanca: También suelen ser llamadas estructurales o de cobertura lógica. En ellas se pretende investigar sobre la estructura interna del código, exceptuando detalles referidos a datos de entrada o salida, para probar la lógica del programa desde el punto de vista algorítmico. Realizan un seguimiento del código fuente según se va ejecutando los casos de prueba, determinándose de manera concreta las instrucciones, bloques, etc. que han sido ejecutados por los casos de prueba.

Complejidad: Es proporcional al número de errores en un segmento de código. *"Entre más complejo, más susceptible a errores."* [2] Se relaciona con el esfuerzo requerido para probar. *"Entre más complejo, mayor atención para probar."* [3]

Prueba de Complejidad Ciclomática: Propuesta por Tom McCabe en 1976 y también llamada Número Ciclomático de McCabe. Es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. *"Se basa en la representación gráfica del flujo de control del programa. De dicho análisis se desprende una medida cuantitativa de la dificultad de prueba y una indicación de la fiabilidad final. Determina la estabilidad y riesgo inherente de un programa."* [4]

Complejidad Ciclomática: Es la "*medida de la complejidad lógica de un módulo "G" y el esfuerzo mínimo necesario para calificarlo. Es el número de rutas lineales independientes de un módulo "G", por lo tanto es el número mínimo de rutas que deben probarse.*" [5]

Camino: "*Secuencia de todas las instrucciones de un programa de principio a fin".* [6]

Camino independiente: "*Es cualquier camino del programa que incluye nuevas instrucciones de un proceso o una nueva condición".* [7]

Cobertura de caminos: "*Comprueba el número de caminos linealmente independientes que se han ejecutado en el grafo de flujo de la unidad que se está probando".* [8] El número de caminos linealmente independientes coincide con la complejidad ciclomática de McCabe.

Camino linealmente independiente de otros: Introduce por lo menos un nuevo conjunto de sentencias de proceso o una nueva condición.

Camino Básico: Es una técnica de prueba de caja blanca que permite obtener una medida de complejidad lógica para generar un conjunto básico de caminos que se ejecutan por lo menos una vez durante la ejecución del programa.

Matriz: Es un conjunto de elementos de cualquier naturaleza aunque, en general, "*suelen ser números ordenados en filas y columnas, dispuestos en forma rectangular."* [9]

Estándar: "*Es la norma técnica que se puede utilizar como punto de comparación para evaluar la calidad del servicio".* [10] O para alertar sobre los problemas en el mismo.

Métrica: "*Medida estadística que se aplica a todos los aspectos de calidad de software, los cuales deben ser medidos desde diferentes puntos de vista."* [11]

Parámetros de codificación: Es una propiedad o características de las métricas, u orden o valor que se le pasa a la función y esta puede variar su comportamiento.

1.2 Características de las pruebas de Caja Blanca.

En las pruebas de Caja Blanca, se pretende indagar sobre la estructura interna del código, omitiendo detalles referidos a datos de entrada o salida. Su objetivo principal es probar la lógica del programa desde el punto de vista algorítmico.

La prueba de Caja Blanca se basa en el diseño de Casos de Prueba que usa la estructura de control del diseño procedimental para derivarlos. Mediante la prueba de la Caja Blanca el ingeniero de software puede obtener Casos de Prueba que: [12]

- Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.

- Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
- Ejecuten todos los bucles en sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

Es por ello que se considera a la prueba de Caja Blanca como uno de los tipos de pruebas más importantes que se le aplican a los software, lográndose como resultado que disminuya en un gran por ciento el número de errores existentes en los sistemas y por ende, que exista una mayor calidad y confiabilidad en la codificación.

1.2.1 Tipos de pruebas de Caja Blanca.

De Estructura de Datos Locales: Se centra en el estudio de las variables del programa. Busca que toda variable esté declarada y que no existan con el mismo nombre, ni declaradas local y globalmente, que haya referencias a todas las variables y para cada variable, analiza su comportamiento en comparaciones.

De Cobertura Lógica: [13]

- De Cobertura de Sentencias: Comprueba que todas las sentencias se ejecuten al menos una vez.
- De Cobertura de Decisión: Ejecuta casos de prueba de modo que cada decisión se pruebe al menos una vez a Verdadero (True) y otra a Falso (False).
- De Cobertura de Condición: Ejecuta un caso de prueba a True y otro a False por cada condición, teniendo en cuenta que una decisión puede estar formada por varias condiciones.
- De Cobertura de Condición/Decisión: Se realizan las pruebas de cobertura de condición y las de decisión a la vez.
- De Condición Múltiple: Cada decisión multicondición se traduce a una condición simple, aplicando posteriormente la cobertura de decisión.
- De Cobertura de Caminos: Se escriben casos de prueba suficientes para que se ejecuten todos los caminos de un programa. Entendiéndose camino como una secuencia de sentencias encadenadas desde la entrada del programa hasta su salida.

Este último criterio es el que se va a considerar y a continuación se abordará sobre el mismo.

Cobertura de Caminos.

La aplicación de este criterio de cobertura asegura que los casos de prueba diseñados permitan que todas las sentencias del programa sean ejecutadas al menos una vez y que las condiciones sean probadas tanto para su valor verdadero como falso. Se dice por varios investigadores que es el criterio de cobertura más elevado.

Una de las técnicas empleadas para aplicar este criterio de cobertura es la prueba del Camino Básico. La misma se basa en obtener una medida de la complejidad del diseño procedimental de un programa (o de la lógica del programa). Cuando se refiere a esta medida, se trata de la complejidad ciclomática de McCabe, y representa un límite superior para el número de casos de prueba que se deben realizar para asegurar que se ejecuta cada camino del programa. Los pasos a seguir para aplicar esta técnica son:

- Representar el programa en un grafo de flujo.
- Calcular la complejidad ciclomática.
- Determinar el conjunto básico de caminos independientes.
- Derivar los casos de prueba que fuerzan la ejecución de cada camino.
- A continuación, se detallan cada uno de estos pasos.

Representación de un grafo de flujo.

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

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

Así, cada construcción lógica de un programa tiene una representación. La Figura 1 muestra un grafo de flujo del diagrama de módulos correspondiente. Nótese cómo la estructura principal corresponde a un while y dentro del bucle se encuentran anidados dos constructores if. [14]

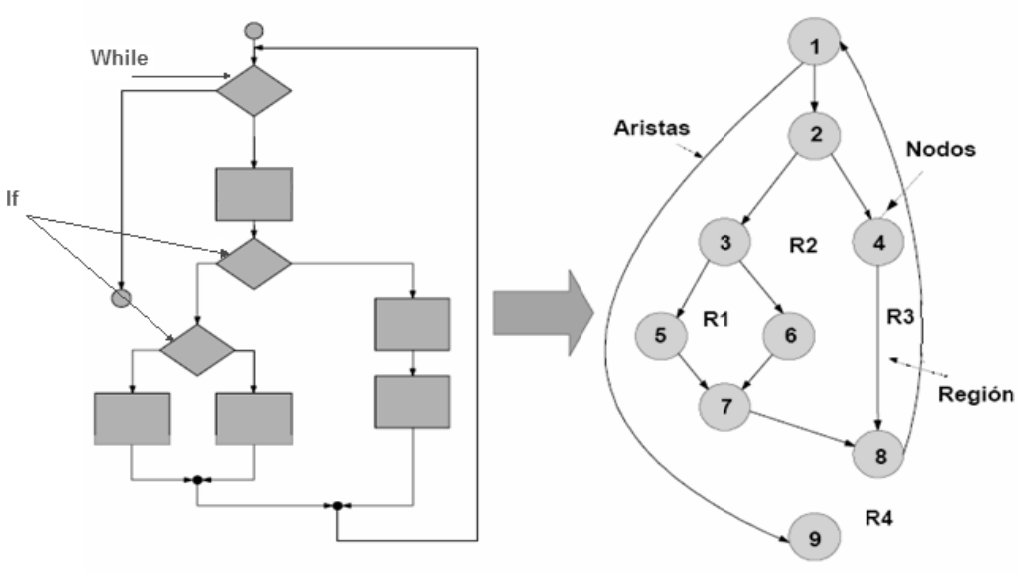


Figura 1: Ejemplo de grafo de flujo correspondiente a un diagrama de módulos.

Calcular la complejidad ciclomática.

La complejidad ciclomática es una métrica del software que proporciona una medida cuantitativa de la complejidad lógica de un programa. Se basa en la representación gráfica del flujo de control del programa, de dicho análisis se desprende una medida cuantitativa de la dificultad de prueba y una indicación de la fiabilidad final. Cuando se utiliza en el contexto del método de prueba del camino básico, el valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y propicia el límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez.

Es una de las métricas de software más ampliamente aceptada, ya que ha sido creada para ser independiente del lenguaje. Se han realizado investigaciones y ensayos, en los cuales se ha medido un gran número de programas, a modo de establecer rangos de complejidad que ayuden al ingeniero de software a determinar la estabilidad y riesgo inherente de un programa. La medida resultante puede ser utilizada en el desarrollo, mantenimiento y reingeniería para estimar el riesgo, costo y estabilidad.

Estos estudios indicaron la existencia de distintas relaciones entre la métrica de McCabe y el número de errores existentes en el código fuente, así como el tiempo requerido para encontrar y corregir esos errores*. "Se suele comparar la complejidad ciclomática obtenida contra un conjunto de valores límites como se observa en la Tabla 1."* [15]

Tabla 1. Complejidad ciclomática vs. Evaluación de riesgo.

Complejidad Ciclomática	Evaluación del Riesgo
1-10	Programa Simple, sin mucho riesgo.
11-20	Más complejo, riesgo moderado.
21-50	Complejo, Programa de alto riesgo.
50	Programa no testeable, Muy alto riesgo.

El ámbito más común de utilización para probar módulos unitarios es la prueba estructural (caja blanca). McCabe también expone que se puede utilizar la complejidad ciclomática para dar una indicación cuantitativa del tamaño máximo de un módulo. A partir del análisis de muchos proyectos encontró que un valor 10 es un límite superior práctico para el tamaño de un módulo.

Cuando la complejidad supera dicho valor se hace muy difícil probarlo, entenderlo y modificarlo. La limitación deliberada de la complejidad en todas las fases del desarrollo ayuda a evitar los problemas asociados a proyectos de alta complejidad. El límite propuesto por McCabe sin embargo es fuente de polémicas. Algunas organizaciones han utilizado el valor 15 con bastante éxito. No obstante, un valor superior a 10 debería ser reservado para proyectos que tengan ventajas operacionales con respecto a proyectos típicos.

La complejidad ciclomática puede ser aplicada en varias áreas incluyendo: [16]

- Análisis de Riesgo en desarrollo de código: Mientras el código está en desarrollo, su complejidad puede ser medida para estimar el riesgo inherente.
- Análisis de riesgo de cambio durante la fase de mantenimiento: La complejidad del código tiende a incrementarse a medida que es mantenido durante el tiempo. Midiendo la complejidad antes y después de un cambio propuesto, puede ayudar a decidir cómo minimizar el riesgo del cambio.
- Planificación de Pruebas: El análisis matemático ha demostrado que la complejidad ciclomática indica el número exacto de casos de prueba necesarios para probar cada punto de decisión en un programa.

- Reingeniería: Provee conocimiento de la estructura del código operacional de un sistema. El riesgo involucrado en la reingeniería de una pieza de código está relacionado con su complejidad.

¿Por qué es tan importante y aún se cita?

Permite apreciar la calidad del diseño software de una manera rápida y con independencia del tamaño de la aplicación. Es una medida esencial cuando se necesita "tomar la temperatura" de un diseño software de un tamaño considerable (más si se está realizando una auditoría externa y no se conocía de antes la aplicación), y que se puede obtener fácilmente de manera automatizada. También se ha utilizado para planificar proyectos de mejora de grandes productos software, para priorizar las partes del diseño a mejorar.

Por otro lado, en muchas ocasiones, es base para calcular el valor de las mejoras del diseño, o el valor que aporta introducir un patrón o buena práctica. Además, da el número de casos de prueba unitarios básicos para obtener una cobertura del 100% y un aproximado al grado de comprensibilidad.

¿Cómo sería su cálculo?

"Existen varias formas de calcular la complejidad ciclomática de un programa a partir de un grafo de flujo: [17]

- *CC = arcos – nodos + 2*
- *CC = número de nodos de decisión + 1*
- *CC = número de regiones"*

Esta complejidad ciclomática determina el número de casos de prueba que deben ejecutarse para garantizar que todas las sentencias de un programa se han ejecutado al menos una vez, y que cada condición se habrá ejecutado en sus vertientes verdadera y falsa.

A continuación se expondrá cómo se identifican estos caminos.

Determinar el conjunto básico de caminos independientes.

Un camino independiente es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una condición, respecto a los caminos existentes. En términos del diagrama de flujo, un camino independiente está constituido por lo menos por una arista que no haya

sido recorrida anteriormente a la definición del camino. En la identificación de los distintos caminos de un programa para probar se debe tener en cuenta que cada nuevo camino debe tener el mínimo número de sentencias nuevas o condiciones nuevas respecto a los que ya existen. De esta manera se intenta que el proceso de depuración sea más sencillo.

El conjunto de caminos independientes de un grafo no es único. No obstante, a continuación, se muestran algunas heurísticas para identificar dichos caminos: [18]

- Elegir un camino principal que represente una función válida que no sea un tratamiento de error. Debe intentar elegirse el camino que atraviese el máximo número de decisiones en el grafo.
- Identificar el segundo camino mediante la localización de la primera decisión en el camino de la línea básica alternando su resultado mientras se mantiene el máximo de número de decisiones originales del camino inicial.
- Identificar un tercer camino, colocando la primera decisión en su valor original a la vez que se altera la segunda decisión del camino básico, mientras se intenta mantener el resto de decisiones originales.
- Continuar el proceso hasta haber conseguido tratar todas las decisiones, intentando mantener como en su origen el resto de ellas.

Este método permite obtener de la complejidad ciclomática (CC), caminos independientes cubriendo el criterio de cobertura de decisión y sentencia.

Así por ejemplo, para la el grafo de la Figura 1 los cuatro posibles caminos independientes generados serían:

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

Derivar los casos de pruebas.

El último paso es construir los casos de prueba que fuerzan la ejecución de cada camino. Una forma de representar el conjunto de casos de prueba sería llenar la "*Tabla 1.2."* [19] que se muestra a continuación.

Tabla 1.2. Posible representación de casos de prueba para pruebas estructurales.

Número del Camino	Caso de Prueba	Resultado Esperado

1.3 Estándares de codificación.

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez.

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. La mantenibilidad del mismo es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores o mejorar el rendimiento. Aunque la legibilidad y la mantenibilidad son el resultado de muchos factores, una faceta del desarrollo de software en la que todos los programadores influyen especialmente es en la técnica de codificación. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas. [20]

Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento. Además, si se aplica de forma continuada un estándar de codificación bien definido, y se utilizan técnicas de programación apropiadas y posteriormente, se efectúan revisiones del código de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

Aunque el propósito principal para llevar a cabo revisiones del código a lo largo de todo el desarrollo es localizar defectos en el mismo, las revisiones también pueden afianzar los estándares de codificación de manera uniforme. La adopción de un estándar de codificación sólo es viable si se sigue desde el principio hasta el final del proyecto de software. No es práctico, ni prudente, imponer un estándar de codificación una vez iniciado el trabajo.

1.3.1 Ventajas de los estándares de codificación.

El uso de estos estándares tiene innumerables mejorías, entre ellas:

- Asegurar la legibilidad del código entre distintos programadores, facilitando el debugueo [1]del mismo.
- Proveer una guía para el encargado de mantenimiento/actualización del sistema, con código claro y bien documentado.
- Facilitar la portabilidad entre plataformas y aplicaciones.

Es por esto que la codificación de los módulos de los sistemas a desarrollar debe cumplir ciertos requisitos. Se propone una serie de aspectos a tener en cuenta para comprobar los estándares según los criterios de programadores, aunque ellos son los que definen cuáles estandarizar y el estilo a aplicar, además de tener presente las características propias del lenguaje de programación, los recursos del mismo que se utilizarán y el tipo de programa que se debe implementar.

1.3.2 Definición de algunos estándares para comprobar código fuente del lenguaje C#.

Identificadores.

Al igual que en cualquier lenguaje de programación, en C# un identificador no es más que, como su propio nombre indica, un nombre con el que se identifica algún elemento del código, ya sea una clase, una variable, un método, etc. Típicamente el nombre de un identificador será una secuencia de cualquier número de caracteres alfanuméricos (incluidas vocales acentuadas y eñes) tales que el primero de ellos no sea un número. Por ejemplo, identificadores válidos serían: Arriba, caña, C3P0, áëÎò; pero no lo serían 3com, 127, entre otros.

Sin embargo, y aunque por motivos de legibilidad del código no se recomienda, C# también permite incluir dentro de un identificador caracteres especiales imprimibles tales como símbolos de diéresis,

[1] Debugueo: término informático que significa compilar código.

subrayados, etc. siempre y cuando estos no tengan un significado especial dentro del lenguaje. Por ejemplo, también serían identificadores válidos, _barco_, c¨k y A·B; pero no C# (# indica inicio de directiva de preprocesado) o a!b (! indica operación lógica “not”)

El lenguaje de C# da la posibilidad de poder escribir identificadores que incluyan caracteres Unicode que no se puedan imprimir usando el teclado de la máquina del programador o que no sean directamente válidos debido a que tienen un significado especial en el lenguaje. Para ello, lo que permite es escribir caracteres usando secuencias de escape, que no son más que secuencias de caracteres con una sintaxis específica.

Este tipo de estándar tiene estilos definidos a nivel mundial como son: lowerCamelCase y el UperCamelCase. El identificador del primero comienza con minúscula y para el segundo, el identificador comienza con mayúscula. Para ambos estilos cada palabra interna en identificadores compuestos comienza con mayúsculas y no se colocan caracteres de separación entre ellas. Cada identificador, debe ser lo suficientemente descriptivo. Es necesario evitar escribir abreviaturas, pues suelen confundir y se sugiere que los sinónimos deben de ser cortos para que el identificador sea sencillo y a la vez expresivo.

Indentación.

A pesar de que el lenguaje de programación C# ofrece una amplia libertad en cuanto a la forma en que pueden escribirse los programas, es necesario establecer algunas normas que ayuden a hacer los programas más legibles y por lo tanto más fáciles de comprender y modificar. Aunque no es posible ofrecer una lista exhaustiva de todas las decisiones de formato que un programador debe enfrentar al escribir sus programas, se intentará establecer algunas pautas.

Las normas que a continuación se expondrán no son absolutas y reflejan únicamente las preferencias del autor. Existen muchas otras, quizá hasta mejores. Lo importante es adoptar un estándar y respetarlo al máximo posible. De esta forma se escribirán programas correctos, no sólo en cuanto a su funcionalidad, sino también en cuanto a su apariencia.

¿Cómo indentar el código?

Los programas normalmente tienen una estructura jerárquica (por niveles) y es importante destacarla al escribirlos. Esto se logra mediante indentación, es decir, se dejan algunos espacios entre el código

que está en un nivel y el del siguiente nivel. Pero ¿cuándo indentar? Una regla sencilla es que cada vez que se abran corchetes se estará creando un nuevo nivel y por lo tanto es necesario indentar.

La utilización de este estilo propicia una mejor legibilidad y comprensión para el programador, obedeciendo el propio estilo de cada persona, del lenguaje y el tipo de programa que se desarrolle. Es sin dudas, la indentación, una buena práctica de programación, que radica en comenzar a escribir cada línea de código a desiguales trayectos desde el borde izquierdo del área de texto del editor.

Llaves.

Algunos programadores prefieren delimitar el cuerpo de los bloques de código en los lenguajes que contienen este tipo de estructuras ubicando la llave de apertura inmediatamente detrás de la línea cabecera del bloque, mientras que otros proponen ubicarlas de forma solitaria en la línea siguiente a la línea cabecera. Algunos eligen usar siempre las llaves mientras otros optan aprovechar las ventajas del lenguaje obviándolas en los casos que el cuerpo del bloque contenga solo una sentencia. Aún no se descubre una definición puntual de qué sería una buena práctica de programación en el uso de llaves debido a que cada método tiene ventajas y desventajas, aunque con este estilo se garantiza la legibilidad del código.

Líneas y espacios en blanco.

El uso de líneas en blanco incrementa la longitud en líneas de código del programa y el uso de espacios en blanco incrementa la longitud de cada línea de código del programa. Esto, sin embargo, mejora la legibilidad del código utilizándose líneas en blanco para separar segmentos de código que se deseen despejar. Así mismo sucede con los espacios en blanco cuando se utilizan para separar elementos dentro de las sentencias de código.

Comentarios.

Con el objetivo de hacer los programas más legibles, es importante la incorporación de comentarios que ayuden a comprender lo que hizo el programador. Es importante documentar dentro del programa cualquier aspecto que no quede claro de la simple lectura del código. Los comentarios son Imprescindible. Con este fin, el lenguaje ofrece la posibilidad de incluir dentro del programa cualquier texto que pretenda comentar el código escrito. Este texto puede abarcar una o varias líneas, y debe incluirse dentro de los delimitadores mostrados a continuación.

Sintaxis General:

// Comentario simple – siempre se separan por líneas, // debiendo comenzar cada nueva línea con otro par de //

/* Comentario – si se usa el asterisco con 'slash', se puede seguir en varias líneas sin volver a declarar en cada línea el símbolo de comentarios... hasta que se termina con */

1.4 Algunos parámetros del código fuente seleccionados de métricas.

La medición es esencial para cualquier disciplina de ingeniería. Las métricas de software se refieren a un amplio rango de medidas para el software de computadoras dentro del contexto de la planificación del proyecto de software, las métricas de calidad pueden ser aplicadas a organizaciones, procesos y productos los cuales directamente afectan a la estimación de 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, por ejemplo la métrica "líneas de código" caracteriza la propiedad de tamaño de un código fuente asociando un número con él.

Las métricas de software permiten estimar costo y esfuerzo en realizar un proyecto, evaluar la calidad del software realizado, predecir el tiempo invertido en mantenimiento de un programa o validar las mejoras prácticas para el desarrollo del mismo.
Si se desea estimar el costo y el esfuerzo entonces se necesita conocer el tamaño de un programa, si lo que se requiere es medir la complejidad, existe diversidad de métricas como McCabe y Haltstead, por lo que no siempre una sola métrica brinda toda la información que se necesita.

Las métricas se miden en: [21]

- Métricas directas: Son aquellas que se obtienen a través de un proceso de medición directo, es decir, que no involucra a ninguna otro atributo.
- Métricas indirectas: Son aquellas que se obtiene a partir de métricas directas.

Y su clasificación consta de tres aspectos:

- Del proceso: Son los atributos de actividades relacionadas con el software. (duración, esfuerzo, número de incidentes...).
- 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.
- Después de una investigación, se determinó utilizar alguno de los parámetros a comprobar de algunas de las métricas que se han definido para comprobar la calidad en el código fuente. A continuación una breve explicación de algunos parámetros elegidos de métricas.

1.4.1 De la métrica: Chidamber & Kemerer.

Chidamber y Kemerer (MOOSE) adoptan tres criterios a la hora de definir las métricas: capacidad de satisfacer propiedades analíticas, aspecto intuitivo a los profesionales y gestores y facilidad para su recogida automática. Son un punto de referencia en las métricas orientada a objetos (OO).

Se seleccionaron una serie de parámetros como:

- Métodos Ponderados por Clase (MPC):

Asumen que se definen n métodos con complejidad c1, c2,...cn se definen para la clase C. La métrica de complejidad específica que se haya elegido (por ejemplo, la complejidad ciclomática) debe normalizarse de manera que la complejidad nominal para un método toma un valor de 10.

MPC = Sci para cada i=1 hasta n.

El número de métodos y su complejidad indican la cantidad de esfuerzo para implementar y verificar una clase. Cuanto mayor sea el número de métodos, más complejo es el árbol de herencia. Además, a medida que crece el número de métodos para una clase dada, más específica se vuelve y, por lo tanto, menos reutilizable. Por eso, el MP debe mantenerse lo más bajo posible. (En forma de resumen: no es más que la sumatoria de las complejidades de cada función para una clase.)

- Camino más Largo desde la Raíz hasta una Clase (APH):

Longitud máxima del nodo a la raíz del árbol. Cuanto mayor sea, las clases de los niveles más bajos heredarán muchos métodos, lo que conlleva dificultades potenciales al predecir el comportamiento de una clase. Además, la complejidad de diseño será mayor. Sin embargo, valores de APH grandes indican mayor capacidad de reutilización de métodos. (En forma de resumen: es el nivel de la clase más profunda de la jerarquía.)

- Número de Hijos de una Clase (NDD):

Cuantos más descendientes, más reutilización, pero también es posible que algunos descendientes no sean miembros realmente apropiados de la superclase. (En forma de resumen: es la cantidad de hijos para cada clase.)

- Respuesta Máxima de una Clase (RPC):

Número de métodos que pueden ser ejecutados en respuesta a un mensaje recibido por un objeto de esa clase. Cuanto mayor sea RPC, mayor esfuerzo se requiere para su comprobación, y más complejo es el diseño. (En forma de resumen: es el número máximo de métodos que se pueden ejecutar al invocar uno cualquiera en una clase.)

1.4.2 De la métrica: Halstead.

Las métricas de Halstead consideran que un programa está formado por una serie de partículas, las cuales pueden ser consideradas como operadores u operandos. Los operandos son definidos como las variables o constantes que se utilizan en la implementación, mientras que los operadores son los símbolos que afectan el valor u orden del operando.

La teoría de la ciencia del software propuesta por Halstead es probablemente la medida de complejidad mejor conocida y minuciosamente estudiada. La ciencia del software propuso la primera ley analítica y cuantitativa para el software de computadora. Utiliza un conjunto de medidas primitivas que pueden obtenerse una vez que se han generado o estimado el código después de completar el diseño.

Se determinaron los parámetros que a continuación se señalan.

- Parámetro básico n1: Consiste en la cantidad de operadores diferentes en el código.
- Parámetro básico N1: Radica en el número total de ocurrencias de operadores.
- Parámetro básico n2: Es la cantidad de operandos diferentes.
- Parámetro básico N2: Es la cantidad de ocurrencias de operandos.

Después de un breve estudio sobre esta métrica, se puede clasificar a los programas por su longitud (N) como pequeños (N<=100), medianos (N>100 y N<=500), grandes (N>500 y N<=1500) y muy grandes (N>1500), entendiéndose como longitud de un programa (N= N1+N2) a la suma de los operandos que se utilizan en la codificación de un algoritmo.

1.4.3 De la métrica: Línea de código.

El número de líneas de código (LOC) es la medida más usada para medir la longitud del código fuente. Se han realizado muchas propuestas para contarlas. Es útil medir por separado las líneas comentadas (CLOC) para calcular esfuerzo, productividad, etc. También puede ser útil calcular la densidad de los comentarios.

Otra propuesta consiste en contabilizar únicamente el código entregado al cliente. Se cuenta el número de DSI (delivered sourceinstruction) que incluye las declaraciones de datos, las cabeceras y las instrucciones fuente.

El uso principal que tiene contar líneas de código, es poder determinar proporciones entre líneas de código y fallos, es decir, generar medidas (y heurísticas) sobre el número de líneas de código y el número de fallos de una aplicación (‘número de defectos por KLOC (1000 LOC)’).

Como estas existen un sin números de propuestas, es por ello que se establecieron alguno de los parámetros de dicha métrica a tener en cuenta para la revisión del código fuente.

- Líneas Declarativas: Sucede cuando no ejecutan acciones, se usan solo para declarar elementos y cabeceras.
- Líneas Procedurales: Ellas no son declarativas ni son Líneas en Blanco ni son Líneas de Comentarios.
- Líneas en Blanco: No tienen ningún elemento.
- Líneas de Comentarios Simples: Solo tienen un elemento de comentario de una solo línea.
- Líneas de Comentarios Delimitados: Solo poseen elementos de comentarios delimitados.
- Comentarios Simples: Es la suma de todos los comentarios simples del código.
- Comentarios Delimitados: Radica en la suma de todos los comentarios delimitados del código.
- Comentarios: Se considera la suma de los Comentarios Simples y los Comentarios Delimitados.
- Líneas de Comentarios: Es la suma de las Líneas de Comentarios Simples y las Líneas de Comentarios Delimitados.

1.4.4 De la métrica: Li – Henry.

Li y Henry (1993) es una extensión de MOOSE con otras métricas de tamaño y diseño y un sistema de predicción para reusabilidad y mantenibilidad.

Entre los diferentes parámetros que posee, se determinaron:

- Tamaño1: Representa la cantidad total de punto-y-coma en una clase.
- Tamaño2: Constituye la cantidad de atributos locales y métodos en una clase.
- Número de Métodos Localmente Definidos y Heredados: Significa la cantidad de métodos de una clase incluyendo los definidos localmente y los heredados en los casos que las clases sean hijas de otras.

1.5 Sistemas automatizados existentes.

En la actualidad, aumenta la capacidad de automatizar el proceso de pruebas, siendo no menos cierto, que son difíciles de escribir y mantener, difíciles de aplicar en la práctica y son raramente utilizadas en entornos industriales. Algunas herramientas a nivel mundial han surgido para las revisiones más rápidas y perfeccionadas del software creadas por grupos de desarrolladores.

1.5.1 Antecedente a nivel internacional.

JTest: Es el primer sistema automático de búsqueda de bugs (errores en el código de programación) para programadores de Java. Esta nueva tecnología desarrollada por la empresa ParaSoft, utiliza la Test Generation Technolgy (cuya patente está pendiente todavía) para analizar programas de Java. Se trata de una herramienta que permite realizar análisis de código, pruebas unitarias automáticas y cobertura de código, así como generación dinámica de pruebas funcionales.

En el ámbito de los análisis dinámicos JTest es capaz de generar automáticamente todas las pruebas unitarias que sean necesarias, teniendo en cuenta los parámetros de cobertura de código e intentando encontrar pruebas que deriven en errores de ejecución. Genera pruebas funcionales teniendo en cuenta las acciones y los datos, incluyendo peticiones HTTP[2] y JDBC[3]. Se le han agregado varios módulos realizando revisiones de código más productivas y prácticas para las organizaciones, sin

[2] HTTP: Protocolo de transferencia de hipertexto.
[3] JDBC: Interfaz de programación de aplicaciones entre los programas Java independiente de la plataforma y del gestor de bases de datos utilizado.

embargo se requiere de conocimientos técnicos para un mejor uso, con necesidad del usuario de recibir cursos de aprendizaje.

Docklight Scripting: Es una herramienta de testeo automatizado para protocolos seriales de comunicación vía COM, TCP y UDP[4]. Es una edición extendida del Docklight RS232 Terminal / RS232 Monitor[5], incluyendo un lenguaje script sencillo de usar para automatización. Utiliza el motor VBScript, permitiendo escribir las pruebas en un sencillo y conocido lenguaje script. Las funciones básicas y características de Docklight son dotadas a través de un conjunto pequeño y conveniente de instrucciones de script. Esta herramienta de testeo automatizado para protocolos seriales de comunicación tiene el inconveniente de la compra de su licencia cada cierto período de tiempo oscilando el precio de 79 USD a más de 159 USD.

AQtime: Incluye docenas de herramientas de productividad que ayudan a bloquear y eliminar todos los problemas de rendimiento y de memoria y/o fugas de recursos dentro del código por completo y la generación de informes detallados de una cuenta. NET y aplicaciones de Windows. Se construye con un objetivo fundamental, ayudar a entender completamente cómo revisar programas en ejecución.

Usando su conjunto integrado de rendimiento y depuración de perfiladores, AQtime recoge el rendimiento y la memoria, la asignación de recursos de información en tiempo de ejecución y entrega un resumen detallado con todas las herramientas que necesita para comenzar el proceso de optimización. Todo esto se hace sin modificar la aplicación del código fuente. Se cuenta para su aprendizaje con materiales en idiomas ajenos al castellano y el precio de las herramientas oscila de 500 euros a más de 2000 euros.

