ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE INGENIERÍA

HERRAMIENTA PARA EVALUAR LA CALIDAD DEL CÓDIGO FUENTE GENERADO EN C ANSI.

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

AMY INDIRA CALAHORRANO NARVÁEZ

DIRECTOR: MSC. ING. CARLOS MONTENEGRO

Quito, septiembre de 2007

DECLARACIÓN

Yo, Amy Indira Calahorrano Narváez, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Amy Calahorrano

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Amy Indira Calahorrano Narváez, bajo mi supervisión.

Msc. Ing. Carlos Montenegro

DIRECTOR DE PROYECTO

DEDICATORIA

A la ciencia y a quien es un icono fundamental de la misma, Albert Einstein y a todos los científicos de mente privilegiada que nos legaron su sabiduría para el bien de la humanidad.

Amy Indira

AGRADECIMIENTOS

A la Escuela Politécnica Nacional, recinto
de prestigiosa formación académica.

A mi Carrera por ser un hito fundamental
en mi superación profesional.

A mis profesores fuente de conocimientos
invaluables y loables que supieron
compartirlos conmigo.

Un especial agradecimiento al
Msc. Ing. Carlos Montenegro
Director de Proyecto por su guía
oportuna y acertada
en la realización de esta tesis.

A mis padres por su total e incondicional apoyo.

A mis hermanos, en especial a Dheyanira
por ser ejemplo de tenacidad y firmeza
en busca de un seguro porvenir.

Y a todas aquellas personas que colaboraron
de manera intelectual, moral y espiritual
para el desarrollo de esta tesis.

CONTENIDO

ÍNDICE FIGURAS

ÍNDICE TABLAS

INTRODUCCIÓN

Obtener productos de software que tengan una alta calidad debe ser el objetivo fundamental de las empresas que se dedican al desarrollo de estos productos, para conseguir esa calidad es necesario seguir algunos procesos de evaluación.

El presente proyecto consta de cuatro capítulos que se describen a continuación: en el primer capítulo se especifica cada una de las partes que conforman el estándar de evaluación ISO / IEC 14598, se expone la métrica de complejidad ciclomática y se describe la metodología RUP que se utilizó para el desarrollo de la herramienta EVAC.

En el segundo capitulo se desarrolla la herramienta para medir la complejidad ciclomática.

En el tercer capítulo se aplica el proceso de evaluación a la aplicación EVAC, es decir, utilizar el estándar ISO/IEC 14598 haciendo referencia al modelo de calidad de la norma ISO/IEC 9126; posteriormente un análisis final de los datos obtenidos.

Finalmente en el cuarto capitulo se presentan las conclusiones y recomendaciones del presente trabajo.

CAPÍTULO 1. MARCO TEÓRICO

1.1. PROCESOS DE EVALUACIÓN DE LA CALIDAD DEL CÓDIGO FUENTE

La calidad del producto de software es el resultado de las actividades realizadas a lo largo del proceso de su desarrollo.
La búsqueda de requerimientos expresa las necesidades de los usuarios en términos cuantitativos, cualitativos y las tendencias de calidad de un producto de software.
Para determinar la calidad del software se aplica métodos y estándares.

Al generar productos de software los esfuerzos se enfocan más en el desarrollo y mantenimiento, considerando que la calidad del software se produce en la evaluación final del producto mismo.
Los métodos de evaluación ayudan a los procesos de desarrollo para mejorar la calidad del producto de software.

La serie de las normas ISO/IEC 14598 ofrece métodos para la medida, valoración y evaluación de calidad de producto de software.
El objetivo de estos métodos es proveer a los evaluadores mecanismos de soporte, para la evaluación de productos de software desde el punto de vista del usuario final [1].
Los procesos de evaluación son utilizados para simular el uso operacional normal del producto de software, comenzando por un análisis de documentación, instalando el producto como se lo especifica en la documentación y utilizando el producto de la mejor manera posible.

En Ingeniería de Software hay tres puntos importantes para la evaluación del software, estos son:

---

[1] http://www.cenpra.gov.br/publicacoes/pdf/2002/evaluation_software.pdf

- **Proceso:** es un conjunto de actividades y subprocesos utilizados para generar un producto específico.
- **Producto:** es aquel al cual se le van a agregar los procesos y será el encargado de ejecutarlos.
- **Recurso:** es una entrada requerida por un proceso para producir algún resultado o salida especificada (recursos de un proyecto pueden ser: humanos, monetarios, materiales, tecnológicos, temporales).

1.1.1. ESTÁNDAR ISO/IEC 14598

La serie de estándares ISO/IEC 14598 proporciona métodos para medida, valoración y evaluación de calidad del producto de software, pero no describen los métodos para los procesos de evaluación de la producción del software o predicciones del costo.

El estándar ISO/IEC 14598 propone las siguientes actividades para los Procesos de Evaluación, estos son:

- Revisión General (ISO/IEC 14598-1)
- Planificación y Administración (ISO/IEC 14598-2)
- Proceso para Desarrolladores (ISO/IEC 14598-3)
- Proceso para Adquisidores (ISO/IEC 14598-4)
- Proceso para Evaluadores (ISO/IEC 14598-5)
- Documentación de Módulos de Evaluación (ISO/IEC 14598-6)

1.1.1.1. Revisión General (ISO/IEC 14598-1)

Proporciona una apreciación global de las demás partes del estándar ISO/IEC 14598.

Contiene la estructura y los requisitos generales para la especificación y evaluación de la calidad del producto de software.

Adicionalmente describe el proceso de evaluación en los pasos siguientes:

- Establecer requisitos de Evaluación
- Especificar la Evaluación
- Diseñar la Evaluación
- Ejecutar la Evaluación

Estos procesos de evaluación se muestran en la Figura 1.1.