McCabe TQ: Es interactivo, cuenta con un entorno visual para la gestión de la calidad del software a través de análisis avanzados, ofreciendo una solución global a los desafíos que enfrenta la garantía de calidad y equipos de desarrollo. Comparar esta herramienta permite localizar código redundante, por lo que puede o bien borrar o reescribir, disminuyendo el tamaño de la aplicación y su mantenimiento de esfuerzos. Admite seleccionar criterios de búsqueda predefinidos o crear sus propios criterios de búsqueda para encontrar módulos similares. Selecciona los módulos que desea el partido y especifica qué programas o repositorios desea buscar, aunque no incluye la comprobación de estándares de codificación.

[4] COM, TCP y UDP: Protocolos seriales o puertos o canales de conección.
[5] Docklight RS232 Terminal / RS232 Monitor: herramienta de evaluación, análisis y simulación de protocolos de comunicación serie (RS232,...).

Resource Standard Metrics (RSM): Es una herramienta de métrica de código fuente y análisis de calidad para ANSI C, ANSI C++, C# y Java[6] para usar en todos los sistemas operativos Windows y UNIX. Es rápido y fácil de usar. RSM proporciona métrica estándar y análisis automatizado del código fuente y es válido para cualquier empresa que apunte a cumplir las certificaciones ISO-9001, TickIt y SEI[7]. El código fuente que no compile no pasará correctamente por el analizador RSM. La misma es una herramienta basada en consola, esto implica que para ejecutarse debe usar una consola de comandos o cualquier consola UNIX o Xterm que desee. RSM utiliza un archivo de licencia y un archivo de configuración en el arranque.

Insure++: Es un entorno automatizado en una aplicación de herramientas de prueba C/C++ que detecta errores difíciles de localizar como corrupción de la memoria, asignación de errores de memoria, errores de iniciación de variables, definición de conflictos entre variables, indicador de errores, errores de biblioteca, errores lógicos y errores en algoritmos. Sin embargo tiene licencia la cual hay que pagar cada cierto período de tiempo.

BullseyeCoverage: Es un analizador de código de la cobertura para C + + y C que indica cómo gran parte del código fuente se pone a prueba. Puede usar esta información para rápidamente centrar su esfuerzo de ensayo y determinar las áreas que necesitan ser revisadas. El código cobertura de análisis es útil durante la unidad de verificación, integración de pruebas y la liberación final. Permite crear código más fiable y ahorrar tiempo. Sin embargo la licencia se debe comprar cada cierto período de tiempo, según el cliente determine, oscilando de 500 euros a 1000 euros.

CodeCheck: Esta herramienta permite ver y comparar los distintos códigos que componen o se extraen del Installcode[8]. Puede ser muy útil para comparar el installcode antes de hacer un cambio en el dispositivo que se obtiene posteriormente. Sin embargo no es gratuita.

LDRA: Es una herramienta de control de calidad que ofrece un potente código fuente de pruebas y análisis para la validación y verificación de aplicaciones de software. Es importante cuando el software informático requiere ser fiable, robusto y libre de errores. Se trata de un potente y plenamente integrado suite de herramientas, que permite al software avanzado de técnicas de análisis que deben aplicarse en las etapas claves del desarrollo del ciclo de vida. Sin embargo sus herramientas no

[6] ANSI C, ANSI C++, C# y Java: Lenguajes de programación.
[7] ISO-9001, TickIt y SEI: Organizaciones de normas o certificaciones de calidad.
[8] Installcode: Instalador de código.

contienen la comprobación de estándares de codificación y para su empleo hay que comprarla a precios estimados.

GlowCode: Un juego de herramientas de diagnóstico C++ para programadores Windows. El juego de herramientas incluye todas las esenciales: rastreo de memoria y recursos, detección de fugas, perfil de desempeño y cobertura rápida de código, todo en un paquete con una impresión de huella (impacto de memoria y recursos) muy pequeña. GlowCode ayuda a los programadores a optimizar el rendimiento de las aplicaciones, con herramientas para detectar fugas de memoria y defectos en los recursos, aislar cuellos de botella de rendimiento, el rastreo en tiempo real en la ejecución del programa, asegurar la cobertura de código, así como aislar los errores de código. Cuenta con una licencia oscilando a más 200 euros, por período de tiempo.

Logiscope TestChecker: Representación gráfica de cobertura del código fuente. Evalúa el nivel del cobertura del código, el usuario debe de comprar la licencia por período de tiempos, no realiza estas evaluaciones al código de C# y no cuenta con la verificación de estándares.

Code Coverage: El code coverage es una medida que se utiliza en las disciplinas de comprobación automática (automated testing) de software. Esta medida indica la fracción del software que las pruebas han cubierto. Si las pruebas indican que el software es correcto, el code coverage indica qué partes del software son correctas (esto no es del todo cierto). Es decir, el code coverage es una medida de calidad, pero no de calidad del software, sino de pruebas: indica las partes del software que están comprobando las pruebas y, lo más importante, cuáles partes no se están comprobando. Sin embargo la mayoría de herramientas de code coverage no son capaces de comprobar más que lo que se llama “cobertura de instrucción” o “de línea” y no resultan ser gratuitas.

Logiscope: Es una herramienta para asegurar la calidad de software (QA) que ayuda a aumentar la cobertura de pruebas automatizando las revisiones de código y la identificación y detección de módulos propensos a contener errores para testeo de software y garantía de calidad. Las características de testeo de software personalizables ayudan a identificar los errores en una fase temprana del proceso de desarrollo, por lo que puede aumentar la calidad del software y entregar dentro del plazo y del presupuesto. La licencia se paga por período de tiempo.

JProbe Suite: Esta herramienta, permite detectar los 'puntos calientes' de los componentes de una aplicación JAVA, tales como el uso de la memoria, el uso del CPU, los hilos de ejecución y a partir de ellos, bajar al nivel del código fuente que los provoca ofreciendo una serie de consejos o buenas

prácticas de codificación para la resolución del problema. Entre otros parámetros, se puede ver los métodos o líneas de código que más tiempo insumen en sus llamadas, los métodos con mayor cantidad de invocaciones, etc.

También permite medir el consumo de CPU por clases, métodos y paquetes, monitoriza el número de instancias de cada clase creadas, el consumo de memoria por clases, además de monitorizar el estado de los hilos de las aplicaciones, así como advertir de potenciales estancamientos y otros problemas relativos a sincronización. La parte de la suite que ahora es gratis es JProbe Profiler, disponible para descarga para Windows y Linux. La suite a completo puede descargarse con una licencia temporal para probarla. Los precios incluyen algunos servicios técnicos y para la mayoría de los productos disponibles para descarga, una copia de seguridad en línea y una actualización gratuita a la nueva versión si ésta se publica en un período de 30 días después de la compra. Todas las ventas están sujetas a sus términos y condiciones estándar y a su política de devolución. Los precios oscilan:

Jprobe Suite V7.0 para Windows, Linux, HP, AIX y Solaris -- $ 1 950.00 a $ 6 825.00.

CMT++: Es una herramienta para la medida de la complejidad para C / C + +, la misma es fácil de utilizar para ambos lenguajes. También el código en ensamblador se puede medir con esta herramienta. CMT + + está destinado para organizaciones desarrolladoras de software que se esfuerzan por un proceso de desarrollo productivo resultante en productos de alta calidad. Ayuda a estimar el mantenimiento general del código base y a localizar fácilmente las partes complejas de este. Entonces se pueden evaluar por separado: poniendo especial atención a las pruebas, o tal vez rediseñándolas. Se puede utilizar CMT + + también para medir la cantidad de código que se tiene: líneas físicas, líneas de comentarios, líneas de programa, declaraciones. Es una herramienta que a pesar de sus características no es gratuita y la documentación para su aprendizaje se encuentra en otros idiomas exceptuando el castellano.

CTC++: Así mismo ocurre con esta herramienta, posee los mismo inconvenientes que CMT+ +, aunque no se debe de dejar de mencionar que Testwell CTC + + (Test Analizador de Cobertura para C y C+ +) es una herramienta de cobertura código/prueba potente y fácil de usar la cual muestra las partes del código que han sido ejecutadas (probadas). La herramienta analiza todos los niveles de cobertura requeridos en proyectos "críticos" y ayuda a garantizar una mayor calidad del código. Testwell CTC+ + puede utilizarse para obtener las certificaciones en la industria automotriz, aérea y médica.

1.5.2 Antecedentes a nivel nacional.

Aún no existe una herramienta basada en pruebas de Caja Blanca con determinadas funcionalidades. En la Universidad de las Ciencias Informáticas, solo se cuenta con la existencia de un trabajo de diploma investigativo que forma parte del material de apoyo consultado en la realización de esta investigación, titulado Procedimiento general de pruebas de Caja Blanca aplicando la técnica del Camino Básico, siendo los autores las Ings. Yaimí Márquez y Yenni Valdés, donde se explica el procedimiento de dicho método. También existen otros trabajos de diploma, pero describiendo procedimiento para diferentes procesos de pruebas a desarrollar referidos al método de prueba de Caja Negra.

Existen en algunos proyectos productivos en la UCI herramientas que aplican diferentes pruebas de Caja Negra, probando la funcionalidad del software como la carga o stress de dichos productos y en la minoría (prácticamente ninguno) se aplica el método de Caja Blanca de forma manual a un pequeño fragmento de código, al ser esta técnica mayormente conocida y a la vez muy compleja de llevarse acabo, resultando un trabajo tedioso, lento y engorroso. La carencia del empleo de este tipo de pruebas impide que el programador conozca el grado o nivel de su codificación y qué tan óptimo puede resultar para su aplicación.

1.6 Análisis comparativo de otras soluciones existentes con la propuesta.

La Universidad de las Ciencias Informáticas como máximo exponente del desarrollo tecnológico del país, ha tenido entre sus prioridades la de convertirse en una gran industria del software y para ello, como antes se ha mencionado, es preciso el desarrollo de productos software con una alta calidad. Al no existir ninguna herramienta encargada de automatizar los procesos de pruebas de Caja Blanca, surge la necesidad de desarrollar un sistema informático que, aprovechando las tecnologías recientes y el conocimiento de las mismas, logre automatizar el proceso de revisiones de software en el grupo de Calidad de la Facultad 7.

No se optó por la selección de ninguna de las herramientas estudiadas anteriormente para la automatización de dicho proceso, pues ninguno de estos sistemas responden a las características particulares del proceso que se lleva a cabo en la Facultad o sea, no solucionan las necesidades del proceso de pruebas de Caja Blanca que se desea por parte del grupo de Calidad.

Además del costo que implica poder adquirir una tecnología propietaria imposibilita utilizar determinados sistemas. Y aunque alguno de estos son gratis, no concuerdan con el propósito y/o

entorno necesario, a esto se le une, la inexperiencia del usuario para manipularla y la falta de documentación para prepararse y operar con las mismas, pues muchas carecen de ella o están en otro idioma.

Incluyéndose la compra de las licencias periódicamente oscilando de 500 USD o euros hasta más de 3000 USD o euros en dependencia de las características de las mismas. Muchas no resultan tener una interfaz amigable para el usuario o demoran cierto tiempo en darle respuesta al cliente cuando analizan el código.

Esto origina ciertas deficiencias en el desarrollo del software y su calidad imperiosa para la comercialización del mismo o la funcionalidad que el cliente persigue. Por tal motivo, se propone el análisis y diseño de una herramienta de pruebas de Caja Blanca que permita comprobar fragmentos de código fuente de otras aplicaciones, demostrando la rapidez de la respuesta, al mismo tiempo que la optimización de la implementación y el cumplimiento de estándares y parámetros de codificación.

1.7 Metodologías de desarrollo de software a considerar.

La metodología de desarrollo de un software es el plano de apoyo, la guía que conducirá a realizar un software de calidad, es el conjunto de métodos que se deben de realizar para el desarrollo de un software práctico.

Una adecuada selección de la metodología servirá para que la confección de la herramienta no parezca complicada, rigurosa y difícil de controlar, además que se podrá lograr la satisfacción por parte del cliente.

En un proyecto de desarrollo de software la metodología define ¿Quién debe hacer Qué?, ¿Cuándo y cómo debe hacerlo? Las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigen que el proceso sea configurable, aunque no exista una metodología de software universal.

1.7.1 RUP.

El Proceso Unificado de Desarrollo de Software (Rational Unified Process) constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos basado íntegramente en Lenguaje Unificado de Modelado UML como soporte a la metodología.

"El RUP es un producto de Rational (IBM). Se caracteriza por ser: [22]

- *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 le denomina iteración y pueden o no representar un incremento en el grado de terminación del producto completo.*
- *Estar 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."*

RUP es uno de los procesos más generales de los existentes actualmente, ya que en realidad está pensado para adaptarse a cualquier proyecto, y no tan solo de software.

El proceso puede describirse en dos dimensiones, o a lo largo de dos ejes:

- El eje horizontal representa tiempo y muestra el aspecto dinámico del proceso, expresado en términos de ciclos, fases, iteraciones, y metas.
- El eje vertical representa el aspecto estático del proceso; como está descrito en términos de actividades, artefactos, trabajadores y flujos de trabajo.

Define cuatro fases: Inicio, Elaboración, Construcción y Transición. Y nueve flujos de trabajo, seis 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).

RUP pretende implementar las mejores prácticas actuales en Ingeniería de Software:

- Desarrollo iterativo del Software.
- Administración de requerimientos.
- Uso de arquitecturas basadas en componentes.
- Modelación visual del software.
- Verificación de la calidad del software.
- Control de cambios.

1.7.2 Metodologías ágiles.

"Los procesos ágiles de desarrollo de software, conocidos también como metodologías livianas, intentan evitar los tortuosos y burocráticos caminos de las metodologías tradicionales enfocándose en la gente y los resultados." [23]

Programación Extrema (XP).

Mientras que RUP intenta reducir la complejidad del software por medio de estructura y la preparación de las tareas pendientes en función de los objetivos de la fase y actividad actual, XP, como toda metodología ágil, lo intenta por medio de un trabajo orientado directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción.

La programación extrema o Extreme Programming (XP) es la más destacada de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Las características fundamentales del método son:

- Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- Programación por parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone que la mayor calidad del código escrito de esta manera el código es revisado y discutido mientras se escribe es más importante que la posible pérdida de productividad inmediata.
- Corrección de todos los errores antes de añadir nueva funcionalidad.
- Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda

corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores sean detectados.

- Integración continua: Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.
- 40 horas por semana: Se debe trabajar un máximo de 40 horas por semana. No se trabaja horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo. [24]

Desarrollo guiado por la funcionalidad (FDD, Feature -Driven Development).

FDD se podría considerar a medio camino entre RUP y XP, aunque al seguir siendo un proceso ligero es más similar a este último. Está pensado para proyectos con tiempo de desarrollo relativamente cortos (menos de un año).

Define un proceso iterativo que consta de 5 pasos:

- Desarrollo de un modelo general.
- Construcción de la lista de funcionalidades.
- Plan de entrega en base a las funcionalidades a implementar.
- Diseñar en base a las funcionalidades.
- Implementar en base a las funcionalidades.

Las iteraciones son cortas (hasta 2 semanas). Se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software.

Scrum.

Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años.

Está especialmente indicado para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos:

- El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente.

- La segunda característica importante son las reuniones a lo largo de toda la duración del proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

Crystal Methodologies.

Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear [9](3 a 8 miembros) y Crystal Orange[10] (25 a 50 miembros).

Dynamic Systems Development Method7 (DSDM).

Define el marco para desarrollar un proceso de producción de software. Se caracteriza por ser un proceso iterativo e incremental y el equipo de desarrollo y el usuario trabajan juntos. Propone cinco fases: estudio viabilidad, estudio del negocio, modelado funcional, diseño y construcción y finalmente implementación. Las tres últimas son iterativas, además de existir realimentación a todas las fases.

Adaptive Software Development (ASD).

Sus principales características son: iterativo, orientado a los componentes software más que a las tareas y tolerante a los cambios. El ciclo de vida que propone tiene tres fases esenciales: especulación, colaboración y aprendizaje. En la primera de ellas se inicia el proyecto y se planifican las características del software; en la segunda desarrollan las características y finalmente en la tercera se revisa su calidad, y se entrega al cliente. La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.

[9] Crystal Clear: color claro.
[10] Crystal Orange: color naranja

1.8 Herramientas utilizadas para el desarrollo del sistema.

Herramientas CASE (ingeniería de software asistida por computadora, en inglés, Computer Arded Software Engineering.)

Se puede definir a una herramienta CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software. [25]

Ventajas con la utilización de las herramientas CASE:

- Permiten el incremento en la velocidad de desarrollo de los sistemas.
- Permiten a los analistas tener más tiempo para el análisis y diseño y minimizar el tiempo para codificar y probar.

En las etapas del proceso de desarrollo de software permiten:

- Automatizar el dibujo de diagramas.
- Ayudar en la documentación del sistema.
- Ayudar en la creación de relaciones en la base de datos.
- Generar estructuras de código.
- Aumentan la productividad. Esto se consigue a través de la automatización de determinadas tareas, como la generación de código y la reutilización de objetos o módulos.

1.8.1 Rational Rose Enterprise Edition.

Es una herramienta para “modelado visual”, que forma parte de un conjunto más amplio de herramientas que juntas cubren todo el ciclo de vida del desarrollo de software. Permite completar una gran parte de las disciplinas (flujos fundamentales) del proceso unificado de Rational (RUP) e incluye un conjunto de herramientas de ingeniería inversa y generación de código que nivelan el camino hasta el producto final.

El Rational es una herramienta CASE basada en UML que permite crear los diagramas que se van generando durante el proceso de ingeniería en el desarrollo del software. Es completamente compatible con la metodología RUP, brinda muchas facilidades en la generación de la documentación del software que se están desarrollando, además posee un gran número de estereotipos predefinidos

que viabiliza el proceso de modelación del software. Es capaz de generar el código fuente de las clases definidas en el flujo de trabajo de diseño, pero tiene la limitación de que aún hay varios lenguajes de programación que no soporta o que sólo lo hace a medias.

Rational se encuentra a la cabeza en cuanto al desarrollo del Unified Modeling Language (UML), que se ha convertido en la notación estandarizada empleada en Rational Rose para especificar, visualizar y construir desarrollos de software y sistemas.

Rose es una herramienta con plataforma independiente que ayuda a la comunicación entre los miembros del equipo a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas.

Una de las grandes ventajas de Rose es que permite a los arquitectos de software y desarrolladores visualizar el sistema completo utilizando un lenguaje común, además los diseñadores pueden modelar sus componentes e interfaces en forma individual y luego unirlos con otros componentes del proyecto.

1.8.2 Visual paradigma-UML.

Visual Paradigm para UML es una de las herramientas UML CASE del mercado, considerada muy completa y fácil de usar, con soporte multiplataforma y que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. Fue creada para el ciclo vital completo del desarrollo de software que lo automatiza y acelera, permitiendo la captura de requisitos, análisis, diseño e implementación. Visual Paradigm-UML también proporciona características tales como generación del código, ingeniería reversa y generación de informes.

Tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de base de datos a partir del esquema de clases. Permite invertir código fuente de programas, archivos ejecutables y binarios en modelos UML al instante, creando de manera simple toda la documentación. Está diseñada para usuarios interesados en sistemas de software de gran escala con el uso del acercamiento orientado a objeto, además apoya los estándares más recientes de las notaciones de Java y de UML. Incorpora el soporte para trabajo en equipo, que permite que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios hechos por sus compañeros.

1.9 Lenguaje Unificado de Modelado (UML).

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un

lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

"El punto importante para notar aquí es que 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 sí mismo qué metodología o proceso usar." [26]

Principales características:

- Posibilita mejores tiempos totales de desarrollo.
- Permite modelar sistemas (y no sólo de software).
- Tecnología orientada a objetos.
- Establecer conceptos y artefactos ejecutables.
- Encaminar el desarrollo del escalamiento en sistemas complejos de misión crítica.
- Crear un lenguaje de modelado utilizado tanto por humanos como por máquinas.
- Mejor soporte a la planeación y al control de proyectos.
- Alta reutilización y minimización de costos.

Ventajas de UML:

- El sistema de software profesional es diseñado y documentado antes de que sea codificado. Se conoce exactamente lo que se puede obtener, por adelantado.
- Los lógicos 'agujeros' en el diseño del sistema podrán ser descubiertos antes sobre los diagramas del mismo. El software se comportará de la forma esperada y surgirán menos sorpresas.
- El diseño total del sistema dicta el modo en que se desarrollará el software. Las decisiones finales se harán antes de que se encuentre código mal escrito. Ahorrándose tiempo en el desarrollo.

- De ser necesario hacer modificaciones en el sistema, es mucho más fácil hacerlo sobre la documentación UML. Hay que recurrir mucho menos a rehacer un nuevo estudio.
- Si se incorporan nuevos desarrolladores al proyecto, los diagramas UML les permitirá hacer rápidamente una idea del sistema.
- *"La comunicación con los desarrolladores del proyecto y con desarrolladores externos, es mucho más eficiente."* [27]

1.10 Lenguaje de Programación.

Un lenguaje de programación permite a uno o más programadores especificar de manera precisa sobre qué datos una computadora debe operar, cómo deben ser estos almacenados, transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Lo anterior ocurre mediante un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural, tal como sucede con el lenguaje léxico. Una característica relevante de los lenguajes de programación es precisamente que más de un programador puede tener un conjunto común de instrucciones que puedan ser comprendidas entre ellos para realizar la construcción del programa de forma colaborativa.

Teniendo en cuenta la variedad de lenguajes de programación actuales: C, C++, Visual Basic, Delphi, toda la gama de programación .NET, Java, Ruby, Phyton, Eiffel, Smalltalk, cuál será el mejor lenguaje de programación para desarrollar aplicaciones robustas y escalables de escritorio.

¿C++?

El C++ (pronunciado "ce más más" o "ce plus plus") es un lenguaje de programación, que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos.

Está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatismos trae (obliga a hacerlo casi todo manualmente al igual que C) lo que "dificulta" mucho su aprendizaje.

Le falta una API[11] concisa y concreta para el desarrollo orientado a objetos, como es el hecho de clases base que brinden soporte a la programación orientada a objetos, y más allá tampoco tiene una

[11] PI: Interfaz de Programación de Aplicaciones.

API breve de programación para aplicaciones GUI[12] (Si bien hay muchos frameworks[13], no hay nada estandarizado, ni siquiera en la sintaxis y normas de programación).

¿Java?

Java es un lenguaje orientado a objetos que alcanzó su madurez con la popularización de Internet. La expansión de este lenguaje entre la comunidad de programadores ha sido vertiginosa y se ha impuesto como el paradigma de los lenguajes de programación orientados a objetos. Es un lenguaje neutral, portable, robusto, estable, independiente de la plataforma, sencillo de aprender para programadores que hayan trabajado previamente con lenguajes orientados a objetos.

El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

En las primeras versiones de la plataforma Java existían importantes limitaciones en las APIs de desarrollo gráfico (AWT). Desde la aparición de la librería Swing la situación mejoró substancialmente. Sin embargo no puede decirse que Java no tenga grietas, ni que se adapta completamente a todos los estilos de programación, todos los entornos, o todas las necesidades.

¿Visual Basic?

Visual Basic constituye un entorno de desarrollo integrado o en inglés Integrated Development Enviroment (IDE) que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código (programa donde se escribe el código fuente), un depurador (programa que corrige errores en el código fuente para que pueda ser bien compilado), un compilador (programa que traduce el código fuente a lenguaje de máquina), y un constructor de interfaz gráfica o GUI (es una forma de programar en la que no es necesario escribir el código para la parte gráfica del programa, sino que se puede hacer de forma visual).

Es un software propietario por parte de Microsoft, por tanto nadie que no sea del equipo de desarrollo de esta compañía decide la evolución del lenguaje. No existe forma alguna de exportar el código a otras plataformas diferentes a Windows. Tiene fuerte dependencia de librerías y componentes en las versiones 6.0 y anteriores, lo que dificulta la distribución de los desarrollos entre máquinas. Visual

[12] GUI: Interfaz gráfica de usuario.
[13] frameworks: marcos.

Basic es fácil de programar pero las aplicaciones que salen no son ni robustas ni escalables en cuanto a código.

¿Eiffel y Smalltalk?

Eiffel es un lenguaje de programación orientado a objetos centrado en la construcción de software robusto. Su sintaxis es parecida a la del lenguaje de programación Pascal. Una característica que lo distingue del resto de los lenguajes es que permite el diseño por contrato desde la base, con precondiciones, poscondiciones, invariantes y variantes de bucle, invariantes de clase y aserciones.

Mientras que Smalltalk es un sistema informático que permite realizar tareas de computación mediante la interacción con un entorno de objetos virtuales. Metafóricamente, se puede considerar que es un mundo virtual donde viven objetos que se comunican mediante el envío de mensajes. Se busca que todo lo que se pueda realizar con un ordenador se pueda realizar mediante un Sistema Smalltalk, por lo que en algunos casos se puede considerar que incluso sirve como Sistema Operativo. Es considerado el primero de los lenguajes orientados a objetos (OOP). Para este lenguaje todo es un objeto, incluidos los números reales o su propio entorno.

¿Ruby o Phyton?

Python es apreciado como la "oposición leal" a Perl, lenguaje con el cual mantiene una rivalidad amistosa. Los usuarios de Python consideran a éste mucho más limpio y elegante para programar.

Permite dividir el programa en módulos reutilizables desde otros programas Python. Es un lenguaje interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa.

Mientras que Ruby es un lenguaje de programación reflexivo y orientado a objetos (lenguaje interpretado), que combina una sintaxis inspirada en Python, Perl con características de programación orientada a objetos similares a Smalltalk. Comparte también funcionalidad con otros lenguajes de programación como Lisp, Lua, Dylan y CLU.

Ruby parece un buen lenguaje orientado a objetos pero para scripts al igual que Phyton.

¿C#?

C# (pronunciado "si sharp") es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma.NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes (más notablemente de Delphi y Java).

Aunque C# forma parte de la plataforma.NET, ésta es una interfaz de programación de aplicaciones; mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado el que provee el Framework de DotGNU - Mono es que no generen programas para dicha plataforma, sino para una plataforma diferente como Win32 o UNIX / Linux.

Presenta entre otras características:

- Sencillez y Modernidad.
- Orientación a objetos y a componentes.
- Gestión automática de memoria.
- Instrucciones seguras.
- Extensibilidad de tipos básicos, de operadores y de modificadores.
- Versionable, eficiente y compatible.

1.11 Tecnologías y herramientas a utilizar.

Como resultado del análisis hecho a partir del estudio realizado durante el presente capítulo, se pudo concluir que, dada la necesidad de diseñar un nuevo sistema que automatice el proceso de pruebas de software, específicamente el método de pruebas de caja blanca en los proyectos de calidad, el sistema deberá estar concebido bajo la metodología RUP que constituye la metodología estándar más utilizada para el diseño, implementación y documentación de sistemas y que se basa a su vez en UML (Lenguaje Unificado de Modelado). Además es altamente configurable, ya que permite construir solamente los artefactos que se necesiten para el desarrollo de un producto software. Para documentar el desarrollo del software se utilizará la herramienta Rational Rose Enterprise Edition

2003 ya que esta herramienta es muy completa y ofrece amplias potencialidades. Para la implementación teniendo en cuenta las particularidades del sistema a desarrollar, se utilizará la plataforma SharpDevelop, con lenguaje de programación C# por las características que posee.

Conclusiones

Durante la descripción del capítulo se detallaron y argumentaron los principales conceptos y aspectos a lo largo de la investigación y que están relacionados con la temática propuesta, seleccionándose la técnica del camino básico por ser la más conocida y de cierta forma, más fácil de automatizar.

Como resultado del análisis realizado, se pudo ultimar que ninguno de los sistemas estudiados, relacionados con la realización de pruebas al código fuente, responden a las características y necesidades del grupo de calidad de la Facultad. Además presentan inconvenientes, por lo que se necesita diseñar un nuevo sistema que automatice el proceso mencionado. También se realizó un estudio de las principales tecnologías y herramientas existentes y se seleccionaron las más adecuadas para el desarrollo de la aplicación.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

En el presente capítulo se concibe la descripción de la propuesta que trae este trabajo de diploma, para ello se describen los procesos del negocio que tiene que ver con el objeto de estudio, pero debido a la poca estructuración de esos procesos, para que resulte más entendible el contexto en que se ubica el sistema se necesita definir conceptos que se agrupan en un Modelo de Dominio, para capturar correctamente los requisitos y poder construir un sistema correcto.

Además se enumeran los requisitos funcionales y no funcionales, incluyéndose la descripción de los actores y casos de uso resultantes del flujo de trabajo de Requisitos.

2.1 Objeto de Automatización.

En el proceso de revisión de un software es muy importante llevar a cabo el análisis del código fuente mediante las pruebas de Caja Blanca, es por ello el intento de automatización de este tipo de pruebas usando la técnica del Camino Básico en el proyecto de Calidad de la facultad 7. Con esta herramienta se pretende viabilizar los procesos de pruebas, además de perfeccionarlos e incrementarlos y de esta forma mejorar la situación existente actualmente en la Facultad en cuanto a la revisión de un sistema.

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

El grupo de Calidad de la facultad 7, que también apoya al Laboratorio Central de Calidad de la UCI, desempeñan diversas tareas relacionadas con la entrega de productos de alta calidad como resultado de un proceso de desarrollo de software, entre ellas se enmarcan el proceso de pruebas a los productos de software terminados, teniendo en cuenta la planificación o las estrategias de pruebas a seguir en el momento que son efectuadas garantizando así una mejor práctica de dicho proceso y lográndose que el sistema obtenga la calidad pretendida por el cliente.

Actualmente las personas que integran dicho proyecto no tienen la experiencia necesaria para llevar a cabo con profundidad y efectividad el proceso de control de la calidad, sin embargo se le realizan las pruebas a un software recurriendo a algunas técnicas de las pruebas de caja negra. Comenzándose con la ejecución de pruebas exploratorias con varias iteraciones, se aplican listas de chequeo y posteriormente se crea el diseño de casos de pruebas basándose en la descripción de los Casos de Uso y en la especificación de los requerimientos, luego es aplicado dicho diseño.

En el año anterior (2007), se hizo una investigación sobre el procedimiento del método de pruebas de Caja Blanca, específicamente la técnica del camino básico.

Este procedimiento comienza cuando se entrega la documentación que contiene el código fuente del sistema o módulo al equipo de probadores, y se determina por los líderes de prueba y del proyecto los estándares que se definieron por los programadores, para así, hacer más legible y fácil la revisión. Posterior a ello, el equipo de probadores determina los criterios de particiones a emplear, siendo:

- Por funcionalidad de los métodos o formas.
- O por regiones de código definidas.
- Y de no ser muy complejos, por el criterio de caso de uso.

Siguiendo el mismo orden del programador y por tanto, del programa en cualquier caso. Después se traduce cada segmento de código en un grafo de flujo, de ahí se calcula la complejidad ciclomática que proporciona una medición cuantitativa de la complejidad lógica del programa y se continúa con la obtención de los caminos básicos. Con el valor de la complejidad ciclomática se define el número de caminos independientes (también se obtienen por el grafo de flujo) y proporciona el límite superior para el número de casos de pruebas que se deben de realizar para avalar que se ejecute al menos una vez cada sentencia.

Al terminar este procedimiento se prosigue con la derivación de los casos de pruebas, que consiste en obtener las entradas de estos casos de pruebas a partir del código fuente y establecer los resultados esperados con la realización de los mismos, conformándose un documento con la inserción de los errores encontrados para emitirlos más tarde al líder del proyecto.

También se realiza una revisión detallada, línea a línea del código, verificando si cumple con los estándares y algunos parámetros de codificación que se definen con anterioridad por ambos equipos.

2.1.2 Análisis crítico de la ejecución de los procesos.

Actualmente el procedimiento general de pruebas de caja blanca, utilizando específicamente la técnica del camino básico no se aplica en el grupo de calidad de la facultad 7, aunque no deja de ser importante dicho método para verificar la calidad de un producto de software, pues sería otra variante para medir la funcionalidad del mismo, además de ponerlo en otra de las tantas situaciones posibles para intentar dejarlo libre de errores.

Pues a pesar de ser esta técnica la más conocida y por consiguiente estudiada, el personal que integra el grupo de calidad no cuenta con la preparación suficiente para aplicarla, además de resultar bastante engorrosa y tediosa la realización de la misma de una forma manual.

No resultaría eficiente la ejecución de dicha prueba de esta forma artesanal pues se derrocharía el tiempo y el probador se agotaría al realizar los cálculos de la complejidad ciclomática o la representación del grafo de flujo de una forma poco factibles y obsoleta. Se tiende a consumir más tiempo, esfuerzo y el proceso puede no resultar del todo satisfactorio, dado que se puede perder información, en caso de que se escapen algunos pasos a tener en cuenta o no detecten algunos métodos u operaciones del código para la representación del grafo o el cálculo de la complejidad ciclomática.

También se retrasaría la entrega de las no conformidades al líder del proyecto de sistema a probar por desempeñar este tipo de prueba de una forma torpe y no automatizada, demorando la liberación de dicho producto al mercado.

Prolongadamente, todos estos aspectos negativos pueden repercutir de forma notoria en la efectividad y duración del período de prueba de un producto, más la validez del mismo, ya que no existen herramientas automatizadas que proveen los mecanismos para darle solución a estas problemáticas, solo resulta favorable en poder aplicar esta otra variante de prueba al software para ponerlo en todos los escenarios permisibles.

2.2 Modelo del Dominio.

El modelo de dominio muestra las clases conceptuales significativas en el dominio del problema, captura los tipos más importantes de objetos que existen o los eventos que suceden en el entorno donde estará la herramienta. Es considerado un subconjunto del llamado modelo de objetos del negocio.

Durante el desarrollo de la herramienta no se pudo contactar procesos bien definidos en el entorno del negocio. Se hizo difícil determinar los elementos más importantes del sistema y sus interconexiones, así como el establecimiento de las reglas de funcionamiento. Sin embargo se pueden identificar personas, eventos, transacciones y objetos involucrados en ese entorno que no está bien delimitado, por lo que se hizo necesario un modelado del domino perteneciente a la solución, mostrado en la Tabla 2.

Esto *"ayuda a los usuarios, a utilizar un vocabulario común para poder entender el contexto en que se ubica el sistema. Y para capturar correctamente los requisitos y poder construir un sistema correcto se necesita tener un firme conocimiento del funcionamiento del objeto de estudio. Este modelo va a contribuir posteriormente a identificar algunas clases que se utilizarán en el sistema."* [28]

2.2.1 Descripción de los conceptos principales.

Se denominará **Usuario** a cualquier persona que trabaje con la aplicación, sin tener en cuenta la categoría.

Se considera **Parámetros del Código Fuente** a la propiedad o característica de una métrica, u orden o valor que se le pasa a la función y ésta variará su comportamiento.

Se llamará **Estándares de Codificación** a la norma técnica que se puede utilizar como punto de comparación para evaluar la calidad del servicio o para alertar sobre los problemas en el mismo, las cuales se comprobarán con el fragmento de código seleccionado por el usuario, verificándose el cumplimiento de la misma.

Se le denomina **Reporte** a la representación de un contenido o datos en un formato determinado.

Se considera **Fragmento de Código Fuente** a la porción de texto escrito generalmente por una persona, se utiliza como base para generar otro código que posteriormente será interpretado o ejecutado por una computadora. Normalmente se refiere a la programación de software. Que este será copiado o cargado desde la plataforma en la que el usuario esté trabajando.

Se denomina **Nodo** a la representación de cero, una o varias sentencias en secuencia o procedimentales. Cada nodo comprende como máximo una sentencia de decisión (bifurcación). Un solo nodo puede corresponder a una secuencia de procesos o a una sentencia de decisión. Puede ser también que existan nodos que no se asocien, se utilizan principalmente al inicio y final del grafo.

Se nombra **Arista** a las líneas que unen dos nodos, representan el flujo de control, son análogas a las representadas en un diagrama de flujo. Una arista debe terminar en un nodo, incluso aunque el nodo no represente ninguna sentencia procedimental.

Se designa **Grafo de Flujo** a la representación del flujo de control lógico de un programa mediante una notación. Para ello se tiene en cuenta los nodos, aristas y regiones.

Se considera **Matriz** al conjunto de elementos de cualquier naturaleza aunque, en general, suelen ser números (en este caso serían ceros y unos) ordenados en filas y columnas, dispuestos en forma rectangular.

Se le llama **Caminos Independientes** a cualquier camino del programa que incluye nuevas instrucciones de un proceso o una nueva condición.

Se considera **Complejidad Ciclomática** a la medida de la complejidad lógica de un módulo “D” y el esfuerzo mínimo necesario para calificarlo. Es el número de rutas lineales independientes de un módulo “D”, por lo tanto es el número mínimo de rutas que deben probarse.

2.2.2 Diagrama del Modelo del Dominio.

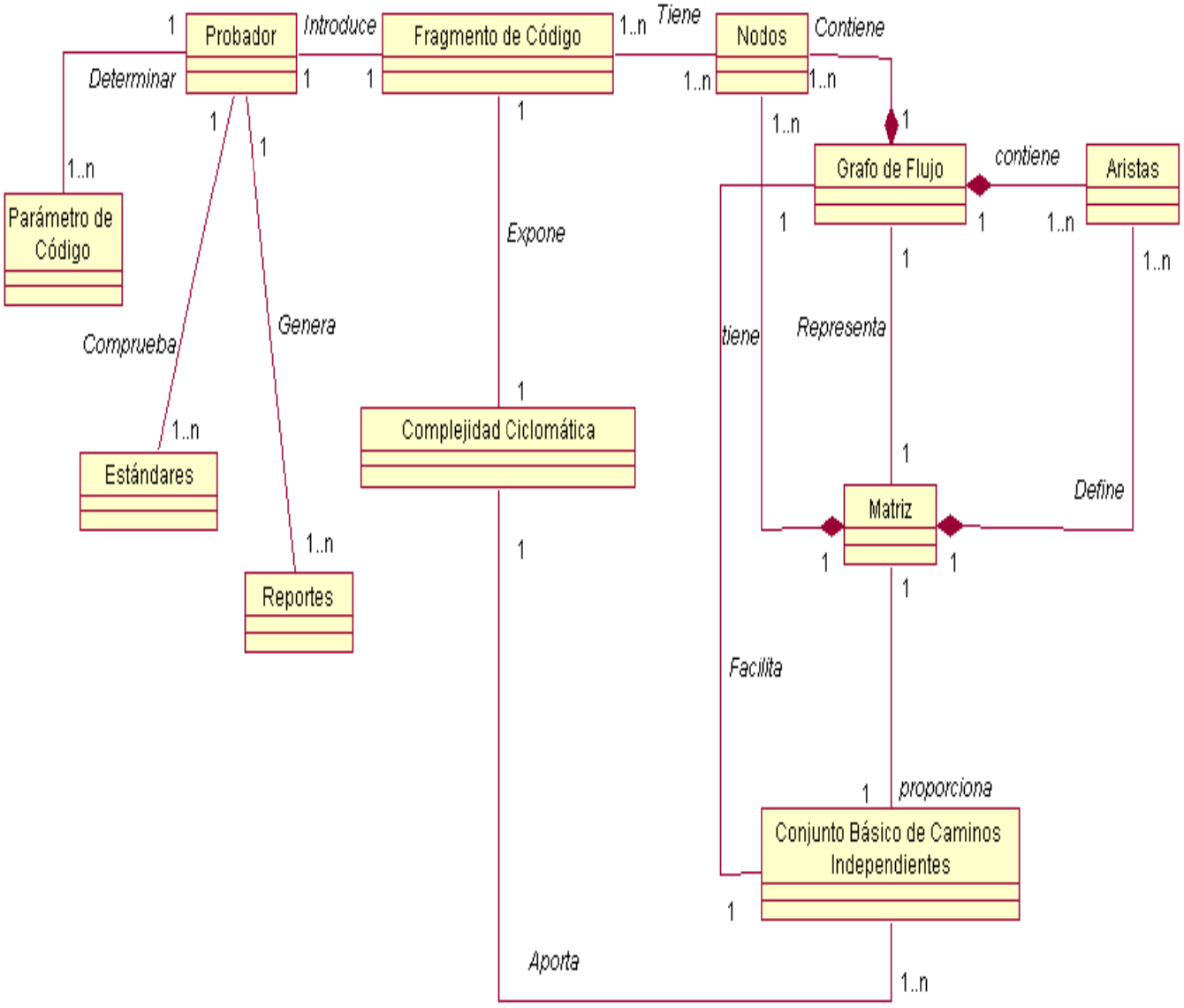
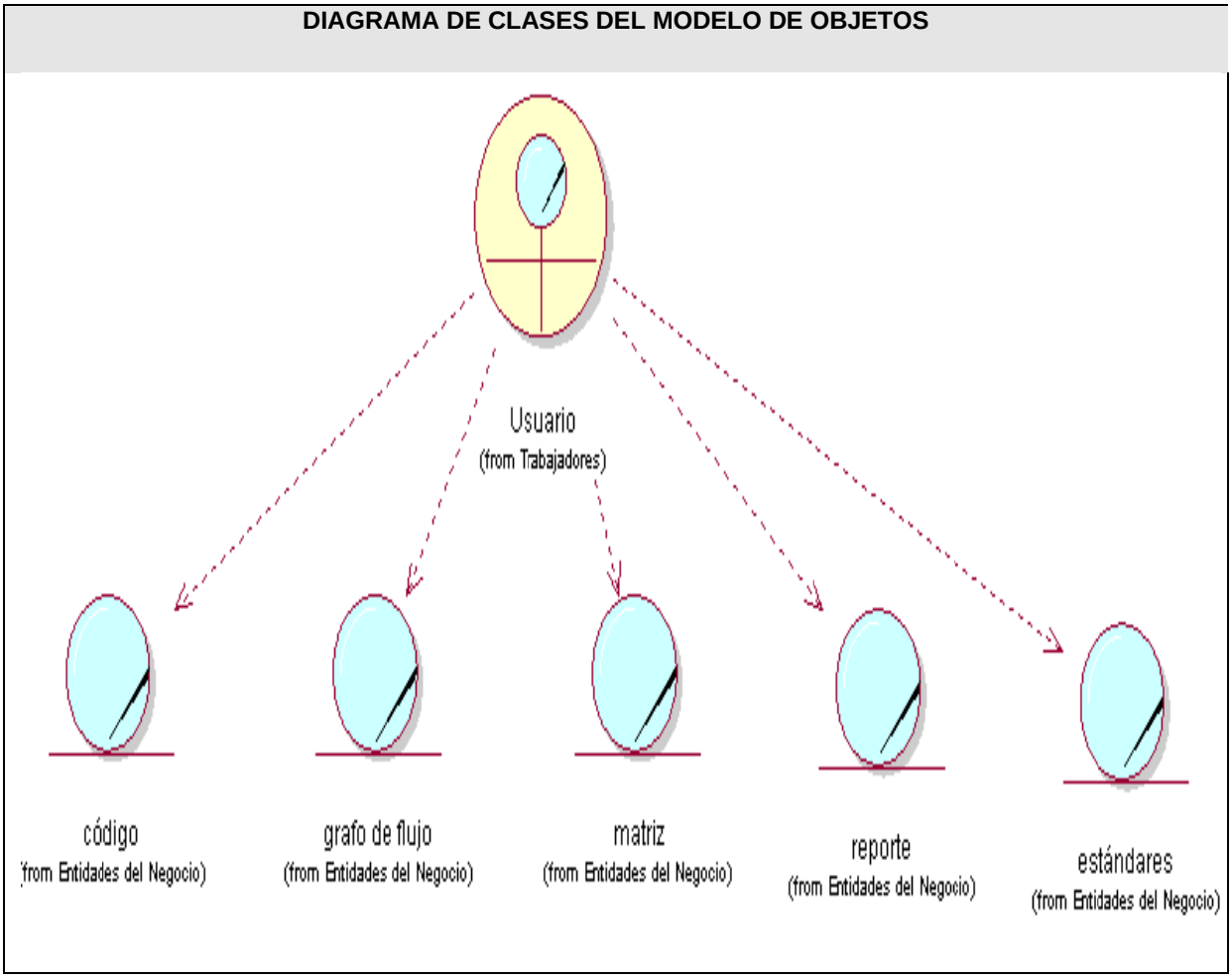


Figura 2: Diagrama del modelo de dominio.

2.2.3 Diagrama de clases del modelo de objetos.

DIAGRAMA DE CLASES DEL MODELO DE OBJETOS



2.2.4 Trabajadores del negocio.

Un trabajador del negocio representa a personas o sistemas dentro del negocio que son los que realizan las actividades que están comprendidas dentro de un caso de uso. Estos trabajadores están dentro de la frontera del negocio, son los que en un futuro se convertirán en usuarios del sistema que se quiere construir. Ver Tabla 2.0.

Tabla 2. Descripción de los trabajadores del negocio.

Trabajadores del negocio	Justificación
Probador	Es el encargado de realizarle las pruebas a un fragmento de código. Puede determinar la complejidad ciclomática y el cumplimiento de los estándares de codificación y los parámetros del código fuente.

2.3 Especificación de los requisitos de software.

Los requerimientos del sistema es uno de los aspectos más importantes a considerar cuando se desarrolla un software, pues estos constituyen la base, el fundamento de la solución propuesta y el argumento para desarrollar el modelado del sistema. Por eso la captura de los requerimientos es uno de los procesos críticos de la ingeniería del software.

2.3.1 Requisitos Funcionales.

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir, especifican acciones que el sistema debe ser capaz de realizar. Ellos deben de ser comprensibles por los clientes, usuarios y desarrolladores, deben tener una sola interpretación y estar definidos en forma medible y verificable.

Es por ello que una vez conocido los conceptos que rodean al objeto de estudio, se analiza ¿Qué debe hacer el sistema para que se cumplan los objetivos planteados al inicio del trabajo de diploma?, enumerándose a través de requerimientos funcionales las instrucciones que el sistema deberá ser capaz de efectuar. Dentro de ellos se incluyen las acciones que podrán ser ejecutadas por el usuario, las acciones ocultas que debe realizar el sistema, y las condiciones extremas a determinar por el sistema.

A continuación se muestran los requisitos que debe cumplir el sistema:

RF1 – Calcular complejidad ciclomática por código fuente.

- RF1.1– Exportar matriz.

RF2 – Calcular complejidad ciclomática por matriz.

RF3 – Determinar conjunto básico de caminos independientes.

RF4 – Importar matriz.

RF5 – Gestionar configuración de estándares de codificación.

- RF5.1 – Abrir configuración de estándares existentes.
- RF5.2 – Guardar como otra configuración de estándares propuesta por el usuario.
- RF5.3 – Mostrar configuración de estándares propuesta por el usuario.
- RF5.4 – Modificar la configuración de estándares de codificación.

RF6 – Comprobar estándares de codificación.

RF7 – Generar reporte de parámetros de codificación.

- RF7.1 – Reporte de Archivos.
- RF7.2 – Reporte de Clases.

RF8 – Gestionar reportes.

- RF8.1 – Guardar reporte.
- RF8.2 – Mostrar reporte.
- RF8.3 – Imprimir reporte.

2.3.2 Requisitos No Funcionales.

Con el propósito de satisfacer al máximo las necesidades del grupo de Calidad de la facultad 7, así como la calidad de la herramienta se definió un listado de las propiedades o cualidades que el producto debe tener, las cuales se describen a continuación.

Apariencia o Interfaz Externa:

- La herramienta será sencilla, fácil de emplear y muy amigable para el usuario ya que no estará cargada de imágenes ni colores, solo referirá las funcionalidades explícitas, dado que es un sistema con fines de trabajo.

Usabilidad:

- La herramienta estará orientada al responsable de ocupar el rol de probador, el cual debe poseer los principales conocimientos sobre el método de prueba de Caja Blanca, específicamente en la técnica del Camino Básico.
- La Interfaz es de un flujo sencillo, típico de herramientas de escritorio.
- Será útil para cualquier usuario que requiera analizar código fuente.

Rendimiento:

- Realización de operaciones complicadas en dos hilos.
- Capacidad de analizar miles de líneas de código en pocos segundos.

Soporte:

- Se requiere una documentación apropiada que describa todas las funcionalidades del sistema desarrollado así como una guía para su uso o poseerá una ayuda para el aprendizaje de la misma.

Portabilidad y software:

- Windows 2000 Professional con SP4 y Microsoft .Net Framework 2.0.
- Windows XP Professional con SP2 y Microsoft .Net Framework 2.0.
- Windows XP Home Edition con SP2 y Microsoft .Net Framework 2.0.

Seguridad:

- El sistema debe proteger la integridad de la información que se maneja.
- El sistema no debe presentar fallos y en casos de alguno debe garantizar que las pérdidas de información sean mínimas.

Políticos – culturales:

- La herramienta sólo podrá ser utilizada dentro de la UCI.
- No debe contener palabras en otros idiomas.

Hardware:

- Procesador 600 MHz o superior.
- 128 MB de memoria RAM.
- Monitor VGA o superior.
- Ratón Microsoft o compatible.
- Espacio en disco duro 30 MB

Restricciones para el diseño de la implementación:

- Utilizar los estándares de diseño establecidos.
- Para la implementación de la herramienta se deberá utilizar SharpDevelop con lenguaje de programación C#.

2.4 Modelo del Sistema. Definición de los Casos de Uso.

Una de las situaciones más difíciles a la hora de construir un sistema es precisamente saber qué construir. El modelo de Casos de Uso del sistema representa las funcionalidades deseadas y el entorno del sistema a través de actores y casos de uso. Este sienta las bases necesarias para el desarrollo del análisis y el diseño del sistema.

La identificación de los casos de uso es precisamente la guía del Ingeniero de Software que lleva adelante el desarrollo de un sistema de software.

2.4.1 Definición de los actores.

Los actores del sistema constituyen personas o sistemas que fueron trabajadores del negocio y que interactúan de alguna forma con el sistema y que están asociados al cumplimiento de los requisitos funcionales o procesos que responden a las funcionalidades definidas en los mismos.

Tabla 2.1. Descripción de los actores del sistema.

Actores	Descripción
Usuario	Es el encargado de realizarle las pruebas a un fragmento de código. Puede determinar la complejidad ciclomática y el cumplimiento de los estándares y los parámetros de codificación.

2.4.2 Listado de Casos de Uso.

A continuación en un formato bastante simple se presenta un breve resumen de todos los casos de uso que se tienen en cuenta para la comprensión de la funcionalidad del sistema, integrando los requisitos que lo conforman.

Tabla 2.2. Caso de Uso Calcular Complejidad Ciclomática por código fuente.

CU 1:	Calcular Complejidad Ciclomática por código fuente.
Actor:	Usuario
Descripción:	En este CU se determina la Complejidad Ciclomática, a partir de un fragmento de código introducido por el usuario y si él lo desea, puede exportar la matriz de dicho código.
Referencias	RF1
Prioridad	Crítico

Tabla 2.3. Caso de Uso Calcular Complejidad Ciclomática por matriz.

CU 2:	Calcular Complejidad Ciclomática por matriz.
Actor:	Usuario
Descripción:	En este CU se va a determinar la Complejidad Ciclomática, a partir de la matriz

	importada por el usuario.
Referencias	RF2
Prioridad	Crítico

Tabla 2.4. Caso de Uso Determinar Conjunto Básico de Caminos Independientes.

CU 3:	Determinar Conjunto Básico de Caminos Independientes.
Actor:	Usuario
Descripción:	En este CU se va a determinar el conjunto básico de Caminos Linealmente Independientes, a partir de un fragmento de código o al importarse una matriz por el usuario.
Referencias	RF3
Prioridad	Crítico

Tabla 2.5. Caso de Uso Importar Matriz.

CU 4:	Importar Matriz.
Actor:	Usuario
Descripción:	En este CU el usuario podrá importar una matriz que esté guardada en la computadora.
Referencias	RF4
Prioridad	Crítico

Tabla 2.6. Caso de Uso Gestionar Configuración de Estándares de Codificación.

CU 5:	Gestionar Configuración de Estándares de Codificación.
Actor:	Usuario
Descripción:	En este CU el usuario puede modificar la configuración de estándares que se abre predefinidamente si la misma no le resulta conveniente. Podrá exportarla como otra configuración o realizar los cambios y guardarla sobre la misma.
Referencias	RF5 (RF5.1, RF5.2, RF5.3, RF5.4)
Prioridad	Crítico

Tabla 2.7. Caso de Uso Comprobar Configuración de Estándares de Codificación.

CU 6:	Comprobar Configuración de Estándares de Codificación.
Actor:	Usuario
Descripción:	En este CU se comprueba el cumplimiento de los estándares de codificación que se definieron, con el fragmento de código que el usuario selecciona.
Referencias	RF6
Prioridad	Crítico

Tabla 2.8. Caso de Uso Generar Reportes de parámetros de código fuente.

CU 7:	Generar Reportes de parámetros de código fuente.
Actor:	Usuario
Descripción:	En este CU se mostrarán reportes con los parámetros de codificación que posee el fragmento de código fuente seleccionado por el usuario.
Referencias	RF7

Prioridad	Crítico

Tabla 2.9. Caso de Uso Gestionar Reportes de estándares.

CU 8:	Gestionar Reportes de Estándares
Actor:	Usuario
Descripción:	En este CU el usuario guarda en la computadora los datos de la comprobación de estándares, además de mostrarlos e imprimirlos según estime conveniente.
Referencias	RF8 (RF8.1, RF8.2, RF8.3)
Prioridad	Medio

Tabla 2.10. Caso de Uso Gestionar Reportes complejidad ciclomática.

CU 9:	Gestionar Reportes de Complejidad Ciclomática.
Actor:	Usuario
Descripción:	En este CU el usuario guarda en la computadora el valor de la complejidad ciclomática calculado a partir del código fuente o por la matriz importada, además de mostrarlos e imprimirlos según estime conveniente.
Referencias	RF8 (RF8.1, RF8.2, RF8.3)
Prioridad	Medio

2.4.3 Diagrama de Caso de Uso del Sistema.

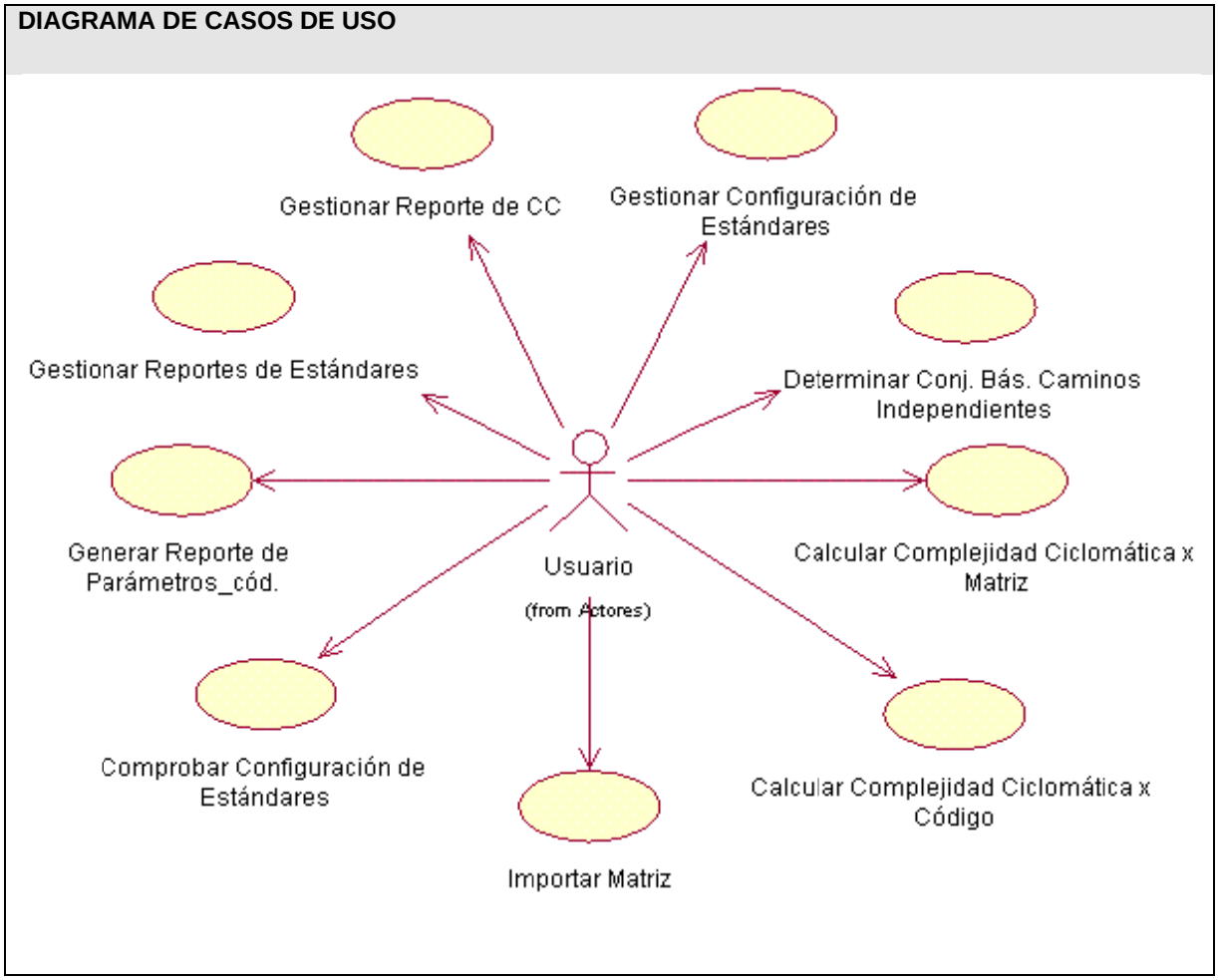


Figura2.1: Diagrama de Casos de Uso.

2.4.4 Casos de Uso expandidos.

Para comprender la funcionalidad que engloba cada caso de uso, no basta con la representación gráfica del diagrama de casos de uso. Debe ser abordada en una descripción con formato expandido que abarque las principales características de los mismos. El formato de descripción que se propone se expondrá en el Anexo 1 con los restantes casos de uso, aglomerándose los elementos más importantes que detallan el flujo de eventos de los caso de uso, solo se expondrán a continuación dos de las descripciones para que se posea una idea. Los casos de uso son: Calcular complejidad ciclomática por código fuente y Gestionar estándares de codificación.

Tabla 2.11: Diagrama de Casos de Uso: Calcular complejidad ciclomática por código.

Caso de Uso #1:	Calcular Complejidad Ciclomática por código fuente.
Actores:	Usuario.
Resumen:	En este CU se determina la Complejidad Ciclomática, a partir de un fragmento de código introducido por el usuario y si él lo desea, puede exportar la matriz de dicho código.
Precondiciones:	El código fuente debe de estar previamente procesado por un compilador.
Referencias	RF1(RF1.1)
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- El usuario en las opciones del menú o mediante la barra de herramientas, selecciona la opción: “Complejidad Ciclomática.”	1.1- La aplicación visualiza una interfaz con el código que el usuario seleccionó, además de la opción de analizar todas las funcionalidades de dicho código.
2- El usuario selecciona el botón “Analizar Todas”.	2.1- La aplicación muestra el valor de la complejidad ciclomática de todas las funciones del código. 2.2- Si el usuario desea exportar la matriz del código fuente seleccionada, ver Sección: “Exportar Matriz”.

Poscondiciones:	Se muestra el valor de la complejidad ciclomática de cada función del fragmento seleccionado.

Flujo Normal de Eventos

Sección "Exportar Matriz"	
Acción del Actor	**Respuesta del Sistema**
3- El usuario selecciona la opción "Exportar matriz", mediante un clic derecho encima de la función o por la opción del menú.	3.1- La aplicación visualiza una interfaz para que se guarde la matriz.
Poscondiciones:	Se guarda la matriz del fragmento de código que el usuario seleccionó.
Prototipo de Interfaz	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

Tabla 2.12: Diagrama de Casos de Uso: Gestionar configuración de estándares de codificación.

Caso de Uso #5:	Gestionar configuración de estándares de codificación.
Actores:	Usuario
Resumen:	En este CU el usuario puede modificar la configuración de estándares que se abre predefinidamente si la misma no le resulta conveniente. Podrá exportarla como otra configuración o realizar los cambios y guardarla sobre la misma.
Precondiciones:	Que exista la configuración de estándares de código fuente predefinida.
Referencias	RF5 (RF5.1, RF5.2, RF5.3)
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- El usuario en las opciones del menú selecciona la	1.1- El sistema visualiza una interfaz con las

opción: “Configuración” y en ella elige: “Estándares de Codificación”. 2- El usuario escoge una de las siguientes opciones: Selecciona el botón “Abrir configuración de estándares existente”. Ver Sección “Abrir”. Selecciona el botón “Guardar configuración de estándares como”. Ver Sección “Guardar”. Selecciona el botón “Mostrar la configuración de estándares”. Ver Sección “Mostrar”.	validaciones de los estándares de codificación predefinidas y las opciones: “Abrir configuración existente”, “Guardar configuración de estándares como” y “Mostrar la configuración de estándares”.
Flujo Normal de Eventos	
Sección “Abrir”	
Acción del Actor	**Respuesta del Sistema**
3- El usuario elige la configuración y da clic en el botón “Aceptar”. 4- Si el usuario desea realizarle algunos cambios a la configuración de estándares, ver Sección “Modificar”.	2.1- El sistema visualiza una ventana para buscar la configuración de estándares que desea. 3.1- El sistema visualiza la configuración escogida y sus validaciones.

Poscondiciones	Se abre una nueva configuración de estándares de codificación.
Flujo Normal de Eventos	
Sección "Guardar"	
Acción del Actor	**Respuesta del Sistema**
3- El usuario realiza los cambios (Ver Sección: 'Modificar') y le pone el nombre con el que desea que se guarde la configuración y da clic en el botón "Aceptar".	2.1- El sistema visualiza una ventana para que el usuario escoja el lugar donde quiere almacenar la configuración de estándares. 3.1- El sistema guarda la nueva configuración de estándares con las validaciones pertinentes en cada pestaña.
Poscondiciones	Se guarda una nueva configuración de estándares.
Flujo Normal de Eventos	
Sección "Mostrar"	
Acción del Actor	**Respuesta del Negocio**
3- El usuario presiona el botón "Cerrar" y si desea modificar algunas validaciones de la configuración, ver Sección "Modificar".	2.1- El sistema visualiza en una ventana la última configuración de estándares que se guardó en forma de texto y los botones: "Modificar" y "Cerrar".
Poscondiciones	Se visualizan las validaciones de la última configuración de estándares en forma de texto.
Flujo Normal de Eventos	

Sección "Modificar"	
Acción del Actor	**Respuesta del Negocio**
2- El usuario cuando está en la pantalla de las validaciones de la configuración de estándares realiza los cambios que desea en cada pestaña y presiona el botón "Aplicar" o en la pantalla "Detalles de la Configuración" modifica según determine, y presiona el botón "Modificar".	2.1- El sistema modifica automáticamente las validaciones de los estándares.
Poscondiciones	Se modifican las validaciones que el usuario estimó de los estándares de codificación.
Flujos Alternos	
Acción del Actor	**Respuesta del Negocio**

2.5 Prototipos de la Interfaz de Usuario:

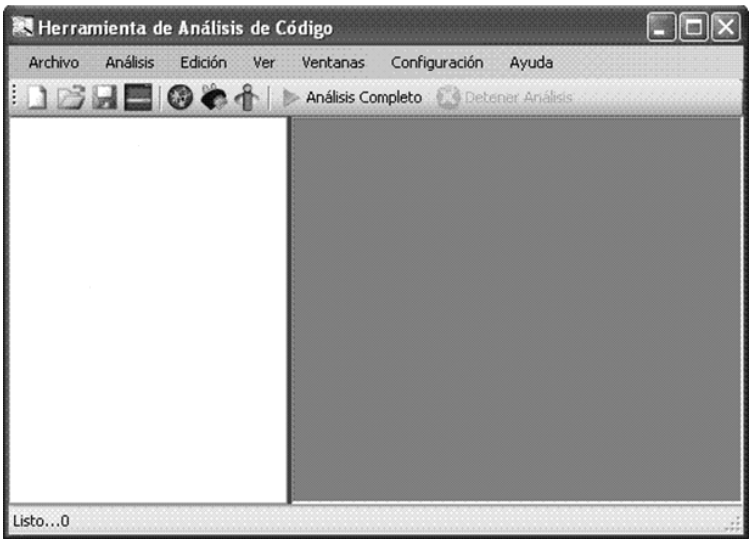


Figura 2.2: Interfaz principal.

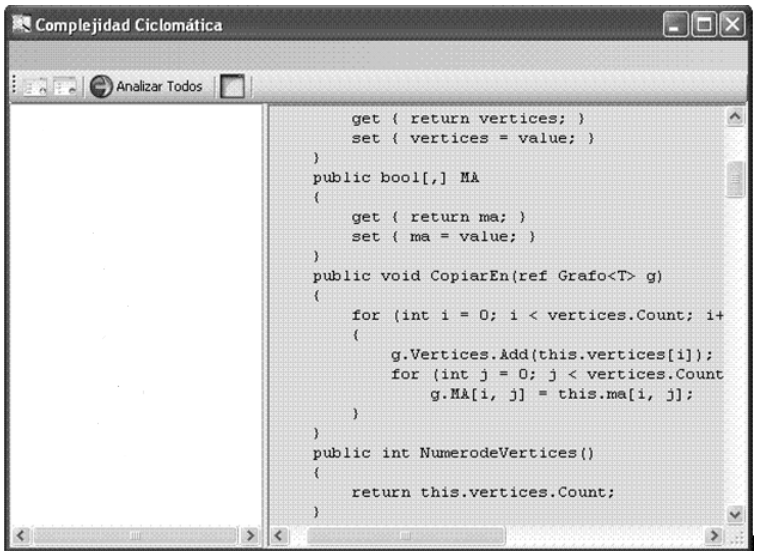


Figura 2.3: Interfaz de la Complejidad Ciclomática.

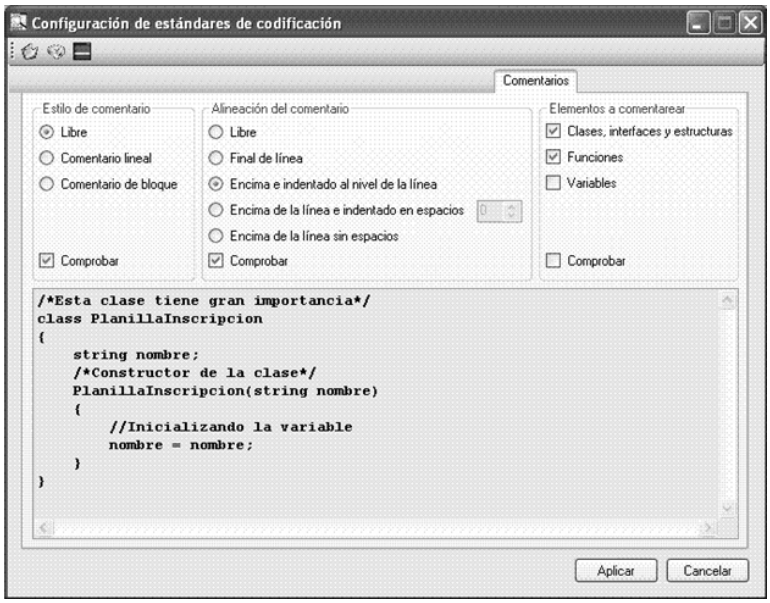


Figura 2.4: Interfaz de la Configuración de estándares de codificación.

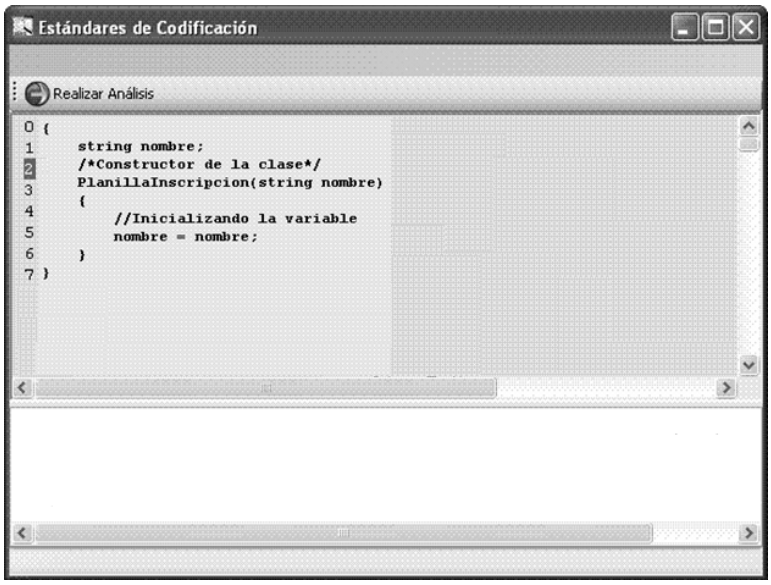


Figura 2.5: Interfaz de Comprobación de estándares de codificación.

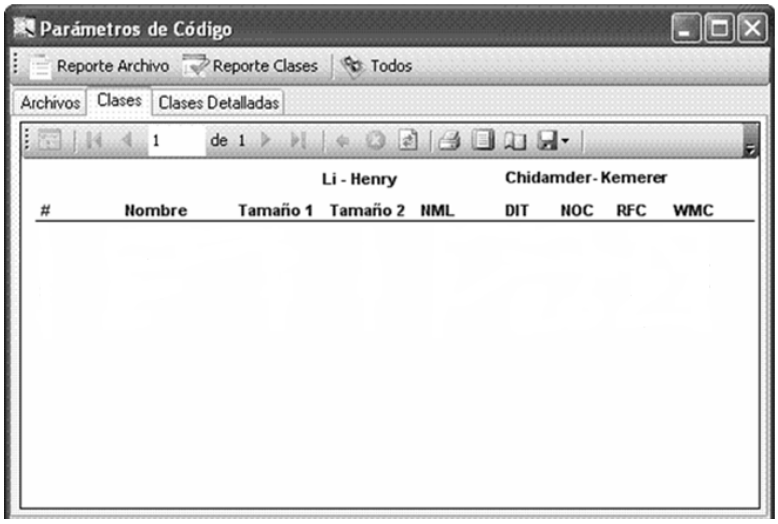


Figura 2.6: Interfaz de Parámetros de código fuente.

Conclusiones

En este capítulo se han abordado aspectos relacionados con la propuesta de una herramienta que utilice parte del método de pruebas de Caja Blanca, específicamente la técnica del Camino Básico. Con el objetivo de automatizar el proceso de pruebas de software y optimizar el proceso de revisiones,

debido a la necesidad de realizar de diferentes pruebas a los productos para garantizarle una mayor calidad.

Se realizó la modelación del dominio, a fin de lograr una mejor comprensión de los procesos del mismo a pesar de ser un tanto abstracto de comprender, para que los usuarios finales y desarrolladores posean un entendimiento común. Así como los conceptos fundamentales a tener en cuenta en dicho proceso. Se identificaron y describieron los trabajadores del negocio, además, se obtuvo el modelo de objetos del negocio.

Se identificaron los requerimientos del sistema, obteniéndose así sus funcionalidades y un listado de requerimientos no funcionales a tener en cuenta para el desarrollo de la herramienta. Se definieron los casos de uso del sistema, mostrándose el diagrama de casos de uso. Además, se describieron cada uno de estos, así como los actores del sistema que se van a relacionar con los mismos, lo que permite obtener una vista global de la concepción y funcionamiento de la herramienta.

CAPÍTULO 3: ANÁLISIS Y DISEÑO DEL SISTEMA.

En este capítulo se desarrolla el flujo de análisis y diseño, es uno de los flujos de trabajo del RUP, metodología empleada en la modelación del sistema. Aunque esta metodología une al Análisis y Diseño en un solo flujo de trabajo ambos tienen funcionalidades diferentes. Estos dos procesos son muy importantes en el desarrollo de software dirigido por modelos ya que constituyen la vista lógica de la arquitectura del software.

Se realiza el análisis del sistema, modelando su principal diagrama (Diagrama de Análisis) y el diagrama de clases del diseño que es uno de los artefactos principales del flujo de trabajo del RUP. Finalmente después de modelar la lógica del negocio a través de las clases, se tratan los principios del diseño de la aplicación, donde se desarrolla el Diagrama de Clases de Diseño, el Modelo de Clases Persistente y el Modelo de Despliegue.

3.1 Flujos de Trabajo de Análisis y Diseño.

El objetivo principal de este flujo de trabajo es transformar los requerimientos a una especificación que describa cómo implementar el sistema. El análisis, fundamentalmente consiste en obtener una visión que se preocupa de ver qué hará el sistema de software a desarrollar, por tal motivo este se interesa en los requerimientos funcionales. Por otro lado, el diseño es un refinamiento que tiene en cuenta los requerimientos no funcionales, por lo cual se centra en cómo el sistema cumple sus objetivos.

Los objetivos específicos del análisis y diseño son:

- Transformar los requerimientos al diseño del futuro sistema.
- Desarrollar una arquitectura para el sistema.
- Adaptar el diseño para que sea consistente con el entorno de implementación.

3.2 Modelo de Análisis.

El modelo de análisis se centra en el tratamiento de requisitos funcionales y pospone los no funcionales para el diseño. Esboza cómo llevar a cabo la funcionalidad dentro del sistema, incluidas las funcionalidades significativas para la arquitectura, sirve como una primera aproximación del diseño y es el resultado de la actividad de analizar los casos de uso.

En la construcción del modelo de análisis se tienen que identificar las clases que describen la realización de los casos de uso, los atributos y las relaciones entre ellas. Con esta información se

construye el diagrama de clases del análisis, estas representan los conceptos en un dominio del problema, sin llegar al nivel de detalle necesario que se persigue en el diseño, constituyendo una primera aproximación de este.

3.2.1 Diagrama de clases del análisis.

Los diagramas de clases del análisis representan la relación entre las clases, las cuales se centran en los requisitos funcionales, tienen atributos y entre ellas se establecen relaciones de asociación, agregación / composición, generalización / especialización y tipos asociativos. RUP propone clasificar a las clases en: interfaz, controladora y entidad.

- Las clases Interfaz modelan la interacción entre el sistema y sus actores.
- Las clases Controladoras coordinan la realización de uno o unos pocos casos de uso coordinando las actividades de los objetos que implementan la funcionalidad del caso de uso.
- Las clases Entidad modelan información que posee larga vida y que es a menudo persistente.

Se presentarán los diagramas de clases del análisis de los casos de uso: Calcular Complejidad Ciclomática por código fuente y Gestionar Estándares de Codificación. En el Anexo 2 se podrán encontrar otros de los restantes diagramas de clases de análisis.

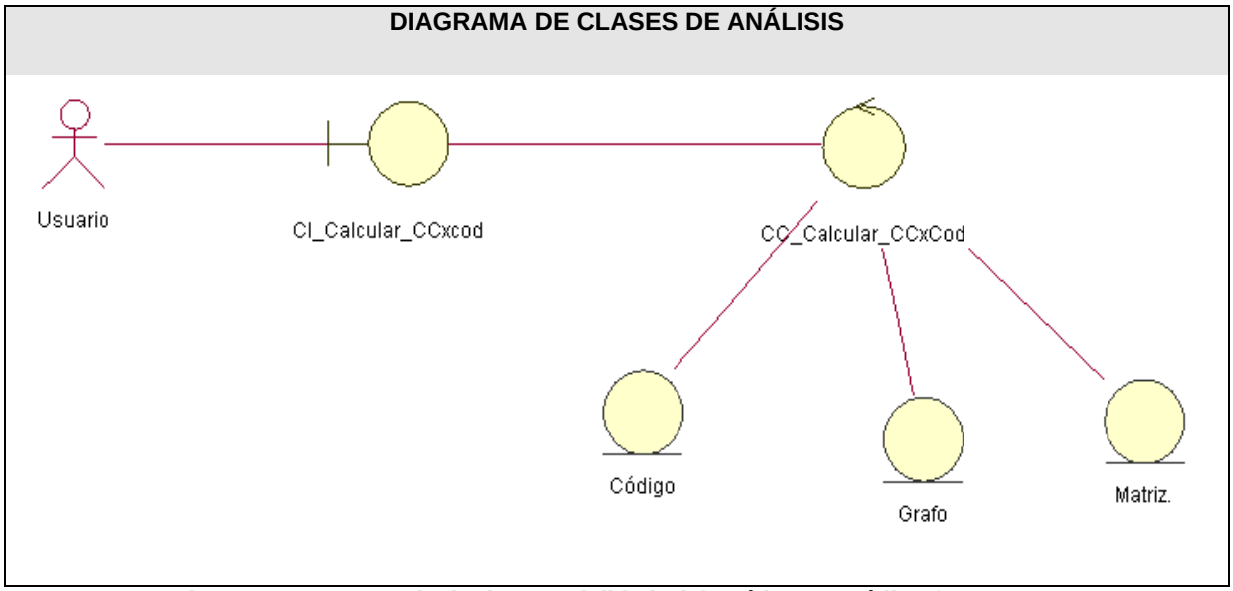


Figura 3. DCA_CU Calcular la Complejidad Ciclomática por código fuente.

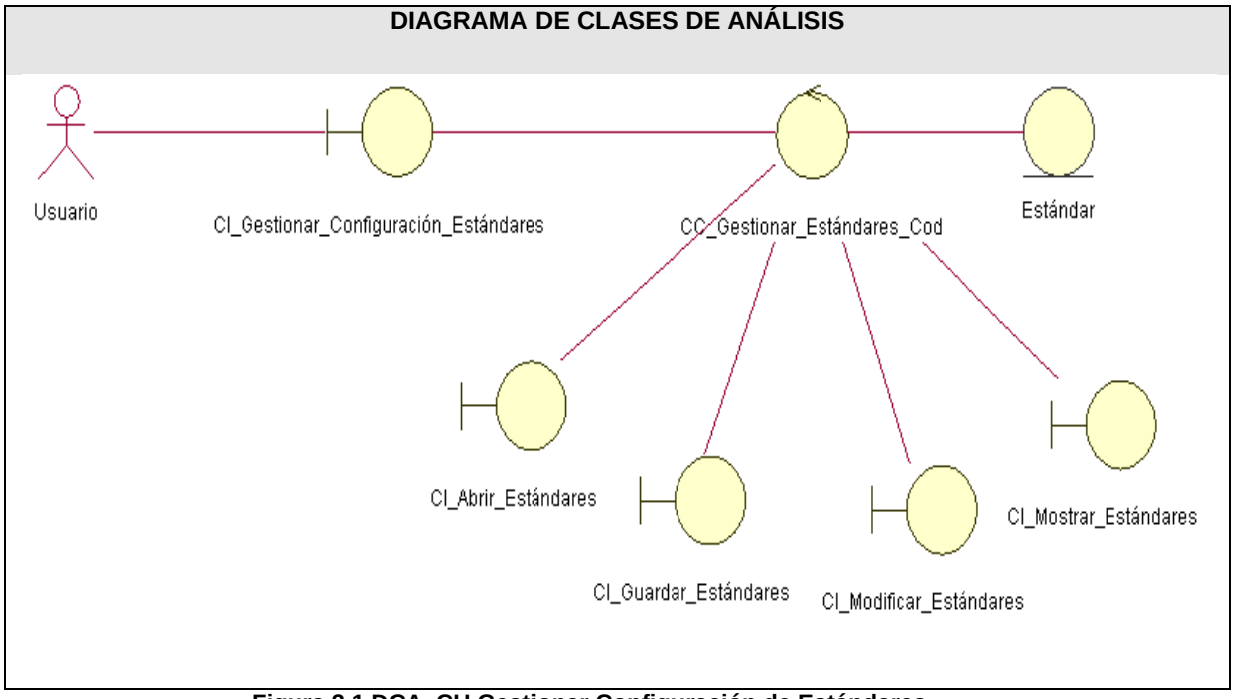


Figura 3.1 DCA_CU Gestionar Configuración de Estándares.

3.2.2 Diagrama de Interacción (Colaboración).

Se presentarán los diagramas de interacción de los casos de uso anteriormente mencionados, divididos por escenarios en el último caso de uso: Gestionar Configuración de Estándares, lo cual permitirá una mejor comprensión de los mismos. En el Anexo 3 se podrán encontrar otros de los restantes diagramas de interacción.

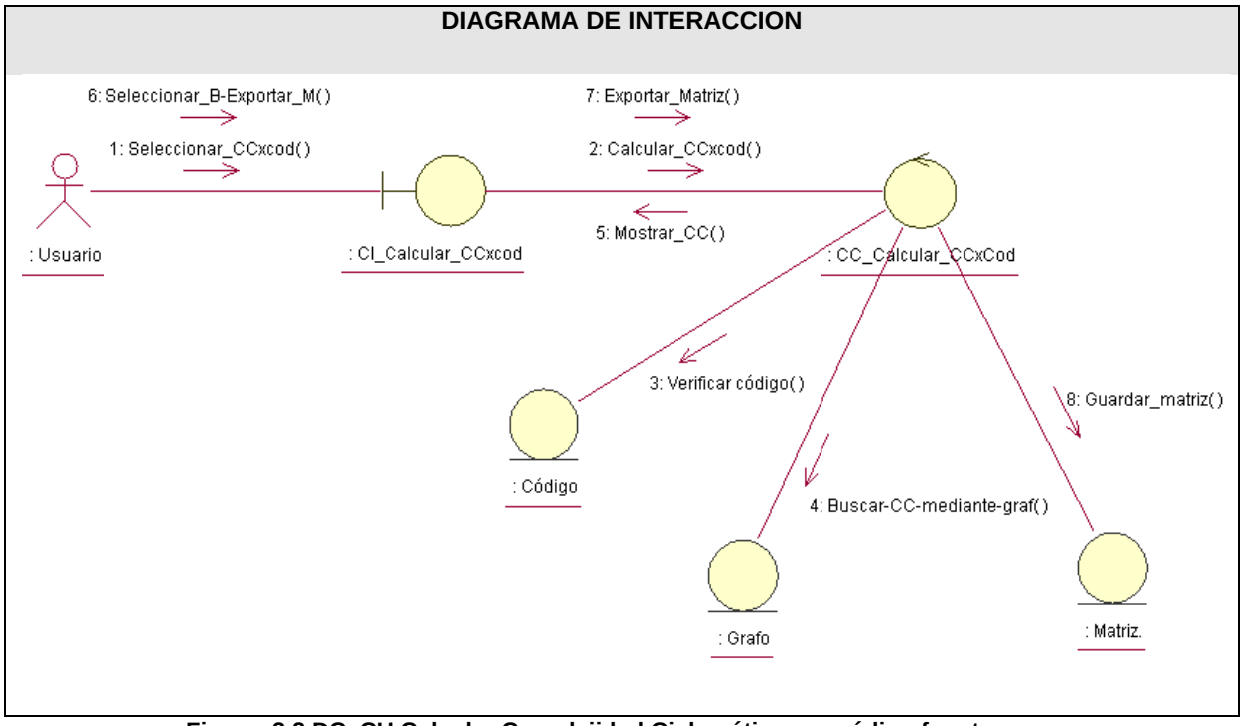


Figura. 3.2 DC_CU Calcular Complejidad Ciclomática por código fuente.

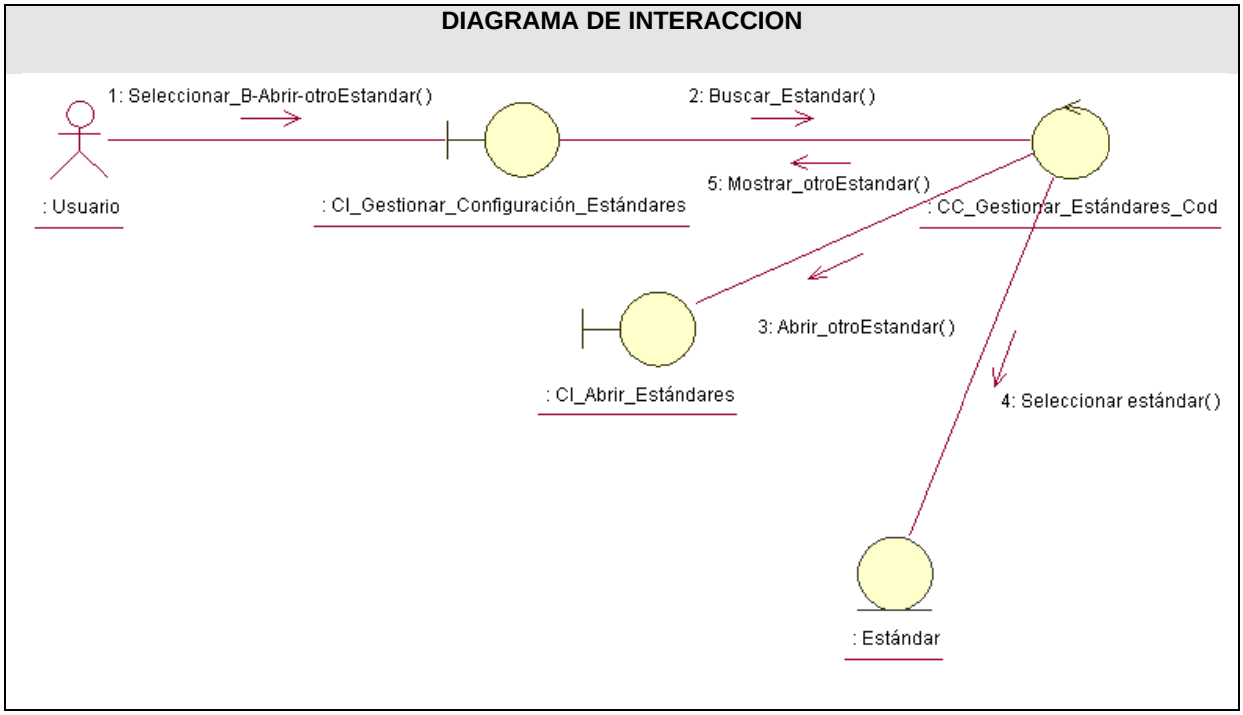


Figura. 3.3 DC_CU Gestionar Configuración de Estándares (Escenario _ abrir)

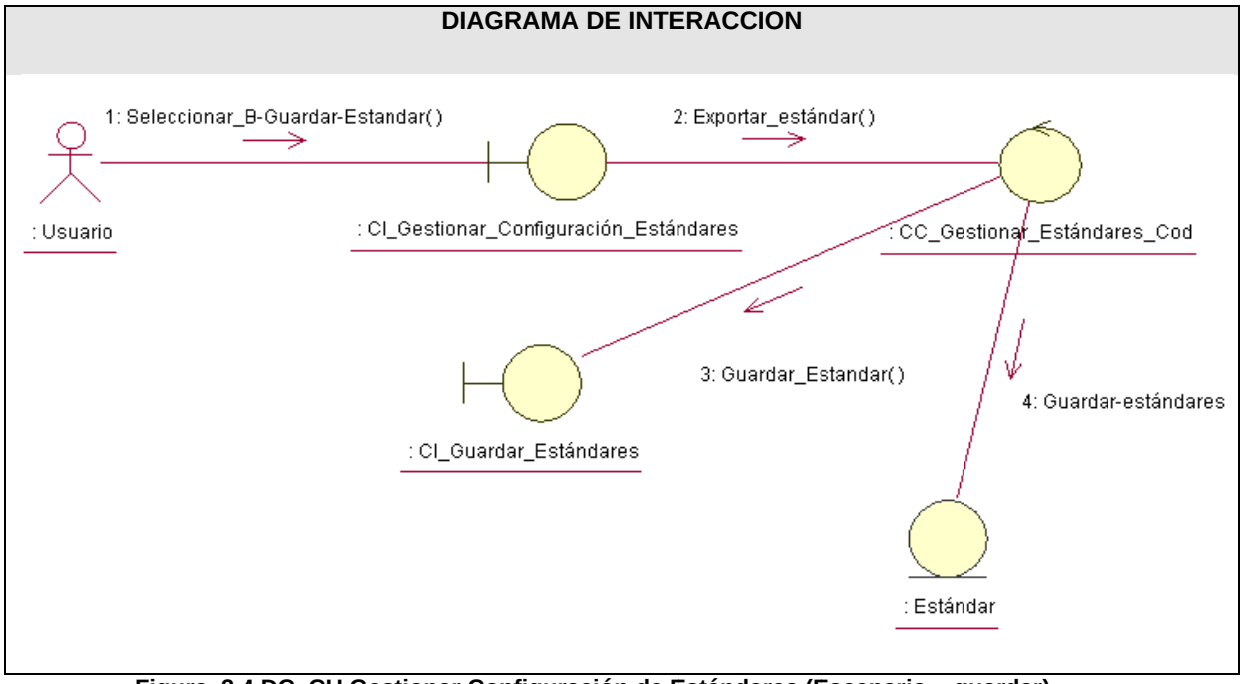


Figura. 3.4 DC_CU Gestionar Configuración de Estándares (Escenario _ guardar)

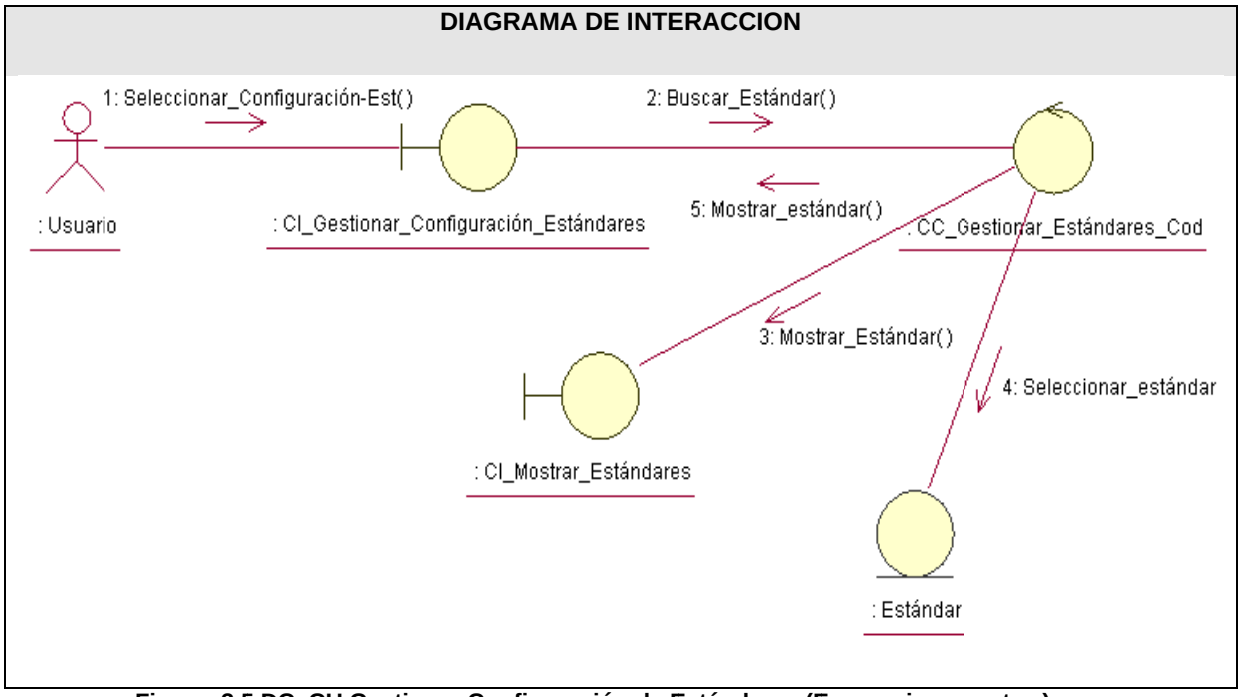


Figura. 3.5 DC_CU Gestionar Configuración de Estándares (Escenario _ mostrar)

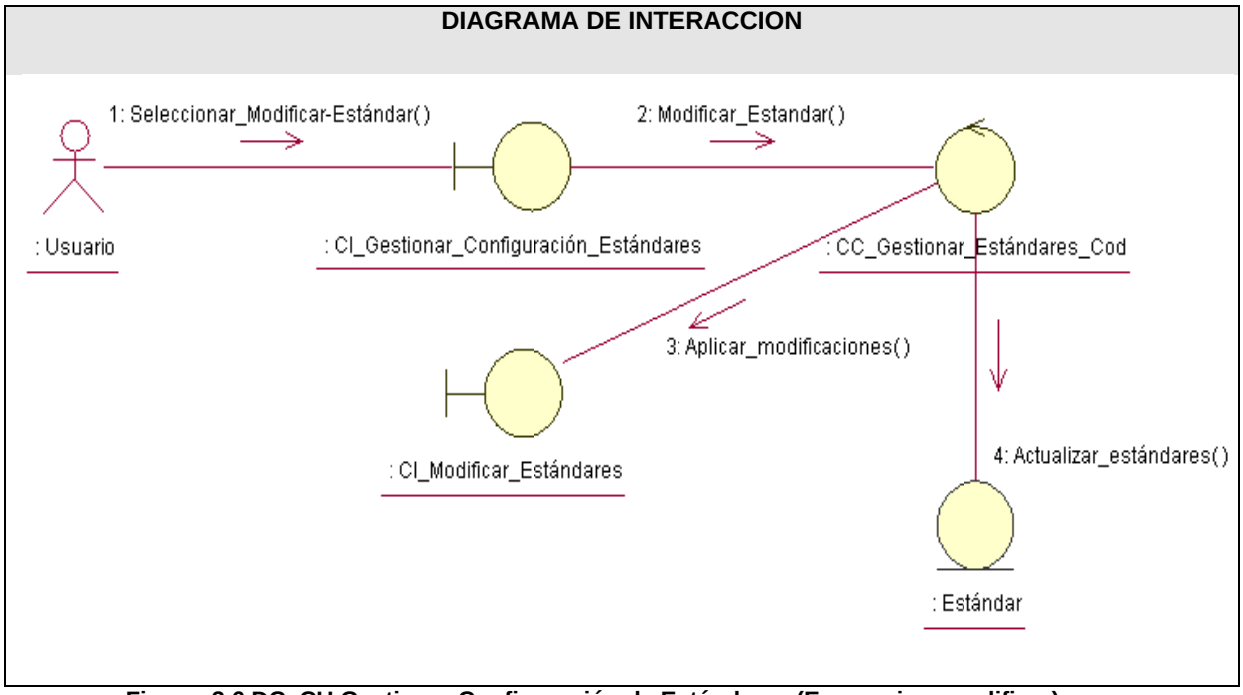


Figura. 3.6 DC_CU Gestionar Configuración de Estándares (Escenario _ modificar)

3.3 Modelo de Diseño.

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso. Se centra en cómo los requisitos funcionales y no funcionales tienen impacto en el sistema a desarrollar. Dentro de sus propósitos están: crear una entrada apropiada y un punto de partida para la implementación, descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo y adquirir una comprensión de los aspectos relacionados con los requisitos no funcionales y restricciones vinculadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia.

La realización de caso de uso del diseño contiene una descripción de flujos de eventos textual, diagramas de clases y diagramas de interacción. Los diagramas de clases son los artefactos más utilizados en el modelado de sistemas orientados a objetos. Un diagrama de clases representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas. Las clases de diseño se especifican utilizando la sintaxis del lenguaje de programación elegido.

3.3.1 Principios de Diseño.

Interfaz de usuario.

La interfaz diseñada para la herramienta está basada en el estándar de ventanas. El tipo de letra a utilizar será Tahoma de estilo regular y tamaño 8.25 para el texto de los botones en la barra de herramientas y Microsoft Sans Serif de 8.25 para el resto de las etiquetas y mensajes de los formularios y el diseño de la aplicación deber ser adecuado para el tipo de usuario, (que en la mayoría de los casos representará el rol de probador) logrando la fácil comprensión del mismo en el lenguaje utilizado para las opciones que brindan. El sistema debe mostrar una barra de menú en la parte superior donde deben estar la mayoría de las opciones de trabajo.

Ayuda.

Una parte importante de cualquier sistema lo constituye la ayuda, en este caso en la barra de menú aparecerá una opción de ayuda con temas relacionado a la utilización de la herramienta con el objetivo de aclarar alguna duda al usuario. Además de temáticas referentes a las pruebas de Caja Blanca, también debe presentar la posibilidad de acceder a la misma con solo presionar la tecla F1.

Tratamiento de errores.

En el diseño de la interfaz se debe tener en cuenta el tratamiento de errores logrando que los mensajes de error que emita el sistema sean de fácil comprensión para el usuario y lo más descriptivos posibles y además debe alertarlos de posibles riesgos de las operaciones que realice.

3.3.2 Patrones.

Un patrón es una pareja de problema / solución con un nombre, que codifica (estandariza) buenos principios y sugerencias. Es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en diferentes situaciones. El objetivo de los patrones es crear un lenguaje común a una comunidad de desarrolladores para comunicar experiencia sobre los problemas y sus soluciones. Pueden referirse a distintos niveles de abstracción, desde un proceso de desarrollo hasta la utilización eficiente de un lenguaje de programación. [29]

Fundamentación del uso de patrones.

La calidad del diseño de la interacción entre los objetos y la asignación de responsabilidades presentan gran variación, decisiones poco acertadas, dan origen a sistemas y componentes frágiles y

difíciles de mantener, entender, reutilizar o extender. Los diseñadores expertos en orientación a objetos van formando un amplio repertorio de principios generales y de expresiones que los guían al crear software, a algunos se les asigna el nombre de patrones, estos se codifican en un formato estructurado que describe el problema y su solución.

Actualmente el uso de los patrones de diseño se ha generalizado, son utilizados y están presentes en casi todos los sistemas informáticos de importancia a nivel mundial.

Un patrón de diseño es una abstracción de una solución en un nivel alto que solucionan problemas que existen en muchos niveles de abstracción.

"Para llevar a cabo un buen diseño se han definido una serie de patrones. Los patrones de diseño de software constituyen un conjunto de principios generales y expresiones que ayudan a desarrollar software." [30]

"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." [31]

Dentro de este grupo se identifican 5 patrones fundamentales:

Experto: Indica que la responsabilidad de la creación de un objeto siempre se debe asignar al experto en información, es decir, la responsabilidad recae sobre la clase que conoce toda la información necesaria para poder crearlo.

Creador: La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. En consecuencia, conviene contar con un principio general para asignar las responsabilidades concernientes a ella. Identifica quién 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."* [32]

Bajo acoplamiento: Asignar la menor cantidad de responsabilidades a cada clase y así evitar que esta recurra a muchas otras. Las clases más independientes, reduce el impacto al cambio y más reutilizables.

Controlador: "*Evita el acceso directo a las clases entidades, evita que la capa de presentación no maneje los eventos del sistema".* [33]

En los diagramas de clases que se elaboraron se aplican dichos patrones, se utilizan a fin de distribuir responsabilidades en las mismas, y establecer sus relaciones, tratando de que no estén muy sobrecargadas de funcionalidades ni exista mucha dependencia entre ellas.

Los Patrones de Diseño posibilitan la reutilización del conocimiento, su utilización como guía para el desarrollo de software, garantiza un producto robusto y acabado.

3.3.3 Diagrama de clases del diseño.

En el diagrama de clases de diseño se muestran los atributos y métodos de cada clase y se representa de una forma sencilla la colaboración y las responsabilidades de las distintas clases que forman el sistema.

Los diagramas de clases se utilizan para modelar la vista de diseño estática de un sistema, esto incluye modelar el vocabulario del sistema, modelar las colaboraciones o modelar esquemas. Los diagramas de clases también son la base para un par de diagramas relacionados: los diagramas de componentes y los diagramas de despliegue. Los diagramas de clases son importantes no sólo para visualizar, especificar y documentar modelos estructurales, sino también para construir sistemas ejecutables, aplicando ingeniería directa e inversa.

Se presentarán los diagramas de clases del diseño de los casos de uso: Calcular Complejidad Ciclomática por código y Gestionar Estándares de Codificación. En el Anexo 4 podrán encontrarse otros de los restantes diagramas de clases del diseño.

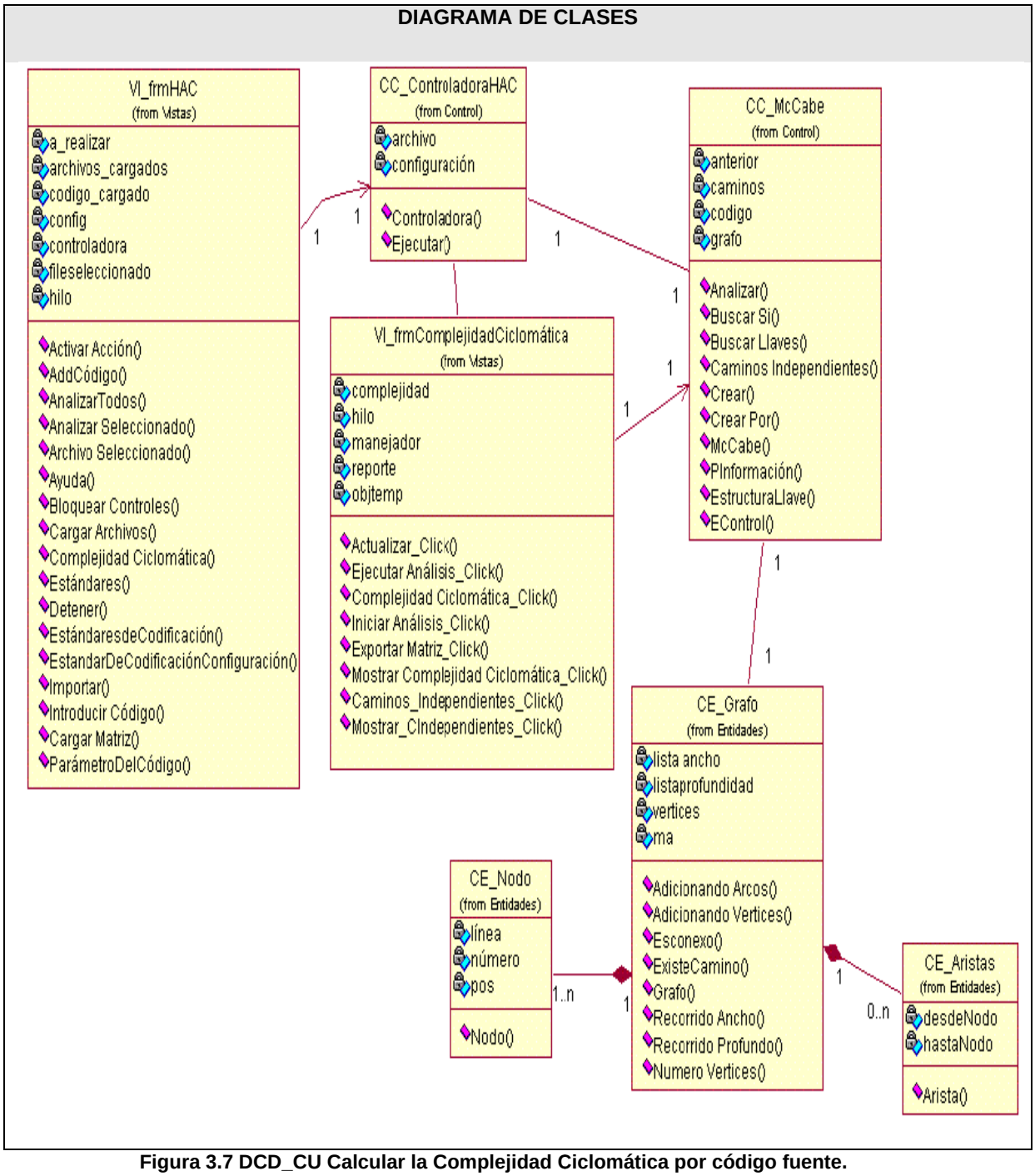


Figura 3.7 DCD_CU Calcular la Complejidad Ciclomática por código fuente.

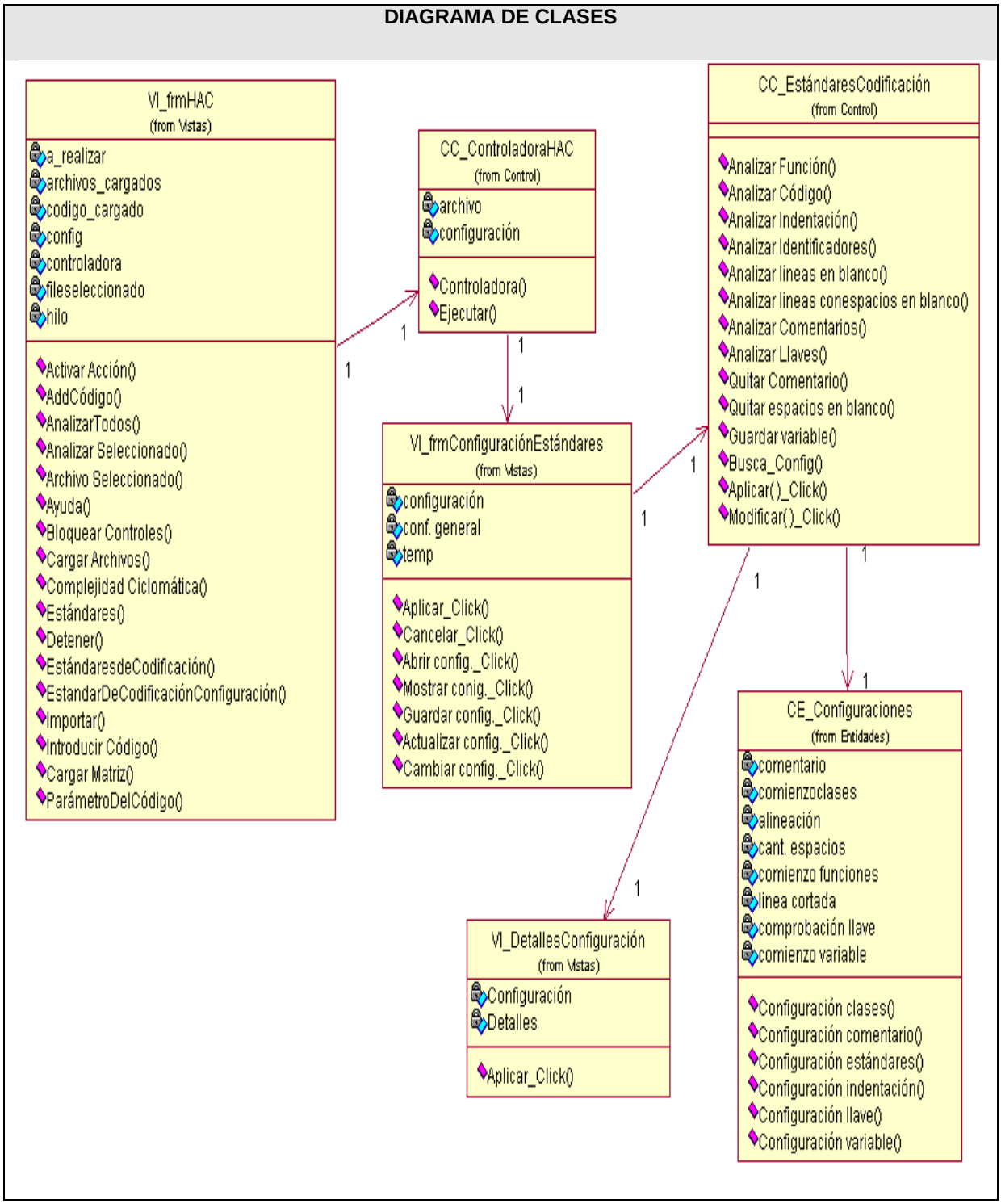


Figura 3.8 DCD_CU Gestionar Configuración de Estándares.

3.3.4 Diagrama de interacciones (Secuencia).

Los Diagramas de Interacción se utilizan para modelar los aspectos dinámicos de un sistema. La mayoría de las veces, esto implica modelar instancias concretas o prototípicas de clases, interfaces, componentes y nodos, junto con los mensajes enviados entre ellos, todo en el contexto de un escenario que ilustra un 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. [34]

Estos diagramas son de dos tipos Diagrama de Secuencia o Colaboración.

Un Diagrama de Secuencia representa las interacciones entre objetos ordenadas en secuencia temporal. Los clientes entienden fácilmente este tipo de diagramas. Contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes pasados entre los objetos. En particular muestra los objetos participantes en la interacción y la secuencia de mensajes intercambiados. [35]

Se presentarán los diagramas de interacción del caso de uso: Calcular Complejidad Ciclomática por código y Gestionar Estándares de Codificación, dividido por escenarios en el caso de uso último, para lograr una mejor comprensión del mismo. En el Anexo 5 podrán encontrarse otros de los restantes diagramas de interacción.

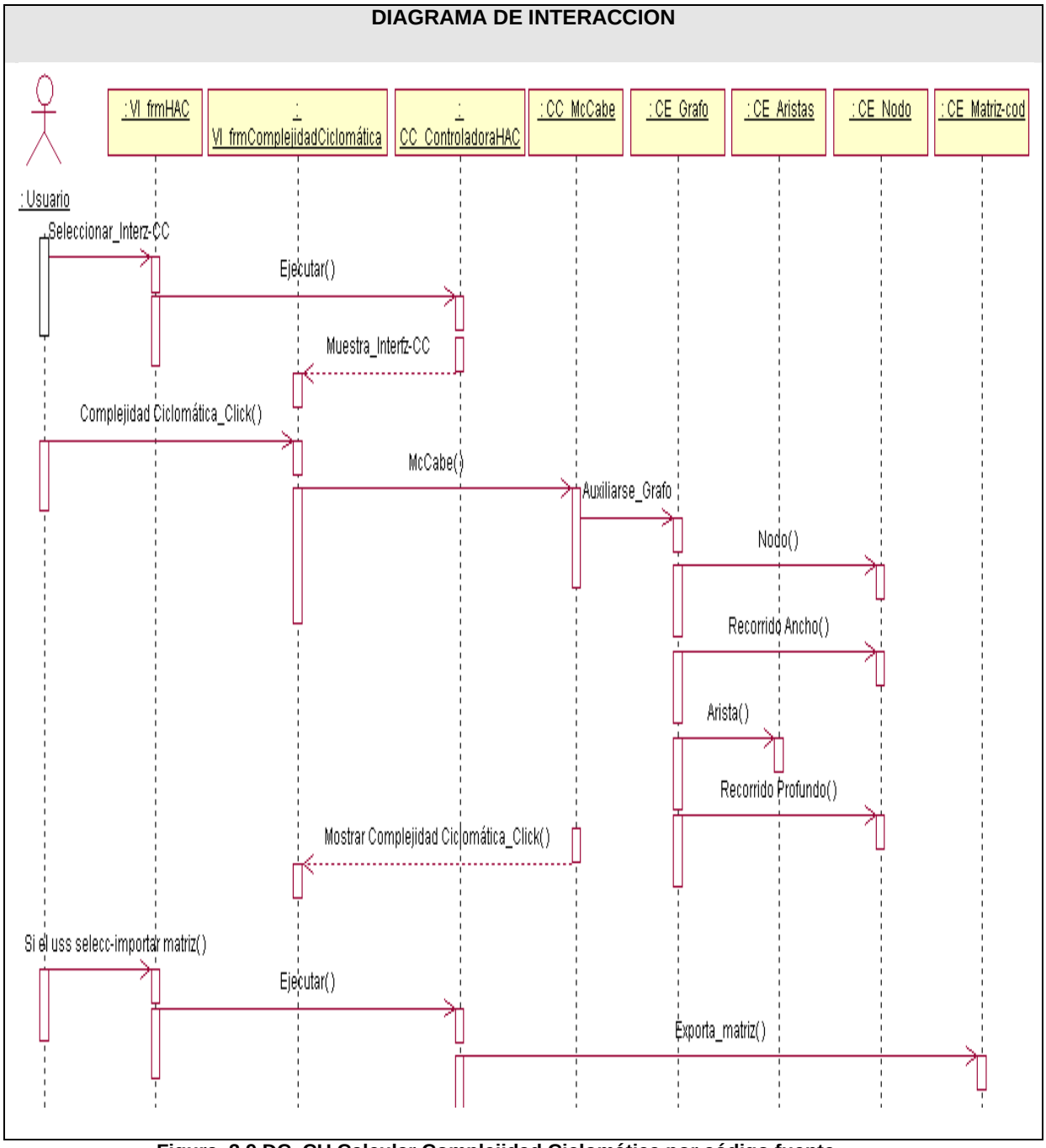


Figura. 3.9 DC_CU Calcular Complejidad Ciclomática por código fuente.

DIAGRAMA DE INTERACCION

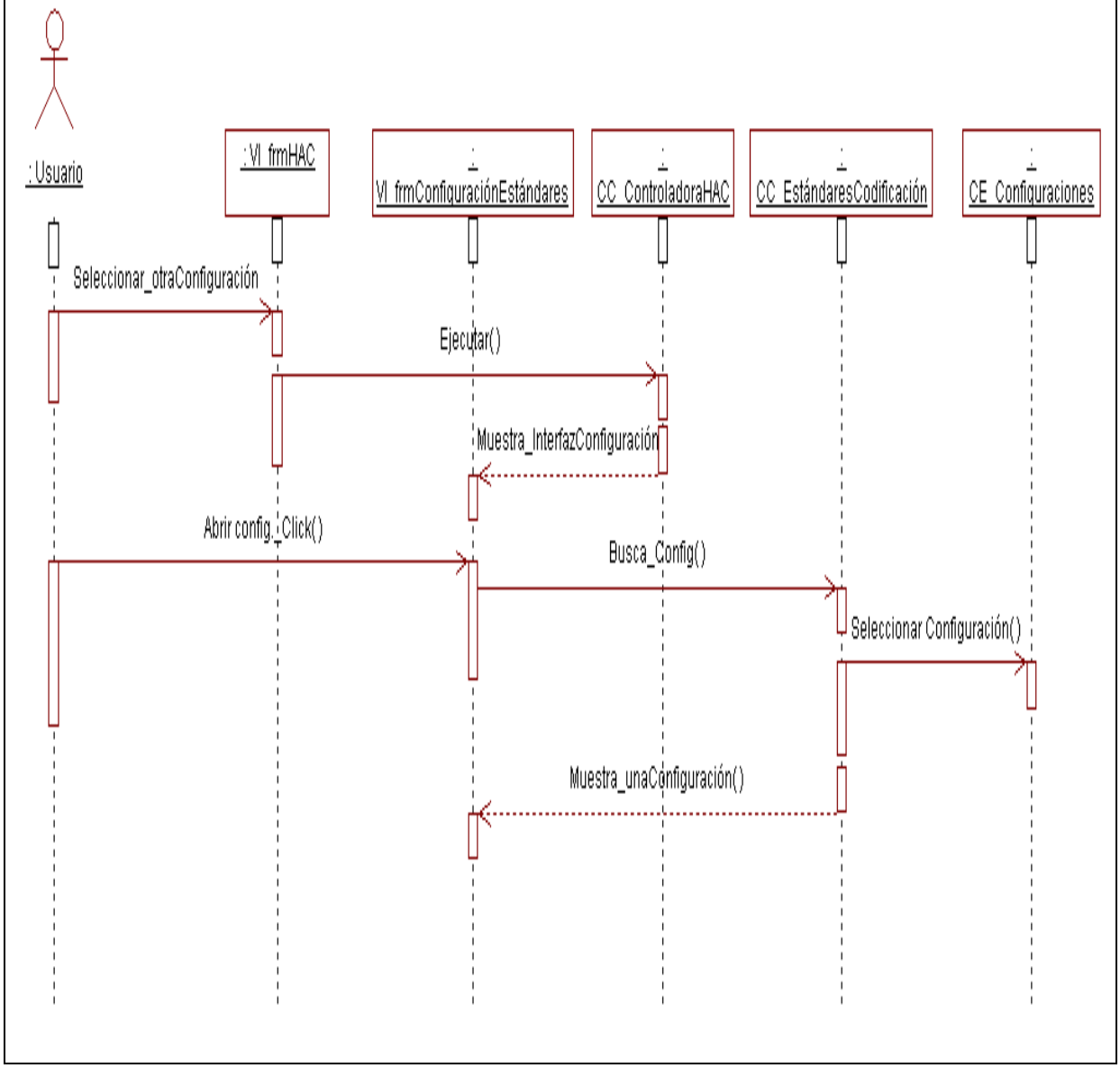


Figura. 3.10 DC_CU Gestionar Configuración de Estándares (Escenario _ abrir)

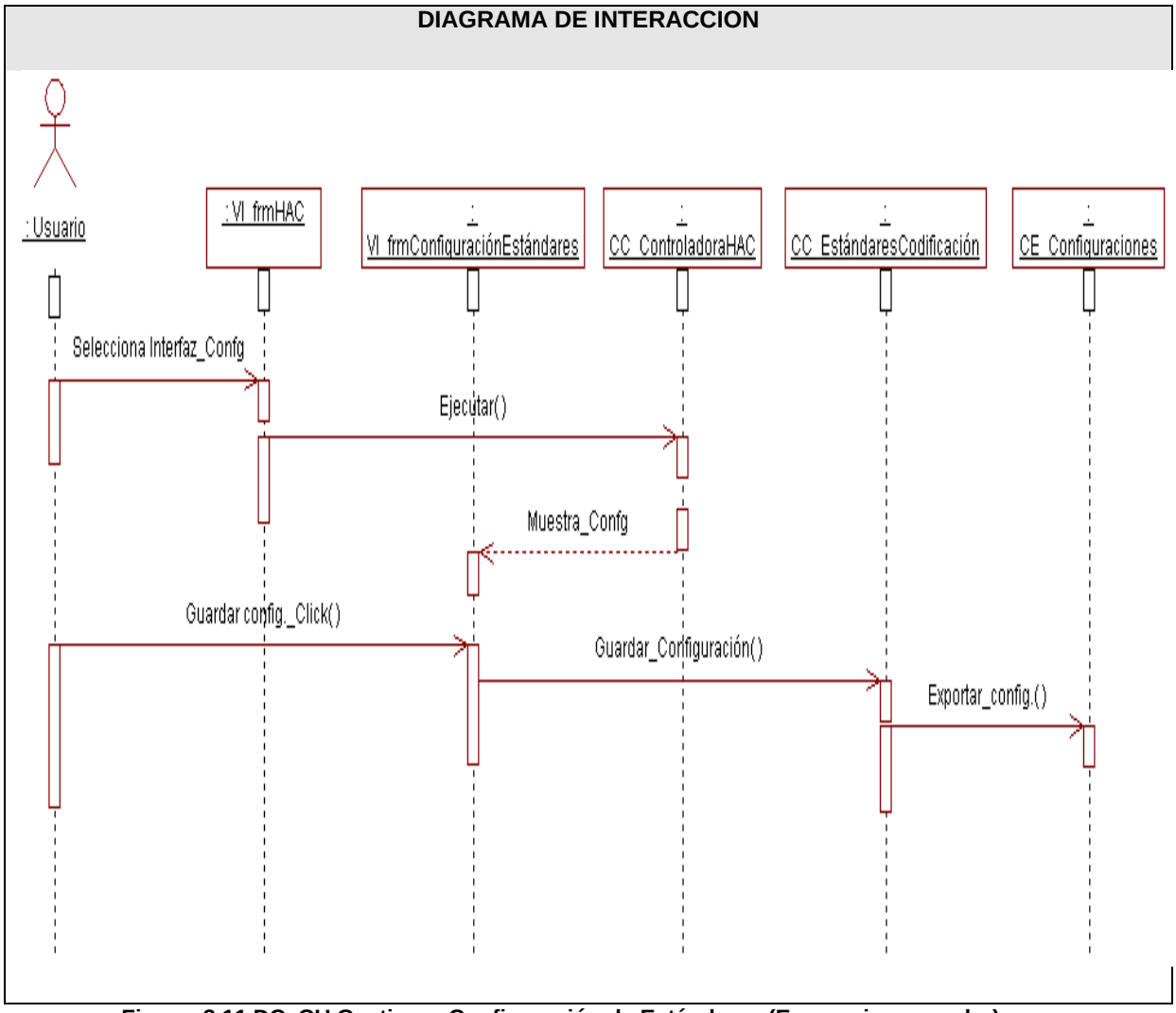


Figura. 3.11 DC_CU Gestionar Configuración de Estándares (Escenario _ guardar)

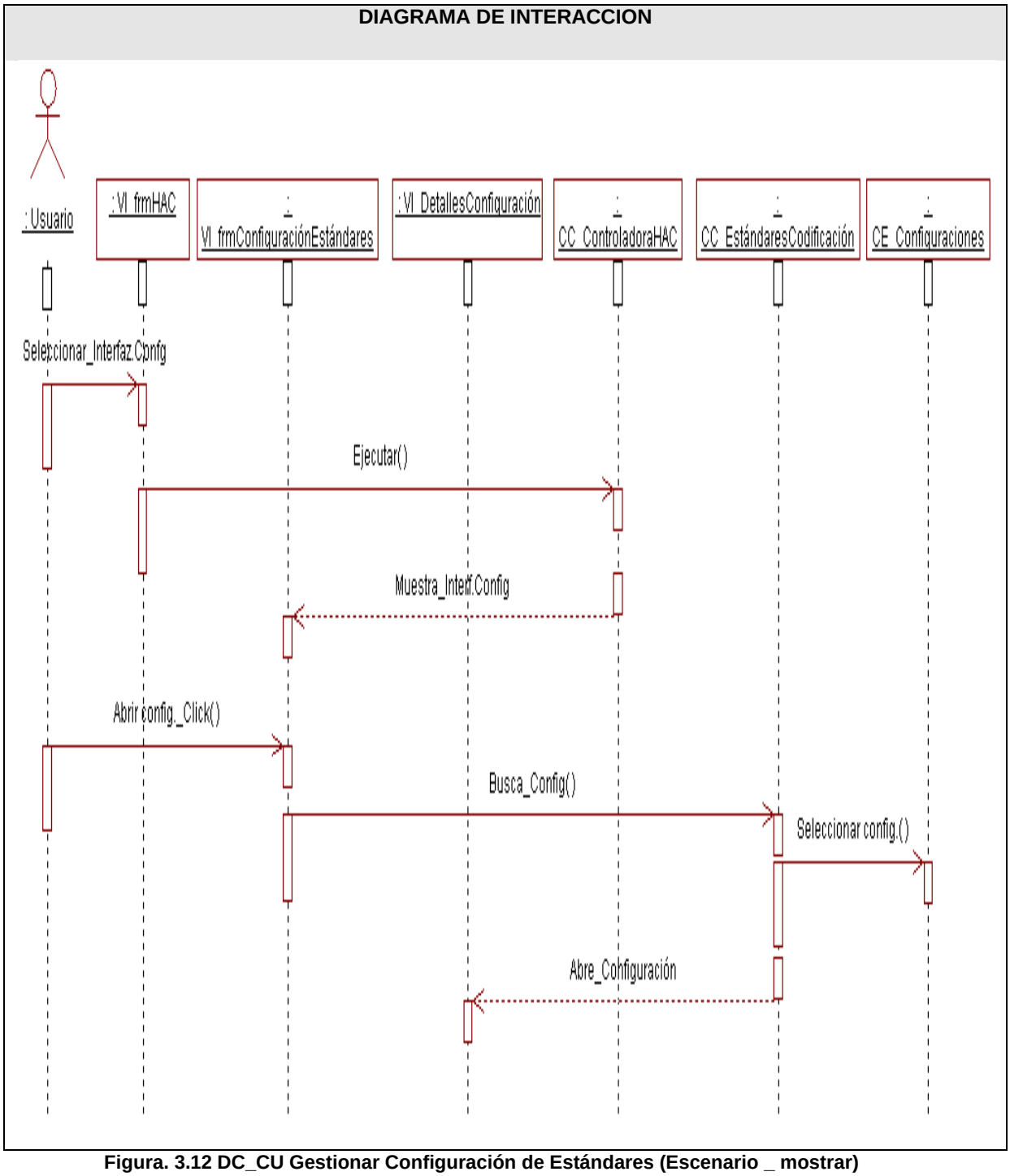


Figura. 3.12 DC_CU Gestionar Configuración de Estándares (Escenario _ mostrar)

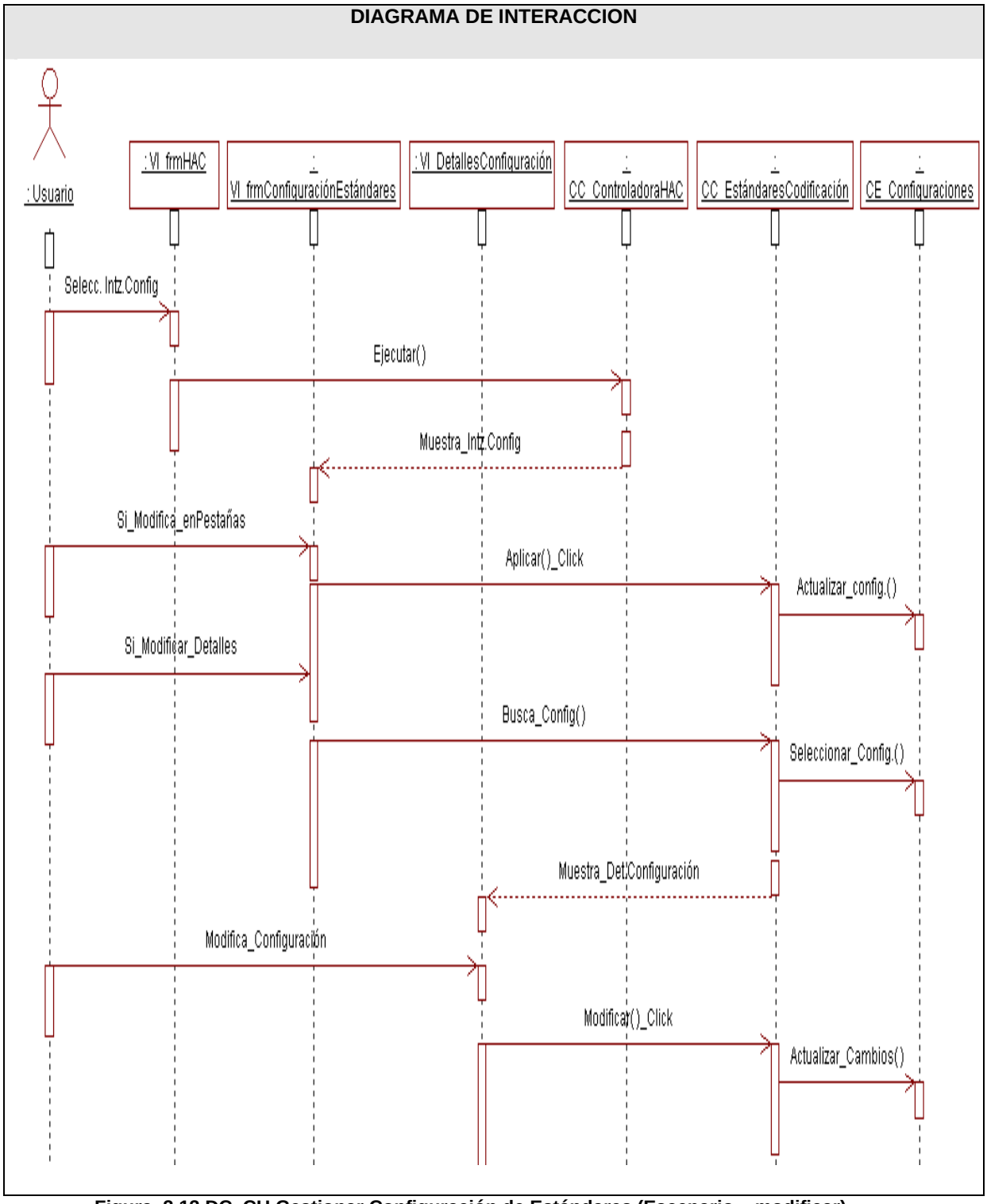


Figura. 3.13 DC_CU Gestionar Configuración de Estándares (Escenario _ modificar)

3.4 Un breve estudio de la factibilidad.

En el proceso de desarrollo de software una de las etapas más importante es la de planificación del proyecto, que no es más que la actividad de estimación de los resultados, el tiempo de desarrollo, desde que comienza el primer ciclo de trabajo hasta la entrega del producto final al cliente, el consumo de recursos tangibles e intangibles, como son: hombres, equipos, materiales gastables, en general, todos los recursos que deben ser invertidos para obtener el resultado deseado.

Para la realización de un proyecto es importante: estimar el esfuerzo humano, el tiempo de desarrollo que se requiere para la ejecución del mismo y también su costo. En este epígrafe se realizará el estudio de factibilidad del sistema utilizando el método mediante el análisis de Puntos de Casos de Uso. Este método permite documentar los requerimientos de un sistema en términos de Actores y Casos de Uso.

3.5 Planificación mediante puntos de casos de uso.

La estimación mediante el análisis de Puntos de Casos de Uso es un método propuesto originalmente por Gustav Karner de Objector y posteriormente refinado por muchos otros autores. Se trata de un método de estimación del tiempo de desarrollo de un proyecto mediante la asignación de "pesos" a un cierto número de factores que lo afectan, para finalmente, contabilizar el tiempo total estimado para el proyecto a partir de esos factores.

Este método consiste en la realización de una secuencia de pasos que se desarrollan a continuación:

El primer paso consiste en el cálculo de los Puntos de Casos de Uso sin ajustar. Este valor, se calcula a partir de 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 calcula mediante un análisis de la cantidad de Actores presentes en el sistema y la complejidad de cada uno de ellos. La complejidad de los Actores se establece teniendo en cuenta en primer lugar si se trata de una persona o de otro sistema, y en segundo lugar, la forma en la que el

actor interactúa con el sistema. En este caso el usuario constituye un actor de tipo complejo, ya que se trata de una persona utilizando el sistema mediante una interfaz gráfica, al cual se le asigna un peso 3.

Tabla 3: Factor de peso de los actores sin ajustar.

Tipo de actor	Descripción	Factor de peso	Actores	Total
Simple	Otros sistemas que interactúan con el sistema a desarrollar mediante una interfaz de programación (API).	**1**	**0**	**0**
Medio	Otros sistemas que interactúan con el sistema a desarrollar mediante un protocolo o una interfaz basada en texto.	**2**	**0**	**0**
Complejo	Una persona que interactúa con el sistema mediante una interfaz gráfica.	**3**	**1**	**3**

Luego **UAW**, que es el factor de peso de los actores sin ajustar, es igual a **3**.

Factor de Peso de los Casos de Uso sin ajustar (UUCW).

Este valor se calcula mediante un análisis de la cantidad de Casos de Uso presentes en el sistema y la complejidad de cada uno de ellos. La complejidad de los Casos de Uso se establece teniendo en cuenta la cantidad de transacciones efectuadas en el mismo, donde una transacción se entiende como una secuencia de actividades atómica, es decir, se efectúa la secuencia de actividades completa, o no se efectúa ninguna de las actividades de la secuencia. Como se tienen 9 casos de uso tipo simple (peso 5).

Tabla 3.1: Pesos de los casos de uso sin ajustar.

Tipo de CU	Descripción	Peso	Cantidad de CU.	Total
Simple	Casos de uso que tienen de 1 a 3 transacciones.	**5**	**9**	**45**

Medio	Casos de uso que tienen de 4 a 7 transacciones.	**10**	**0**	**0**
Complejo	Casos de uso que tienen más de 8 transacciones.	**15**	**0**	**0**

Entonces, el factor de peso de los casos de uso sin ajustar resulta:

UUCW = 9 x 5 = 45

Finalmente, los Puntos de Casos de Uso sin ajustar resultan:

UUCP = UAW + UUCW = 3+45= 48

Una vez que se tienen los Puntos de Casos de Uso sin ajustar, se debe ajustar éste valor mediante la siguiente 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 calcula mediante la cuantificación de un conjunto de factores que determinan la complejidad técnica del sistema. Cada uno de los factores se cuantifica con un valor de 0 a 5, donde 0 significa un aporte irrelevante y 5 un aporte muy importante.

Para calcular **TCF** se realiza mediante la siguiente fórmula:

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

Los pesos se obtienen utilizando la siguiente tabla:

Tabla 3.2: 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**	**0**	**0**	No hay cálculos complejos.
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 = 29

TCF = 0.6 + 0.01 * 29 = 0.89

Factor de ambiente (EF).

Estos factores son los que se contemplan en el cálculo del Factor de ambiente. El cálculo del mismo es similar al cálculo del Factor de complejidad técnica, es decir, se trata de un conjunto de factores que se cuantifican con valores de 0 a 5.

Para el cálculo de **EF** se sigue un procedimiento similar.

EF = 1.4 - 0.03 * Σ (**Pesoi * Valori**), donde **Valor** es un número del 0 al 5.

Los pesos se obtienen de la siguiente tabla:

Tabla 3.3. Pesos de los factores de 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 objetivos.	**1**	**4**	**4**	Se programa 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 programación	**-1**	**3**	**-3**	El lenguaje que se utiliza es C#.

Sumatoria = 19.5.

EF = 1.4 – 0.03 * 19.5 = 0,815

Quedando como puntos de casos de uso ajustados lo siguiente:

UCP = 48 * 0,89 * 0,815 = 34.8168

De los Puntos de Casos de Uso a la estimación del esfuerzo.

Karner originalmente sugirió que cada Punto de Casos de Uso requiere 20 horas-hombre. Posteriormente, surgieron otros refinamientos que proponen una granularidad algo más fina. Siguiendo dichos criterios:

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

Luego se calcula el esfuerzo estimado en horas - hombres.

E = UCP * CF = 34.8168 * 20 = 696.336 horas / hombre.

Se debe tener en cuenta que éste método proporciona una estimación del esfuerzo en horas-hombre contemplando sólo el desarrollo de la funcionalidad especificada en los casos de uso.

Finalmente, para una estimación más completa de la duración total del producto, hay que agregar a la estimación del esfuerzo obtenida por los Puntos de Casos de Uso, las estimaciones de esfuerzo de las demás actividades relacionadas con el desarrollo de software. Además se considera que este esfuerzo representa un porcentaje del esfuerzo total del proyecto, de acuerdo a los valores porcentuales:

Tabla 3.4: Cantidad de Horas-Hombres en cada fase de desarrollo de software.

Actividad	**Porcentaje %**	**Horas-Hombres**
Análisis	10	174.084
Diseño	20	348.168
Implementación	40	696.336
Pruebas	15	261.126

Sobre cargas (otras actividades)	15	261.126
Total	100	1740.84

Para convertirlo a hombres-mes

Como la jornada laboral de un día de trabajo es de 8 horas y en un mes se trabaja un aproximado de 24 días, entonces una persona en 1 mes trabaja 192 horas, por tanto:

Et= E (Horas-Hombres)/192horas-mes
Quedaría, Et= 1740.84 horas-Hombres/192horas-mes = 9 mes-hombres
Si en el proyecto trabajan tres hombres entonces el tiempo de desarrollo es:
Tiempo de desarrollo = Et /cantidad de hombres
Tiempo de desarrollo = 3 meses
El tiempo a emplear para el desarrollo de la aplicación es de 3 meses.

Salario

Para determinar el salario mensual se tiene en cuenta que los desarrolladores del sistema pueden ser ingenieros recién graduados, por lo que se toma como salario mensual: $225.

Costo.

Como anteriormente se definió el salario promedio de un ingeniero y son 3 personas para desarrollar el proyecto, entonces para hallar el costo total, sería:

Ct= Salario mensual*Cantidad de hombres*Tiempo de desarrollo
Ct= $ 2025
Ct= $ 81 cuc

3.6 Beneficios Tangibles e intangibles.

Beneficios Tangibles.

La universidad tiene como expectativas insertarse en el mercado internacional con sus productos software y no es menos cierto que en la actualidad la competencia de sistemas informáticos se incrementa a medida que transcurre el tiempo y se intenta que cada sistema posea las funcionalidades que exige el interesado, agregándole nuevas y eficientes características. Es por tal motivo la

importancia de garantizar que cada aplicación que sea desarrollada en la universidad, reúna la calidad esperada, sin embargo no existen medidores informatizados y abarcadores que respondan a dicho resultado. La herramienta que se propone encierra algunas características para aplicar un método de prueba que muy pocas veces se lleva a cabo y sería beneficioso para el desarrollo del proceso de pruebas.

Beneficios Intangibles.

Uno de los objetivos del país y particularmente el de la UCI es la exportación de software, el cual debe de poseer una alta calidad, es por ello la automatización del proceso de pruebas, siendo de vital importancia y bastante económico el desarrollo de herramientas en el proyecto para dicho proceso.

Esta herramienta permite realizarle pruebas al código de un software, para determinar la complejidad del código fuente y así conocer las probabilidades de errores que puede poseer, además de demostrarse qué tan válido y efectivo es el mismo. También verifica el cumplimiento de algunos parámetros de codificación de ese lenguaje de programación y si los estándares de codificación que se trazan los programadores se cometen.

Este tipo de prueba no se había realizado en el grupo de Calidad con anterioridad y resulta ser otra de las formas de poner al software en todas las situaciones posibles para verificar su adecuada funcionalidad y atestiguar la calidad del mismo, ahorrándose tiempo por parte del usuario, al dejar de ser manual dichas pruebas, uniéndosele la viabilidad, eficacia y amigabilidad que proporciona dicha herramienta al probador. Esto trae aparejado el incremento de la capacidad organizativa en el proyecto a la hora de realizar las pruebas, y sobre todo el perfeccionamiento del proceso de pruebas.

3.7 Análisis de costos y beneficios.

El costo de un producto debe estar justificado con los beneficios tangibles e intangibles que debe reportar. Como se apreció en el epígrafe anterior, uno de los posibles beneficios económicos que pueda aportar la herramienta es asegurar que un producto de exportación posea la calidad que el comprador espera, mediante la informatización de una herramienta desarrollada por parte del estudiantado, economizándose dinero en la utilización de otros sistemas para hacer pruebas, además de estar concebida con características específicas.

No obstante, por los beneficios intangibles que fueron mencionados anteriormente, el producto es viable y justificado, reportando grandes beneficios al grupo de Calidad e impactando en una de sus

más grandes necesidades, la informatización de la sociedad cubana y en particular del proceso de prueba que hoy en día es uno de los más importantes en el desarrollo de software, debido a las exigencias del cliente y la competencia existente en el mercado internacional.

Conclusiones.

En este capítulo se representaron las diferentes partes de la solución propuesta a través de diferentes diagramas para modelar los elementos relativos a la aplicación. Han sido definidas las clases de análisis y las clases del diseño así como sus relaciones en el Diagrama de Clases del Diseño de dos de los casos de uso principales de la herramienta. Además se confeccionó el Diagrama de Despliegue.

De esta manera, se ha logrado modelar todos los procesos que han sido objeto de estudio durante el transcurso del trabajo investigativo, proporcionando una idea completa de lo que realmente es el software. Se materializa con precisión los requerimientos del cliente. Se plantearon los principios de diseño de la aplicación, la concepción del Tratamiento de Errores y la concepción general de la Ayuda del Sistema.

Los diagramas y especificaciones de diseño propuestos constituyen una guía que puede ser fácilmente leída y comprendida por aquellos que construirán el código, lo probarán y le darán mantenimiento; lo único que resta es implementar la aplicación que se ha diseñado. También con el breve estudio de factibilidad del sistema, se estimó un tiempo de 3 meses para su construcción con 3 hombres y un costo de $ 2025 MN.

La aplicación propuesta brinda una serie de beneficios para los probadores de los diferentes proyectos productivos, ya que la misma va a contribuir a viabilizar el proceso de pruebas de caja blanca en el grupo de Calidad de la Facultad 7 y que puede ser extendido a otros proyectos, junto con los demás beneficio que trae consigo, lo que indica que es factible implementar la herramienta propuesta.

CONCLUSIONES

- Para llevar a cabo el modelamiento de la aplicación se realizó el análisis de algunas soluciones existentes en el mundo con características similares al sistema propuesto.
- Se realizó un estudio de las características del método de prueba de Caja Blanca, específicamente la técnica del camino básico, sugiriéndose que mientras menor sea el valor de la complejidad ciclomática, la codificación estará más libre de errores.
- La investigación permitió obtener parámetros de métricas para el código fuente, que deberá cumplir el programador para lograr una adecuada codificación. Además de establecer estándares de codificación para verificar mediante comprobaciones si se cumplen los mismos en el código fuente.
- El sistema se diseñó siguiendo la metodología RUP y se utilizaron representaciones para la modelación de todas las fases del proyecto excepto las de: Implementación y Prueba, que no constituyen objetivos del trabajo.
- En el desarrollo del trabajo se realizó el modelamiento del dominio y del sistema propuesto delimitado por el alcance del trabajo. Seguidamente se realizó el análisis de la aplicación para facilitar la comprensión y modelamiento del diseño propuesto. El sistema resultante está provisto de un ambiente cómodo, fácil de entender y cumple los estándares del diseño.
- Se han cumplido los objetivos trazados, modelándose una aplicación que satisface las necesidades y los requisitos solicitados por el cliente.

RECOMENDACIONES

Luego de la presentación del estudio realizado que culmina con el diseño de la Herramienta para viabilizar los procesos de pruebas del Grupo de Calidad de la Facultad 7, a continuación, se realizan recomendaciones para la ampliación, modificación, mejora y construcción de nuevas versiones de esta herramienta:

- Realizar la implementación de la aplicación, para la solución real de los problemas existente en el Grupo de Calidad de la Facultad 7.
- Continuar el estudio del tema con el objetivo de encontrar nuevas funcionalidades utilizando otras técnicas de pruebas, para futuras versiones de la aplicación.
- Integrar la herramienta al proceso de pruebas de software en otros grupos de Calidad.
- Incluir nuevos parámetros de codificación de las métricas seleccionadas y enriquecer la interpretación de las mismas.

REFERENCIAS BIBLIOGRÁFICAS

[1]. Díaz Yanersy, Molina Yenisel. Diseño de una aplicación para el Seguimiento de Errores de los productos software de la Facultad 7. UCI. Cuidad de la Habana. 2007.196 p.

[2]. El texto que ha sido referenciado está disponible en:

http://www.institutomardecortes.edu.mx/apuntes/sexto/desistemas01/Plan%20de%20PruebasIDS.ppt#257,64,Bibliografia.

[3]. Remitiéndose a [2].

[4]. Marcelo Rizzi, Francisco. Reportes Técnicos, Complejidad Ciclomática. ITBA. CAPIS. 8 p.

[5]. El texto que ha sido referenciado está disponible en:

www.institutomardecortes.edu.mx/apuntes/sexto/desistemas01/Plan%20de%20PruebasIDS.ppt.

[6]. Remitiéndose a la [2].

[7]. Capítulo 4: Técnicas de verificación y pruebas. Facultad de Informática. 17 p.

[8]. Polo Usaola, Dr. Macario. Curso de doctorado sobre Proceso software y gestión del conocimiento. Pruebas del Software. Departamento de Tecnologías y Sistemas de Información. Ciudad Real. 2006. (2008). 46 p.

[9]. El texto que ha sido referenciado está disponible en:

http://personal5.iddeo.es/ztt/Tem/T6_Matrices.htm

[10]. Está disponible en: http://www.tuobra.unam.mx/publicadas/040803214240.html

[11]. Juan Pablo Giraldo, 2001. (2008). Se encuentra en:

http://www.monografias.com/trabajos15/ingenieria-software/ingenieria-software2.shtml

[12]. Roger S. Pressman: “Software Engineering: A Practitioner's Approach” (European Adaptation), McGrawHill. 2000. (2008). ISBN: 0077096770.

[13]. Juristo Natalia, Moreno Ana M., Vegas Sira. Técnicas de Evaluación de Software. 2006. (2008). 131 p.

[14]. Remitiéndose a [13]. Juristo Natalia, Moreno Ana M., Vegas Sira. Técnicas de Evaluación de Software. 2006. (2008). 131 p.

[15]. Remitiéndose a [4].

[16]. Remitiéndose a [4].

[17]. Remitiéndose a [13].

[18]. Remitiéndose a [13].

[19]. Remitiéndose a [13].

[20]. Fernández, Giovanny. Estándar codificación DOTNET. España. 2005. (2008). 20 p.

[21]. Ingeniería de Software de Gestión III. Tema 5: Gestión de Proyectos Software Métricas. Dpto. de Lenguajes y Sistemas Informáticos. Escuela técnica superior de ingeniería informática (etsii). Universidad de Sevilla.

[22]. 2007. En la dirección:

https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Introducción%20a%20RUP.doc

[23]. 2007. En la dirección:

http://es.wikipedia.org/wiki/Metodolog%C3%ADas_%C3%A1giles

[24]. Remitiéndose a [1].

[25]. El texto que ha sido referenciado está disponible en:

http://www.monografias.com/trabajos24/herramientas-case/herramientas-case.shtml

[26]. 2007. El texto que ha sido referenciado está disponible en:

http://deigote.blogspot.com/2006/03/extreme-programming.html

[27]. 2007. SE puede encontrar en: http://cfrela.en.eresmas.com/uml/umlanalisis.htm

[28]. Rational Enterprise Edition, Ayuda extendida.

[29]. UML en 24 horas.

[30]. ERAZO, J. Un gran inicio: Patrones GRASP. 2007. (2008). Se localiza en:

http://jorgeerazo.blogspot.com/2006/08/patrones-grasp.html

[31]. SAAVEDRA, J. Patrones GRASP (Patrones de Software para la asignación General de Responsabilidad).Parte II, 2007. (2008). Se puede encontrar en:

http://jorgesaavedra.wordpress.com/2007/05/08/patrones-grasp-patrones-de-software-para-la-asignacion-general-de-responsabilidadparte-ii/

[32.] Autores: Colectivo de la Universidad de las Ciencias Informáticas. Conferencia 1. Introducción a la Ingeniería de Software.s.n.2005-2006.

[33]. Remitiéndose a [28].

[34]. Jacobson, I. y Booch, G. y Rumbaugh, J. El Proceso Unificado de Desarrollo de Software, Volumen I y II. La Habana: s.n., 2004. (2008).

[35]. Socorro Suarez, Sianny; Álvarez Pérez, Rigoberto Miguel. Diseño de una aplicación para el análisis de imágenes en la caracterización genética de microorganismos. UCI. Cuidad de la Habana. 2007. (2008). 103 p.

BIBLIOGRAFÍA

- Acuña, Cesar Javier. Pruebas de Software. Universidad Rey Juan Carlos. 65 p. http://kybele.escet.urjc.es/documentos/ISG/%5BISG-2006-07%5DPruebasSoftware.pdf
- A Aquiliano; Fuente Juan. Necesidad de sistemas formales de métricas para proyectos de software OO. Universidad de Oviedo. Dpto. de Matemáticas.
- Arbert, Germán; Faesch, Paula; Moran H, HéctorÍ. Técnicas de Pruebas de Software. Ppt.
- Asignatura: Práctica Profesional. Tema 6 Buenas prácticas. Clase #6 Pruebas de software. 11p.
- AQtime. http://www.automatedqa.com/products/aqtime/index.asp
- Beizer, Boris. Software Testing Techniques. Van Nostrand Reinhold (N.Y.). 2a ed. 1990.
- BRUEGGE, Bernd, DUTOIT, Allen; "Ingeniería de software orientado a objetos".2002. (2008). Prentice Hall - Pearson Educación. Capítulos 9 Páginas 326-369.
- BoundsChecker. http://www.compuware.com/products/devpartner/visualc.htm
- Bullseye Coverage. http://www.bullseye.com/productInfo.html
- Carrasco, Luis de Salvador. Métricas del Software. 9 p
- CMT++: http://www.verifysoft.com/en_csharp_testing.html
- Code Coverage. http://dynamic-memory.com/code_coverage.html
- Code Sheck. http://www.abxsoft.com/codchk.htm
- Consultoría en Seguridad, diagramas de colaboración. [En línea]. http://www.creangel.com/uml/interaccion.php.
- CTC++. http://www.testwell.fi/
- Cuadrado-Gallego, Juan J. Adaptación de las Métricas de Reusabilidad de la Ingeniería del Software a los Learning Objects. Disponible en: http://spdece.uah.es/papers/Cuadrado_Final.pdf

- Departamento de Informática, Universidad de Valladolid 40005, Plaza de Santa Eulalia, 9, Segovia, España.
- Díaz Yanersy, Molina Yenisel. Diseño de una aplicación para el Seguimiento de Errores de los productos software de la Facultad 7. UCI. Cuidad de la Habana. 2007. (2008). 196 p.
- Diseño de Casos de Pruebas. 2006. (2008). Disponible en:

  http://my.opera.com/pelican0/blog/show.dml/564829
- Diseño de Casos de Pruebas.

  http://www.angelfire.com/empire2/ivansanes/bywbox.htm
- Docklight Scripting. http://es.fileaward.com/docklight_scripting.html
- Garcia Pérez, Carlos. JUnit 4. Pruebas de Software Java.

  http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=junit4
- García Rubio, Félix Óscar; Bravo Santos, Crescencio. Ingeniería de Software.
- Ingeniería del Software e Gestión. Tema 7. Pruebas del Software. 17 p. http://alarcos.inf-cr.uclm.es/per/fgarcia/isoftware/doc/tema7_2xh.pdf
- Insure++. http://www.parasoft.com/jsp/products/home.jsp?product=Insure&/index.htm
- JACOBSON, Ivar; RUMBAUGH, James, BOOCH, Grady, “El lenguaje unificado de modelado”. 2000. (2008). Addison Wesley. Capítulo 11 Página 281-302 Capítulo 16 Página 381-394 Páginas 339-340, 361-363, 377-378.
- Jiménez, Darwin. Ciclo de vida del Software. Se encuentra en:

  http://darwinjimenezgarzon.blogspot.com/2007/08/ciclo-de-vida-de-software.html
- JProbe. http://www.quest.com/jprobe/
- JProbe. http://www.componentsource.com/products/jprobe-suite/index.html
- JTest.http://www.als-es.com/home.php?location=herramientas%2Fentorno-desarrollo%2Fjtest

- Juristo Natalia, Moreno Ana M., Vegas Sira. Técnicas de Evaluación de Software. 2006. (2008). 131 p.
- Koomen, T., Pol, M. Test Process Improvement - Leidraad voor stapsgewijs beter testen. Editorial Kluwer Bedrijfsinformatie. ISBN.1998. (2008).
- López Quesada, Juan Antonio. Ingeniería del Software asistida por Computador (CASE). Tema 3. Facultad de Informática. http://dis.um.es/~lopezquesada.
- LADRA Testbed. http://www.ldra.co.uk/testbed.asp
- Logiscope. http://www.telelogic.com
- Mañas, José A. Pruebas de Programas. Marzo.1994. (2008).
- Marcelo Rizzi, Francisco. Reportes Técnicos, Complejidad Ciclomática. ITBA. CAPIS. 8 p.
- Márquez Apízar, Yaimí; Valdés Hecheverría, Yenni. Procedimiento general de pruebas de caja blanca aplicando la técnica del camino básico. UCI. Cuidad de la Habana. 2007. (2008). 131 p
- Ministerio de Administraciones Públicas (MAP). Guía de Técnicas de Métrica y Guía de Referencia. v.2.1. 1995. (2008).
- Molpeceres, Alberto. Procesos de desarrollo: RUP, XP, FDD. Al AT javahispano DOT org. 2002. (2008). 11 p. http://www.javahispano.org/licencias.
- Mora Rioja, Arturo. Prueba de Aplicaciones. Desarrollo de Aplicaciones Informáticas. Módulo: Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión.2005. (2008). 9 p.
- MsCabe QT. http://www.mccabe.com/iq_developers.htm
- Myers, Glenford J. El Arte de Probar el Software (The Art of Software Testing). El ateneo. 1983. (2008).
- Otras Herramientas. http://www.parasoft.com/jsp/home.jsp
- Otras herramientas. http://www.codework.com/systemstools.html
- Patrones Del *Gang of Four.* Facultad de informática, Universidad de Madrid. Pepe Sacau Fontenla, 2004. (2008).

  http://descartes.cnice.mec.es/materiales_didacticos/Calculo_matricial_d3/defmat.htm

- PRESSMAN, Roger “Ingeniería del Software. Un enfoque práctico”. McGraw-Hill/Interamericana de España.2002. (2008).
- Pressman. 2002. (2008). Capítulos 17 y 18
- Piattini et al. 96. Capítulo 12.
- Proceso Unificado de Rational.
- Rational Enterprise Edition, Ayuda extendida.
- Rational Rose: Procedimientos básicos para desarrollar un proyecto con UML 2006]. Disponible en: http://www.vico.org/TallerRationalRose.pdf
- Rational Unified Process(RUP). 2006. (2008). Disponible en:

  https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Introducción%20a%20RUP.doc
- RSM: http://www.fileheaven.com/descargar/resource-standard-metrics-for-c-c-and-java/33666.htm
- Se encuentra en:

  http://es.wikipedia.org/wiki/Matriz_%28matem%C3%A1tica%29
- Se encuentra en:

  http://www.institutomardecortes.edu.mx/apuntes/sexto/desistemas01/Plan%20de%20PruebasIDS.ppt#257,64,Bibliografia
- Se encuentra en:

  http://www.institutomardecortes.edu.mx/apuntes/sexto/desistemas01/Plan%20de%20PruebasIDS.ppt#257,64,Bibliografia.
- Se puede localizar en: http://en.wikipedia.org/wiki/CamelCase.
- Se puede localizar en: http://www.faqs.org/rfcs/rfc2606
- Se puede localizar en: http://en.wikipedia.org/wiki/CamelCase

- Se puede localizar en: http://www.monografias.com/trabajos55/evolucion-lenguajes-de-programacion/evolucion-lenguajes-de-programacion2.shtml.
- Se puede localizar en: http://www.elforro.com/programacion/11729-aplicaciones-desktop.html
- Sitios: http://pear.php.net/manual/en/standards.php.
- Sitios: http://wwwdi.ujaen.es/asignaturas/computacionestadistica/pdfs/tema6.pdf.
- Sitios: http://trevinca.ei.uvigo.es/~adomin/csi/web/pdfs/csi_met.pdf
- Sobre herramientas de Pruebas. http://www.als-es.com/home.php
- Sobre herramientas de Pruebas. http://www.quest.com/
- "The Complete Guide to software testing". http://www.softwareqatest.com
- T.Koomen, M. Pol. Mejora del Proceso de Pruebas usando TPI.Sogeti Nederland B.V. 11 p
- UML-Análisis y diseño de Software. 2006]. Disponible en:

  http://cfrela.en.eresmas.com/uml/umlanalisis.htm
- Unified Modeling Language. 2006. (2008). Disponible en: http://www.uml.org/
- UML en 24 horas.
- Validación y Verificación. 18 p
- Vegas Fernández, Julio Antonio. Diplomado en Calidad en el Software. Derechos Reservados. 1999.
- Velásquez, Juan R (Prof.); Martínez Igor (Ing.); Batista, Hilda Elizabeth (Ing.); Reynoso, Daina Ivette (Lic.). Métricas en Ingeniería de Software. MO-TIC. Gestión de Proyectos
- Villena Romásn, Julio. Laboratorio de Programación. Pruebas de Programas.

ANEXOS

Anexo 1: Descripción Expandida de CU.

Tabla A1.1: Diagrama de Casos de Uso: Calcular complejidad ciclomática por matriz.

Caso de Uso #2:	Calcular Complejidad Ciclomática por matriz.
Actores:	Usuario.
Resumen:	En este CU se determina la Complejidad Ciclomática, a partir de la matriz importada por el usuario.
Precondiciones:	La matriz debe de estar generada.
Referencias	RF2
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- El usuario presiona el botón "Complejidad Ciclomática" después de tener la matriz cargada.	1.1- La aplicación visualiza una ventana con el valor de la complejidad ciclomática de la matriz seleccionada.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

Poscondiciones:	Se determina el valor de la complejidad ciclomática de la matriz.

Tabla A1.2: Diagrama de Casos de Uso: Determinar conjunto básico de caminos independientes.

Caso de Uso #3:	Determinar Conjunto Básico de Caminos Independientes.

Actores:	Usuario
Resumen:	En este CU se determina el conjunto básico de Caminos Linealmente Independientes, a partir de un fragmento de código o por la importación de una matriz que el usuario seleccione.
Precondiciones:	Se tiene que haber calculado la complejidad ciclomática del fragmento de código o de la matriz seleccionada para determinar el conjunto básico de Caminos Linealmente Independientes.
Referencias	RF3
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- El usuario selecciona la opción "Caminos Independientes" que se encuentra en la interfaz donde se determina la Complejidad Ciclomática.	1.1- La aplicación visualiza en la misma interfaz todos los caminos linealmente independientes existentes.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

Poscondiciones:	Se visualizan los caminos linealmente independientes, del código o la matriz según la selección del usuario.

Tabla A1.3: Diagrama de Casos de Uso: Importar matriz

Caso de Uso #4:	Importar Matriz.
Actores:	Usuario
Resumen:	En este CU se importa la matriz de un fragmento de código que se haya guardado por el usuario anteriormente.

Precondiciones:	Tener la matriz almacenada.
Referencias	RF4
Prioridad	Medio

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- El usuario mediante la opción del menú importa una matriz.	1.1- La aplicación visualiza una interfaz que muestra la matriz seleccionada.

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

Poscondiciones:	Se importa la matriz seleccionada.

Tabla A1.4: Diagrama de Casos de Uso: Comprobar configuración de estándares de codificación.

Caso de Uso #6:	Comprobar Configuración de Estándares de Codificación.
Actores:	Usuario
Resumen:	En este CU se comprueba el cumplimiento de los estándares de codificación que se definieron para el lenguaje en que se encuentra el fragmento de código que el usuario selecciona.
Precondiciones:	Que estén definidos los estándares de codificación que se comprobarán.
Referencias	RF6
Prioridad	Crítico

Flujo Normal de Eventos

Acción del Actor	Respuesta del Sistema
1- El usuario en las opciones del menú o mediante la barra de herramientas, selecciona la opción: “Estándares de Codificación”.	1.1-La aplicación visualiza una interfaz con el código que está dentro del archivo, además de la opción de “Realizar Análisis.”
2- El usuario selecciona el botón “Realizar Análisis”.	2.1- La aplicación visualiza en la misma interfaz, un resultado satisfactorio, revelando el cumplimiento de los estándares de codificación. Si no se cumplen, ver el Flujo Alterno correspondiente “Comprobar estándares”.
Poscondiciones:	Se muestra un mensaje: “Comprobación satisfactoria”.
Flujo Alterno “Comprobar estándares”	
Acción del Actor	**Respuesta del Sistema**
	2.2- La aplicación muestra el incumplimiento de los estándares de codificación, señalando la (s) línea (s) en la (s) cual (es) no se cumple (n) los mismos.
Poscondiciones:	Se muestran los tipos de incumplimientos y el lugar de los mismos.

Tabla A1.5: Diagrama de Casos de Uso: Generar reporte de parámetros de codificación.

Caso de Uso #7:	Generar Reporte de Parámetros de Codificación.
Actores:	Usuario
Resumen:	En este CU se verifica la cantidad de líneas que cumplen con los parámetros de codificación del fragmento de código seleccionado por el usuario. Además de tener las opciones de imprimir y guardar el resultado.

Precondiciones:	Que el fragmento de código esté generado. Y para activarse los botones de impresión y exportar, se tiene que visualizar un reporte antes. En el caso de imprimir, también necesita de una impresora conectada a la computadora.
Referencias	RF7 (RF7.1, RF7.2)
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- El usuario en las opciones del menú o mediante la barra de herramientas, selecciona la opción: “Parámetros del Código”. 2- El usuario selecciona una de las siguientes opciones: Si selecciona la opción: “Reporte Archivo”. Ver Sección: “Reporte Archivo”. Si selecciona la opción: “Reporte Clases”. Ver sección: “Reporte Clases”	1.1- La aplicación visualiza una interfaz con las opciones: “Reporte Archivo” y “Reporte Clases”.

Flujo Normal de Eventos	
Sección “Reporte Archivo”	
Acción del Actor	**Respuesta del Sistema**
2- El usuario puede seleccionar las siguientes	1.1-La aplicación internamente analiza el fragmento de código fuente y posterior a ello, visualiza una interfaz con la cantidad de líneas del código, que cumplen con los parámetros de codificación.

opciones: Si selecciona la opción de imprimir. Ver sección “Imprimir”. Si selecciona la opción de exportar. Ver sección “Exportar”.	
Poscondiciones:	Se visualiza el reporte de archivos con los datos de los parámetros.
Flujo Normal de Eventos	
Sección “Imprimir”	
Acción del Actor	**Respuesta del Sistema**
2-El usuario escoge el tipo de impresora y a vía para la impresión y presiona el botón: “Aceptar”.	1.1-La aplicación visualiza una ventana para que el usuario seleccione el tipo de impresora. 2.1-La aplicación imprime el reporte de los parámetros de codificación.
Poscondiciones:	Se imprime el reporte con los datos de los parámetros.
Flujo Normal de Eventos	
Sección “Exportar”	
Acción del Actor	**Respuesta del Sistema**
2-El usuario determina el lugar y presiona el	1.1-La aplicación visualiza una ventana para que el usuario seleccione el lugar que desea guardar el reporte. 2.1-La aplicación exporta el reporte de los

botón: “Aceptar”.	parámetros de codificación.
Poscondiciones:	Se exporta el reporte con los datos de los parámetros.
Flujo Normal de Eventos	
Sección “Reporte Clase”	
Acción del Actor	**Respuesta del Sistema**
2- El usuario puede seleccionar las siguientes opciones: Si selecciona la opción de imprimir. Ver sección “Imprimir”. Si selecciona la opción de exportar. Ver sección “Exportar”.	1.1-La aplicación internamente analiza el fragmento de código fuente y posterior a ello, visualiza una interfaz con la cantidad de líneas del código, que cumplen con los parámetros de codificación.
Poscondiciones:	Se visualiza el reporte de clases con los datos de los parámetros.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

Tabla A1.6: Diagrama de Casos de Uso: Gestionar Reportes de comprobación de estándares.

Caso de Uso #9:	Gestionar Reportes de comprobación de estándares

Actores:	Usuario
Resumen:	En este CU el usuario genera a partir de la comprobación de estándares un reporte de incumplimientos, además de exportarlo e imprimirlo según estime conveniente.
Precondiciones:	Debe de existir al menos un incumplimiento. Que exista conexión a una impresora en caso de que el usuario desee imprimir un reporte.
Referencias	RF8 (RF8.1, RF8.2, RF8.3)
Prioridad	Medio.

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1-El usuario selecciona en la interfaz de estándares la opción “Reporte” y escoge una de las opciones que se despliegan: Si selecciona el opción Generar Reporte”. Ver sección “Generar Reporte”. Si selecciona la opción “Exportar”. Ver sección “Exportar”. Si selecciona la opción “Imprimir”. Ver sección “Imprimir”.	

Flujo Normal de Eventos	
Sección “Generar Reporte”	
Acción del Actor	**Respuesta del Sistema**
	1.1- Crea una interfaz con un nuevo reporte donde se visualiza los incumplimientos por líneas de

	código.
Poscondiciones:	Se visualiza el reporte creado.
Flujo Normal de Eventos	
Sección "Exportar"	
Acción del Actor	**Respuesta del Sistema**
3- El usuario selecciona el lugar para guardar el resultado de la comprobación de estándares.	2.1-La aplicación visualiza una interfaz para elegir el lugar donde se guarden los reportes. 3.1- La aplicación exporta el reporte con los datos que el usuario eligió.
Poscondiciones:	Se exporta el reporte con los datos de incumplimiento de la comprobación de estándares.
Flujos Alternos	
Sección "Imprimir"	
Acción del Actor	**Respuesta del Sistema**
	1.1-El sistema imprime el resultado del incumplimiento de los estándares.
Poscondiciones:	Se imprime el reporte con los datos de incumplimiento de la comprobación de estándares.

Tabla A1.7: Diagrama de Casos de Uso: Gestionar Reportes de complejidad ciclomática.

Caso de Uso 11:	Gestionar Reportes de complejidad ciclomática.

Actores:	Usuario
Resumen:	En este CU el usuario genera a partir del cálculo de la complejidad ciclomática un reporte con el valor de la misma para cada función del fragmento de código seleccionado, además de exportarlo e imprimirlo según estime conveniente.
Precondiciones:	Que se determine el valor de la complejidad ciclomática del código seleccionado por el usuario.
Referencias	RF8 (RF8.1, RF8.2, RF8.3)
Prioridad	Medio.

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- El usuario selecciona en la interfaz de complejidad ciclomática la opción “Reporte” y escoge una de las opciones que se despliegan: 2- El usuario selecciona una de las siguientes opciones: Si selecciona el botón “Generar Reporte”. Ver sección “Generar Reporte”. Si selecciona el botón “Exportar”. Ver sección “Exportar”. Si selecciona el botón “Imprimir”. Ver sección “Imprimir”.	

Flujo Normal de Eventos	
Sección “Generar Reporte”	
Acción del Actor	**Respuesta del Sistema**

	2.1- La aplicación visualiza una interfaz con el valor de la complejidad ciclomática de cada función.
Poscondiciones:	Se genera el reporte con el valor de la complejidad ciclomática de cada función del fragmento de código.
Flujo Normal de Eventos	
Sección "Exportar"	
Acción del Actor	**Respuesta del Sistema**
3- El usuario selecciona el lugar para guardar el reporte.	2.1-La aplicación visualiza una ventana para elegir el lugar donde se guarden los reportes con el valor de la complejidad ciclomática de cada función. 3.1- La aplicación exporta el reporte con los datos que el usuario eligió.
Poscondiciones:	Se exporta el reporte con el valor de la complejidad ciclomática de cada función del fragmento de código.
Flujos Alternos	
Sección "Imprimir"	
Acción del Actor	**Respuesta del Sistema**
3.- El usuario configura la impresora y presiona el botón "Aceptar."	2.1- El sistema visualiza una ventana para que el usuario escoja y configure la impresora. 3.1- El sistema imprime el resultado con los valores de la complejidad ciclomática de las funciones del fragmento de código.
Poscondiciones:	No se imprime los valores de la complejidad ciclomática de otro código.

Anexo 2: Representación de algunos Diagramas de las clases del análisis de los CU.

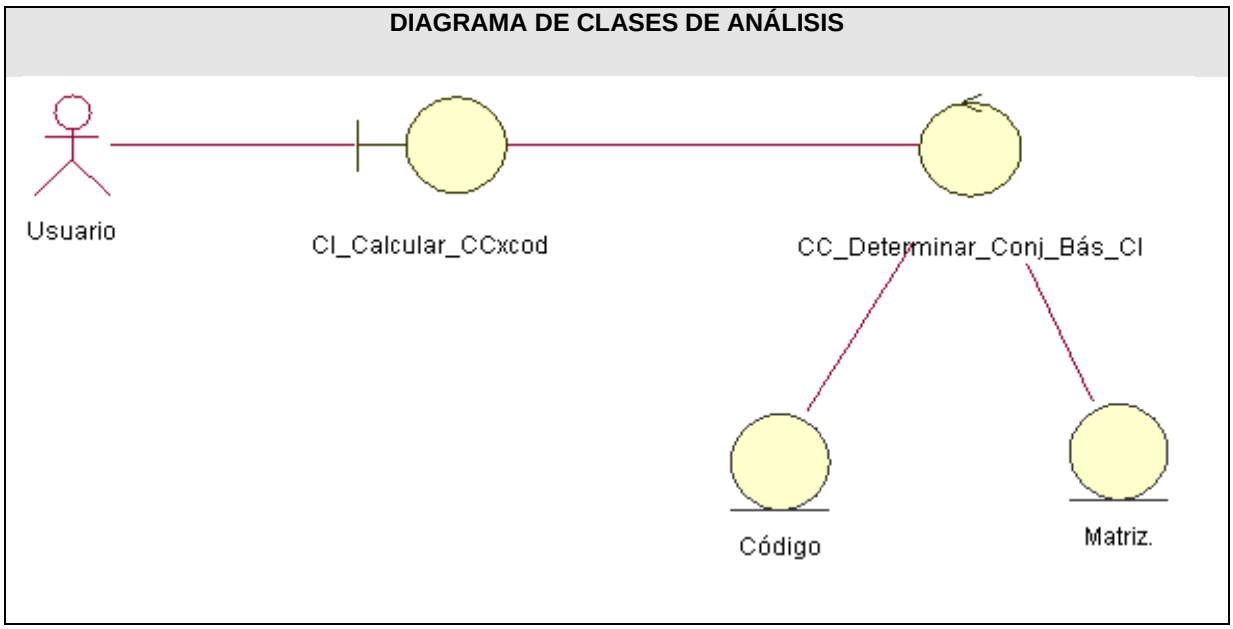


Figura A1. DCA_CU Determinar Conjunto Básico de Caminos Independientes.

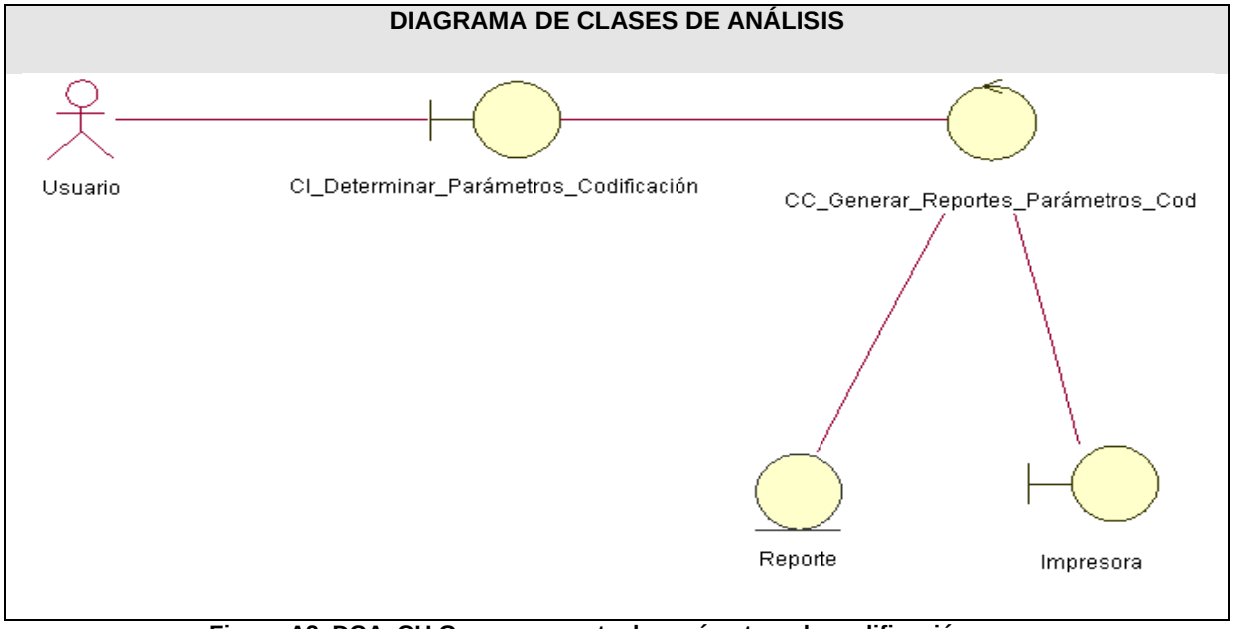


Figura A2. DCA_CU Generar reporte de parámetros de codificación.

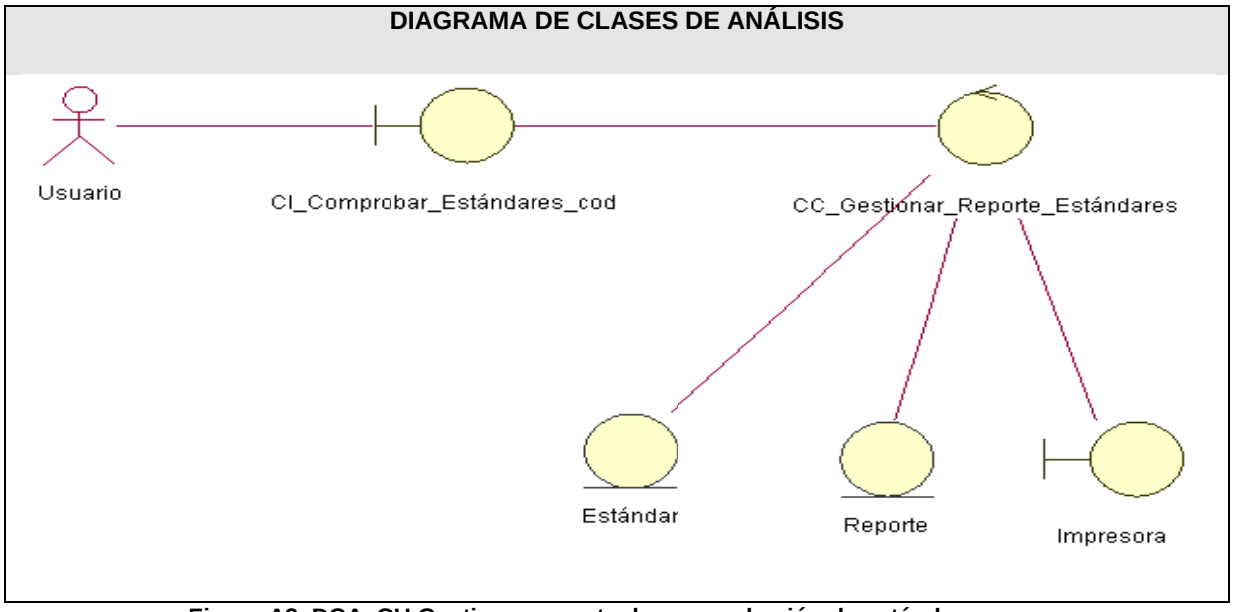


Figura A3. DCA_CU Gestionar reporte de comprobación de estándares.

Anexo 3: Representación de los Diagramas de interacciones. Colaboración.

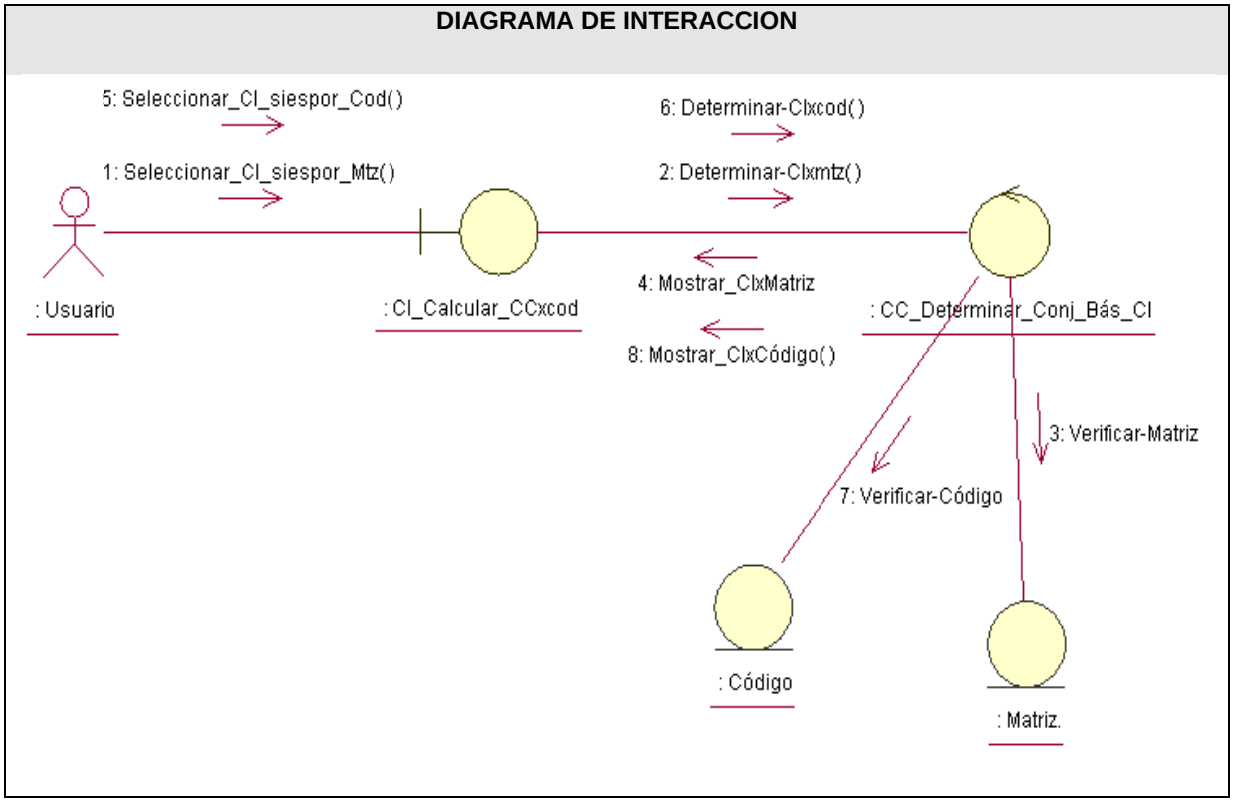


Figura A4. DCA_CU Determinar Conjunto Básico de Caminos Independientes.

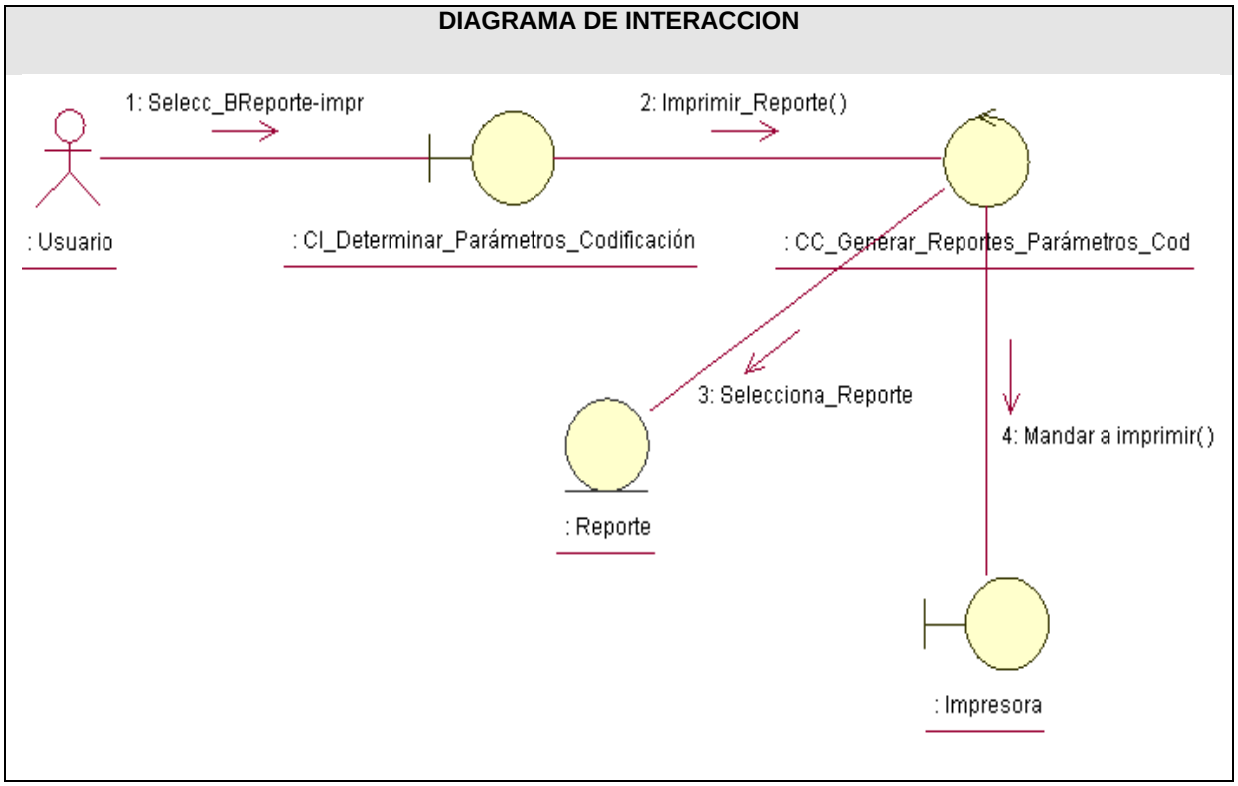


Figura A5. DCA_CU Generar reporte de parámetros de codificación. (Sección: Imprimir)

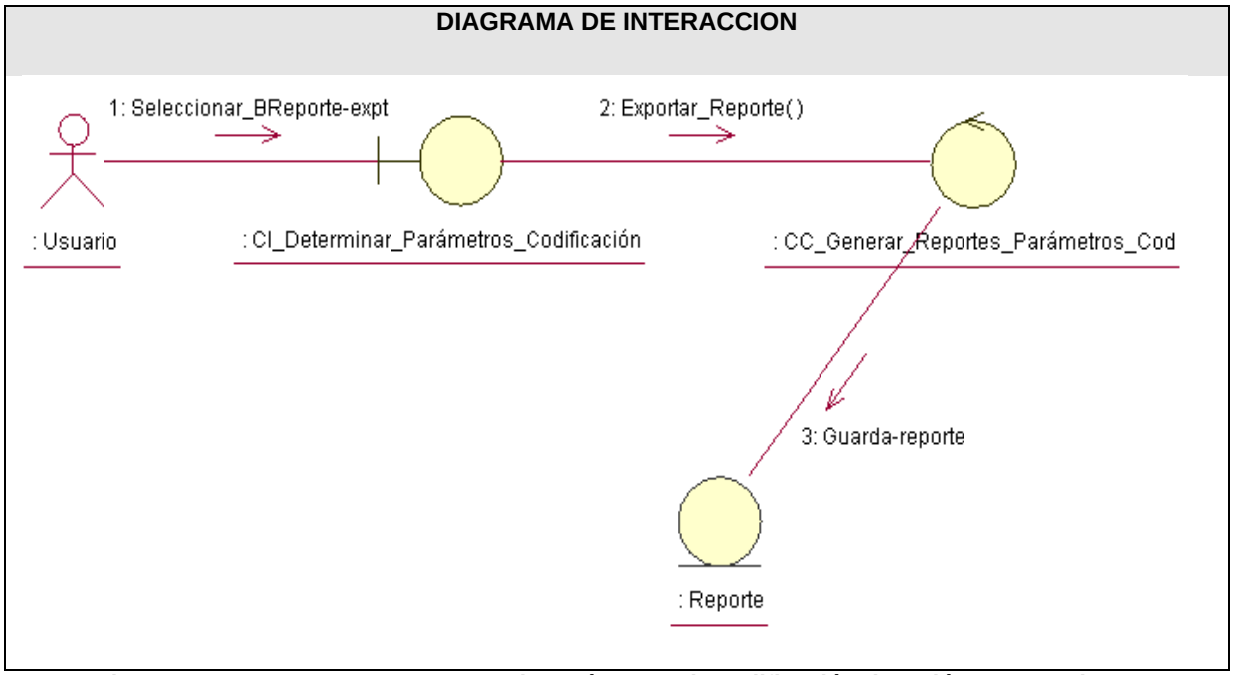


Figura A6. DCA_CU Generar reporte de parámetros de codificación. (Sección: Exportar)

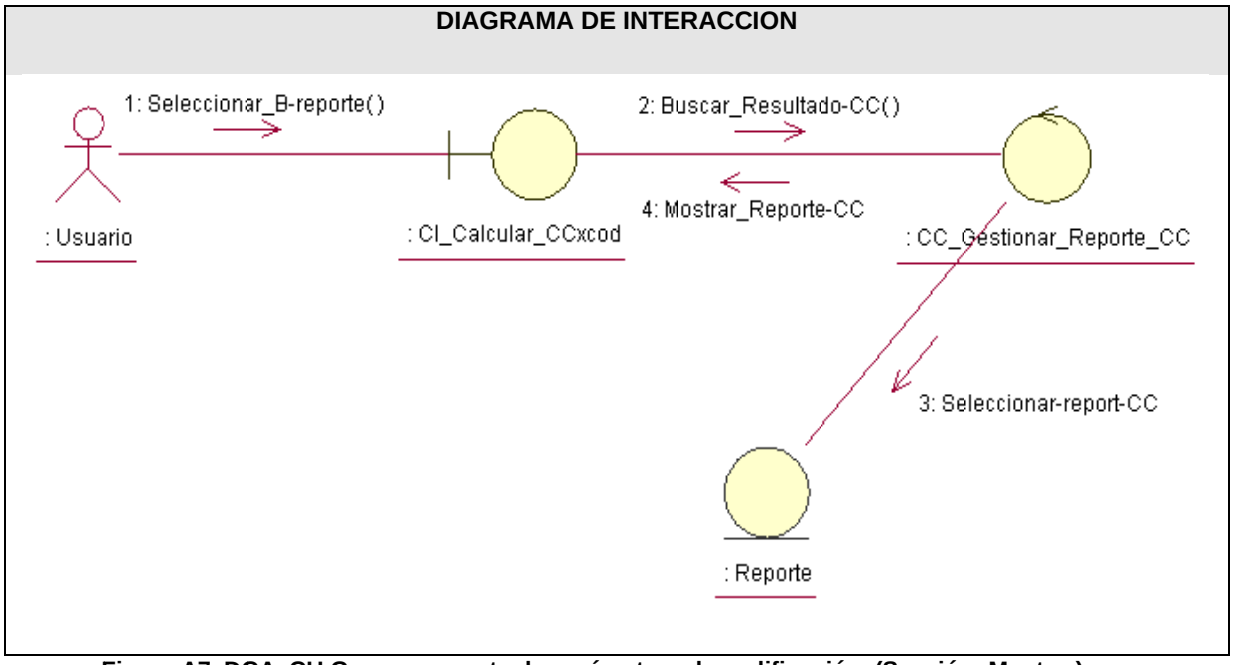


Figura A7. DCA_CU Generar reporte de parámetros de codificación. (Sección: Mostrar)

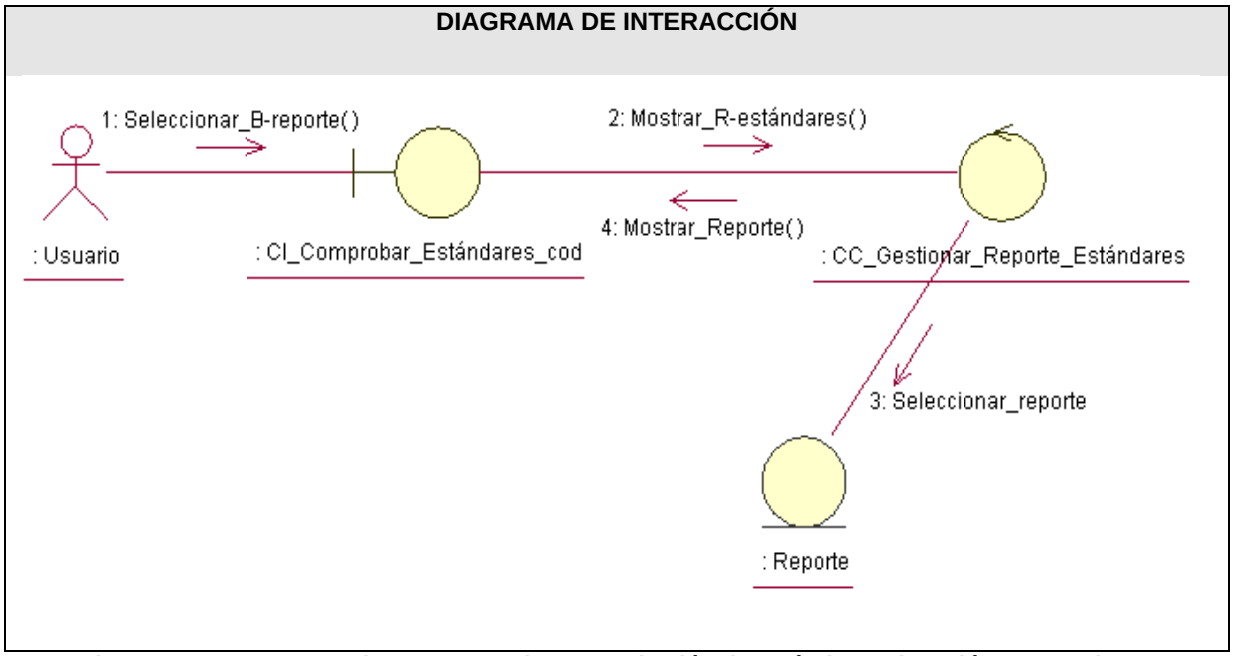


Figura A8. DCA_CU Gestionar reporte de comprobación de estándares. (Sección: Mostrar)

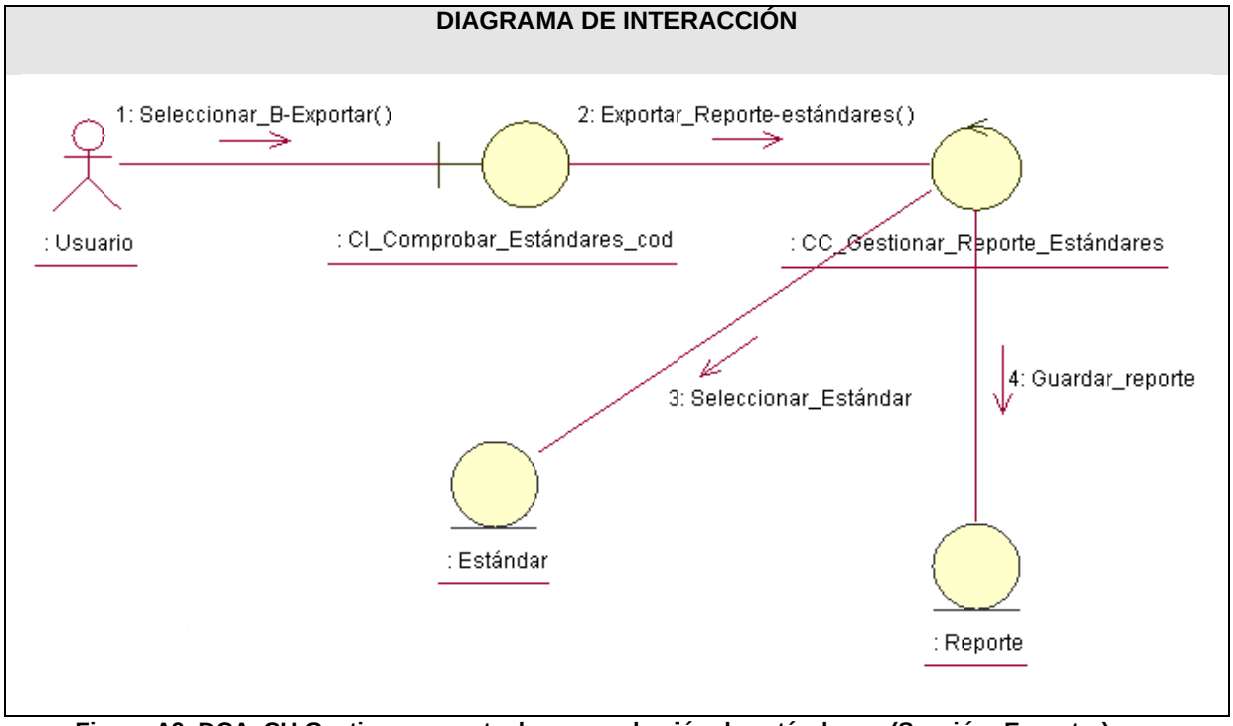


Figura A9. DCA_CU Gestionar reporte de comprobación de estándares. (Sección: Exportar)

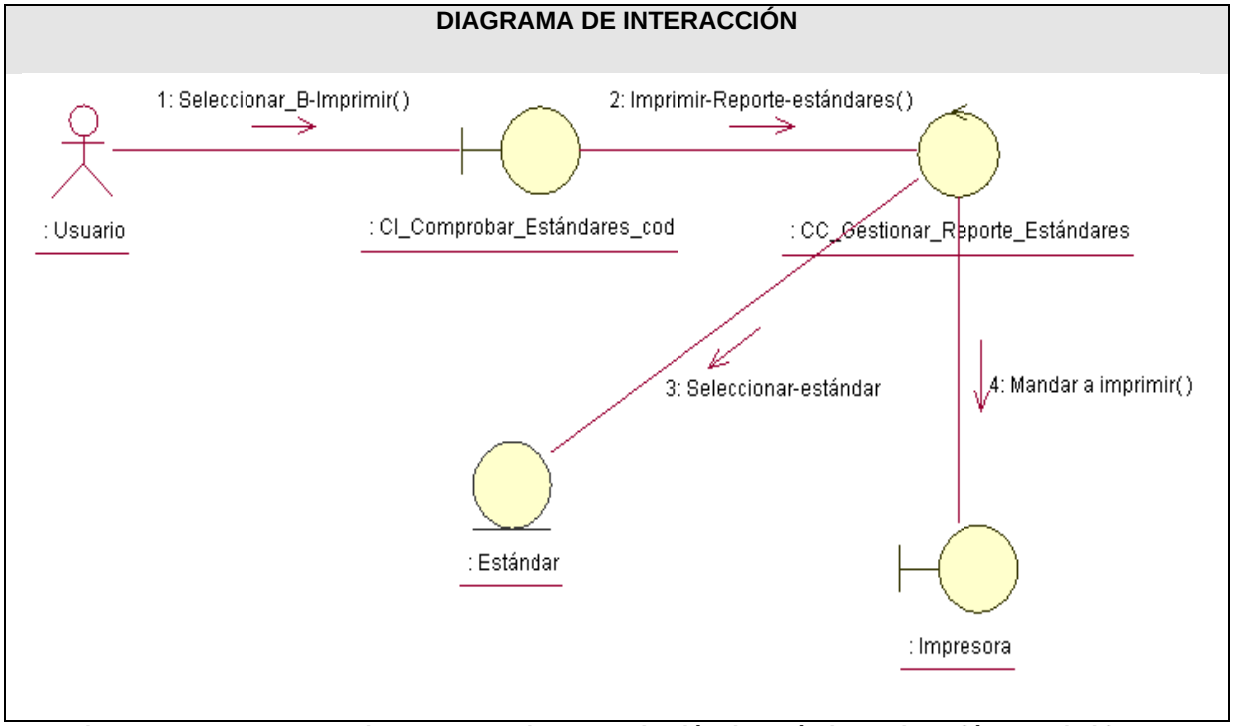


Figura A10. DCA_CU Gestionar reporte de comprobación de estándares. (Sección: Imprimir)

Anexo 4: Representación de los Diagramas de clases del diseño de los CU.

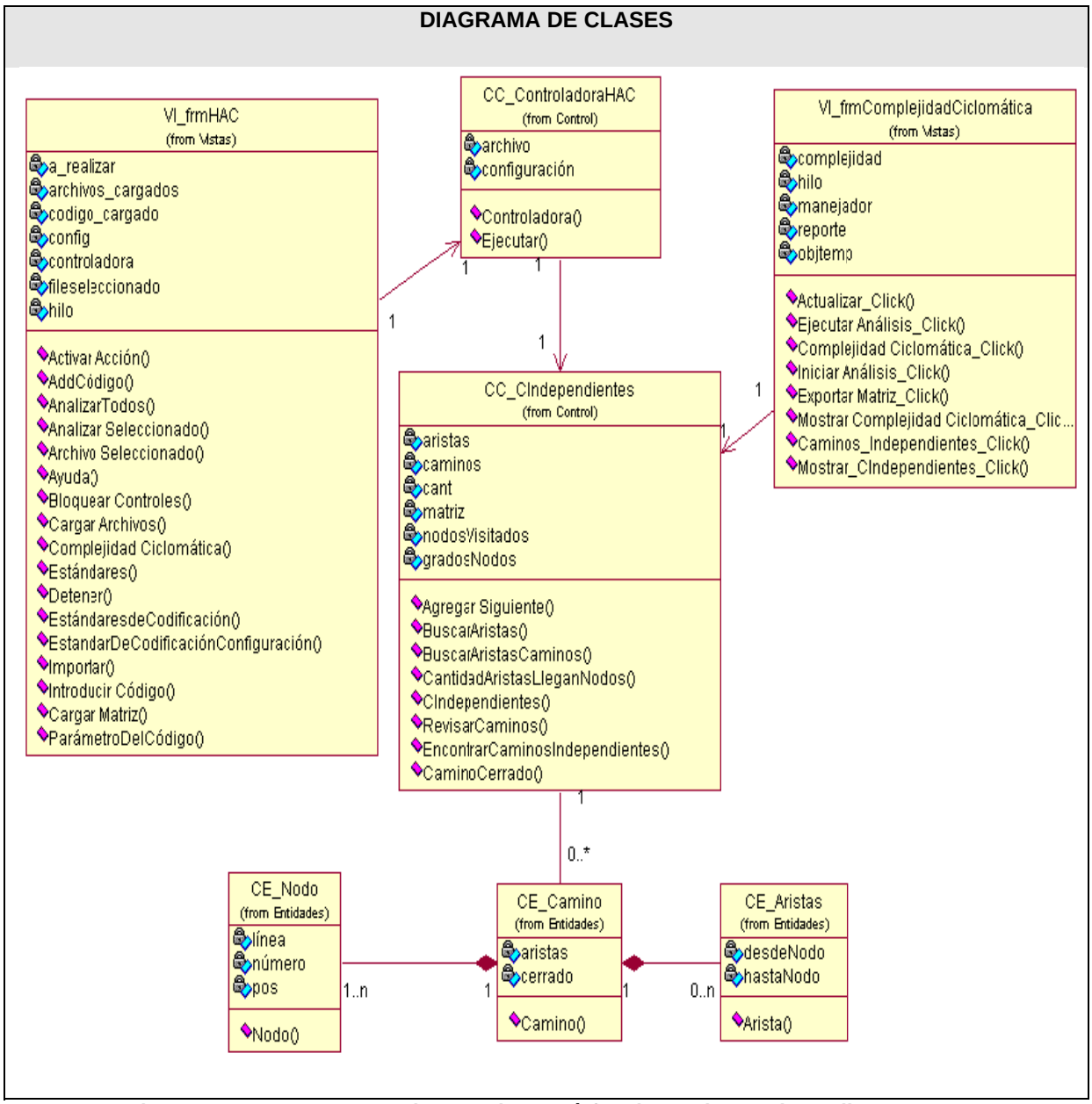


Figura A11. DCA_CU Determinar Conjunto Básico de Caminos Independientes.

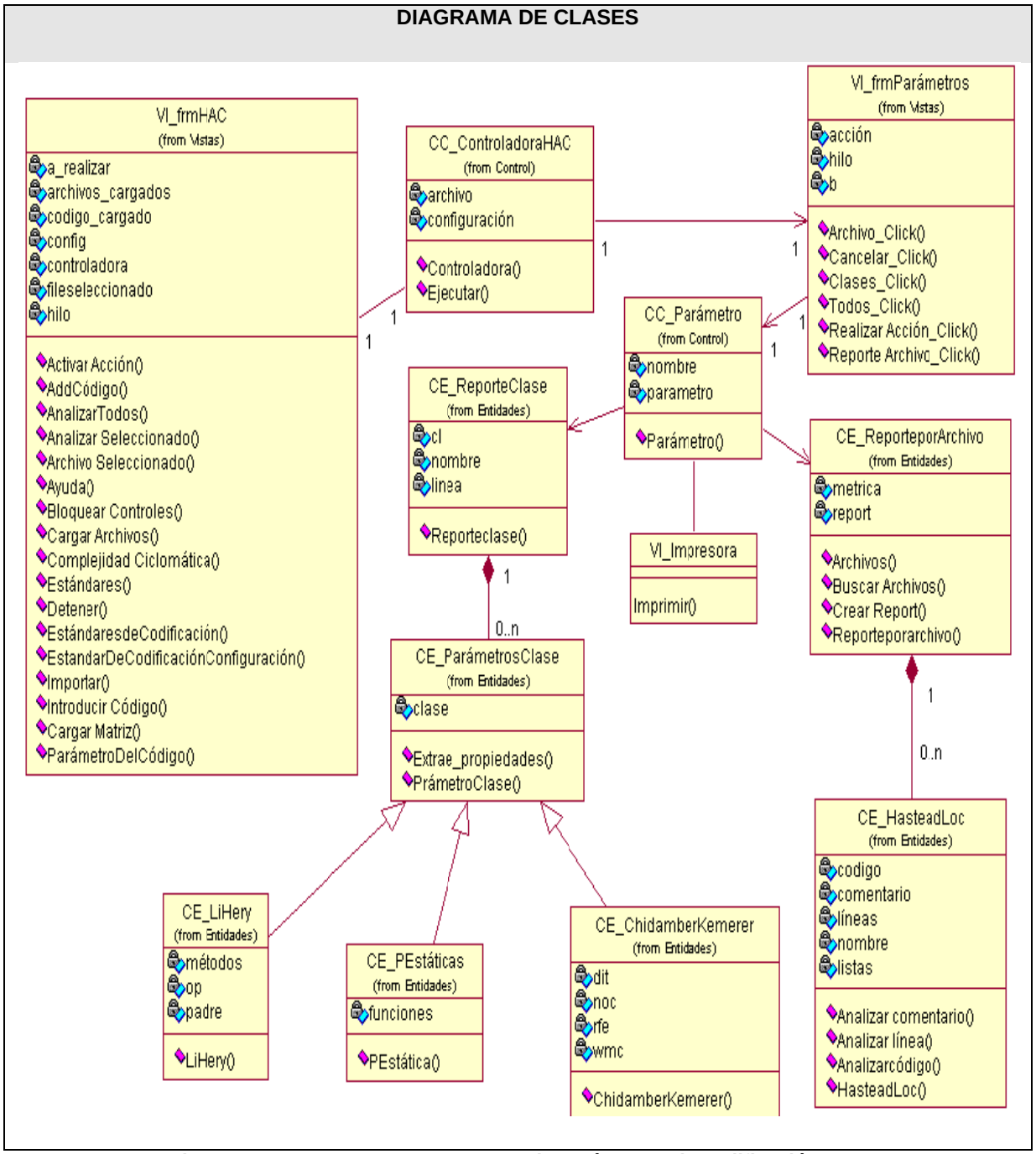


Figura A12. DCA_CU Generar reporte de parámetros de codificación.

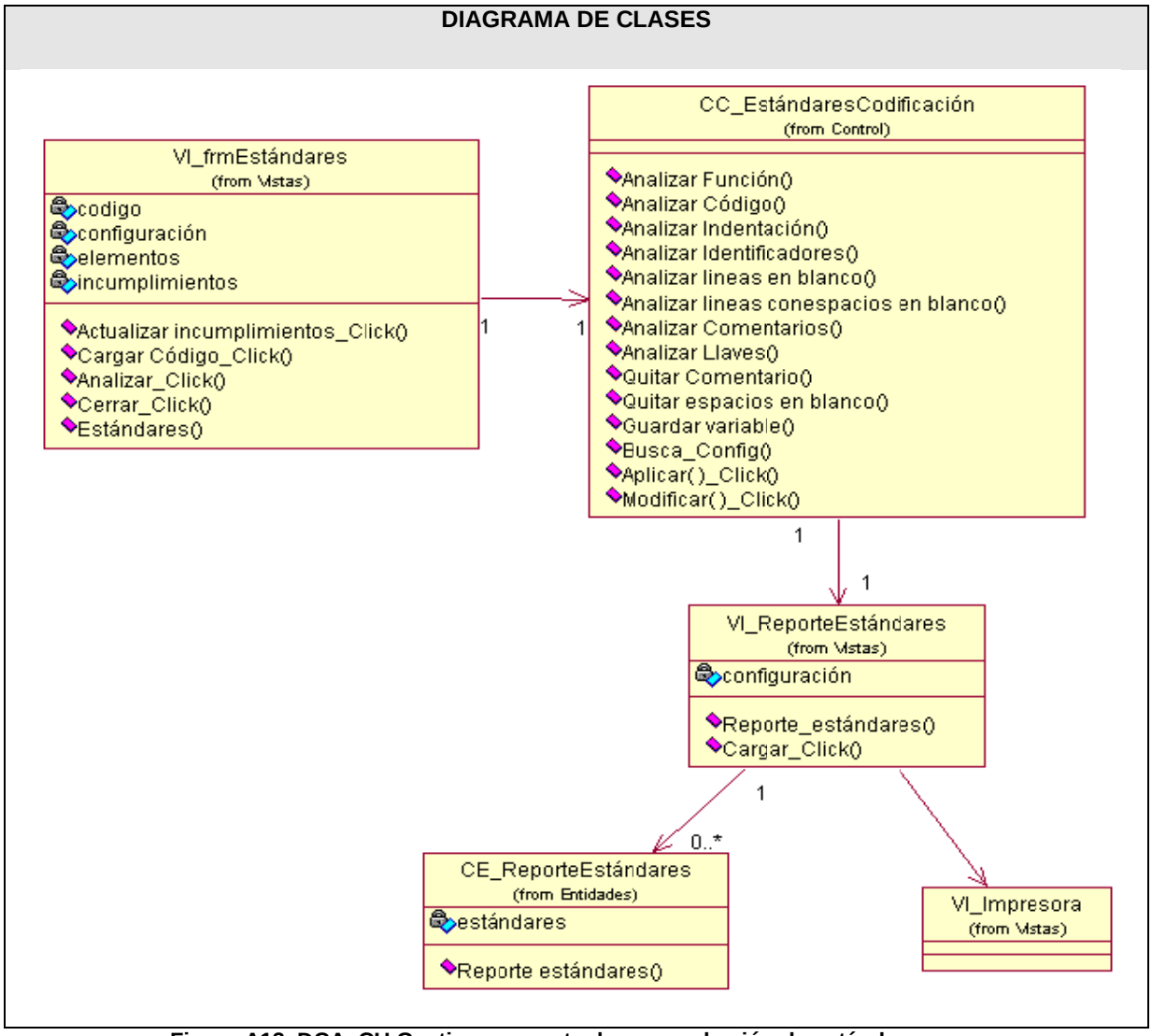


Figura A13. DCA_CU Gestionar reporte de comprobación de estándares.

Anexo 5: Representación de los Diagramas de interacciones. Secuencia.

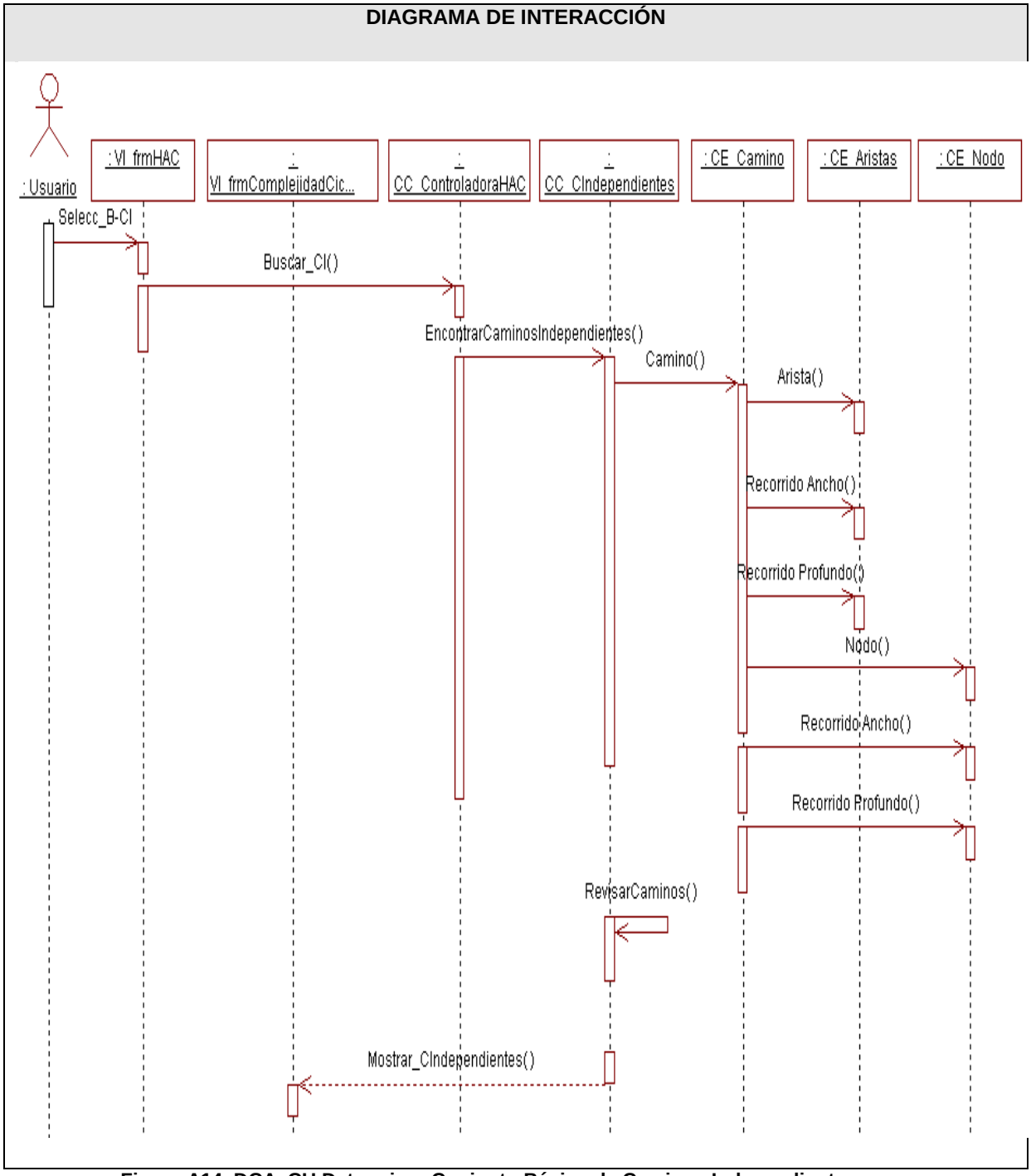


Figura A14. DCA_CU Determinar Conjunto Básico de Caminos Independientes.

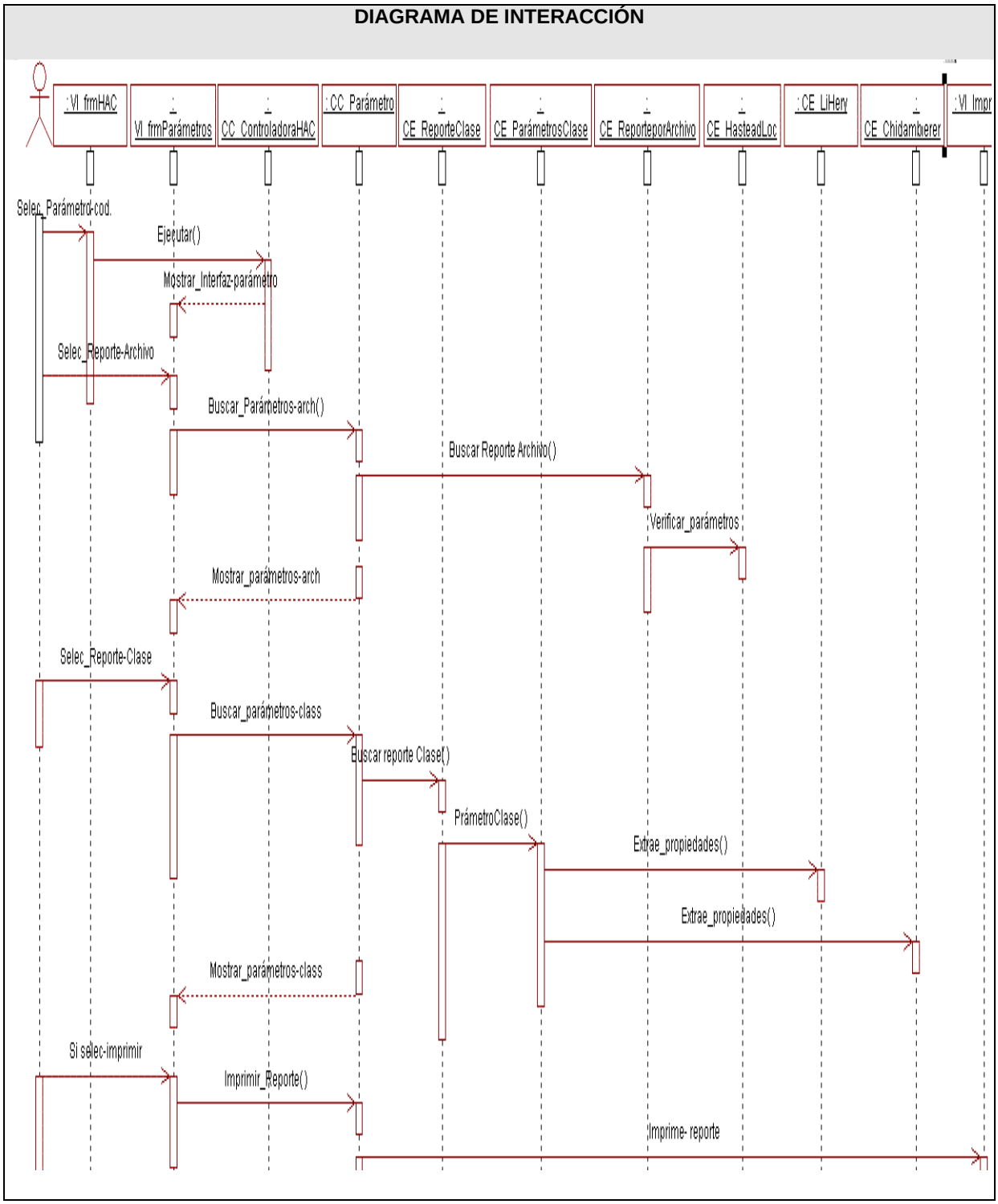


Figura A15. DCA_CU Generar reporte de parámetros de codificación.

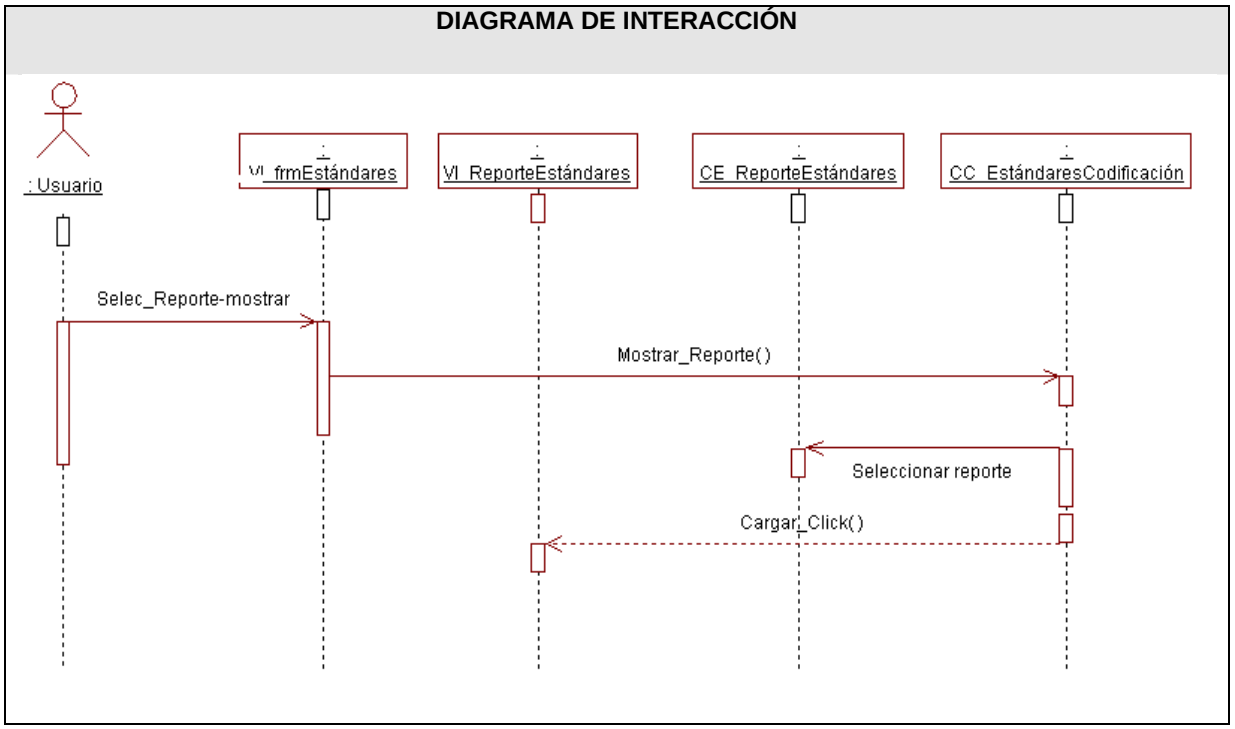


Figura A16. DCA_CU Gestionar reporte de comprobación de estándares. (Escenario: Mostrar)

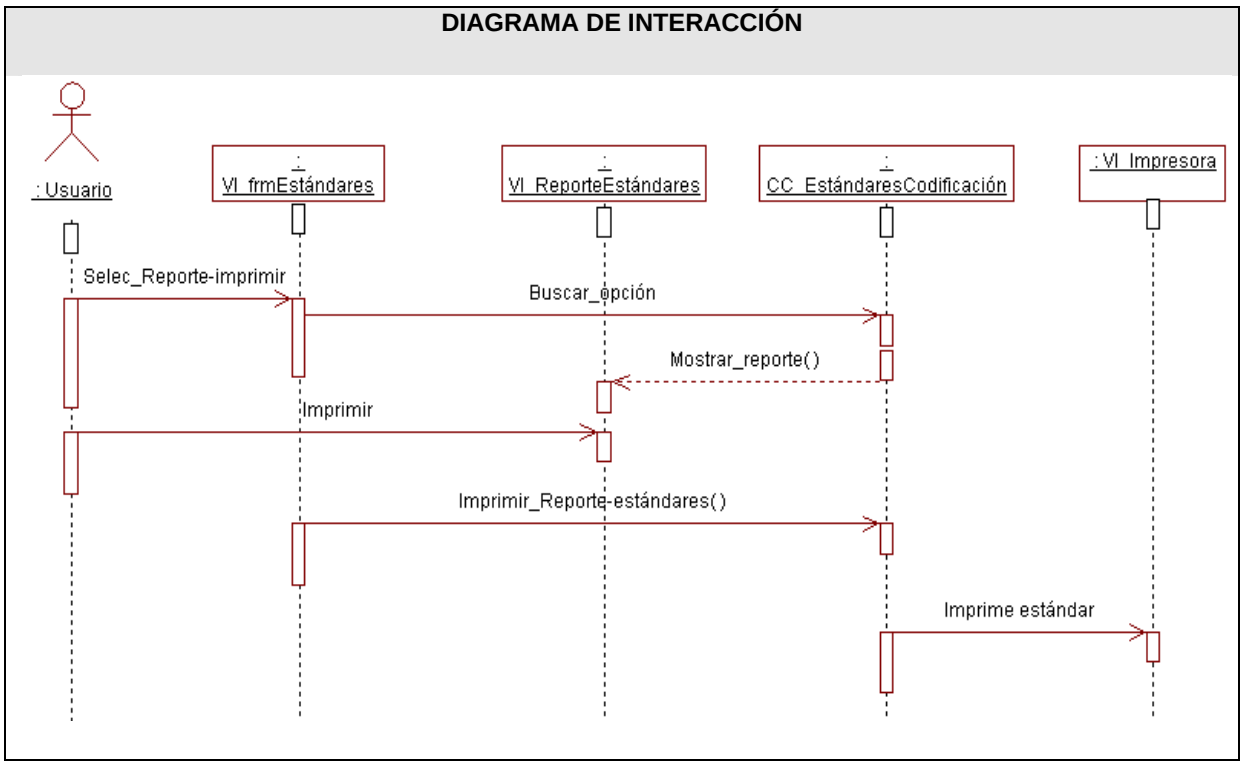


Figura A17. DCA_CU Gestionar reporte de comprobación de estándares. (Escenario: Imprimir)

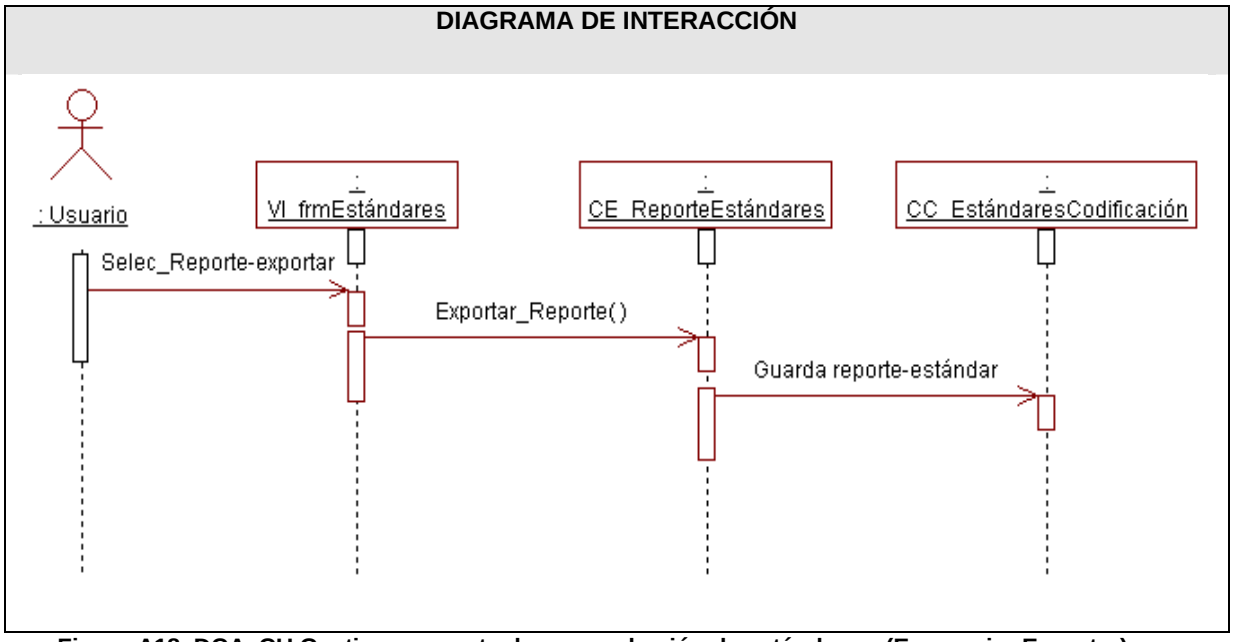


Figura A18. DCA_CU Gestionar reporte de comprobación de estándares. (Escenario: Exportar)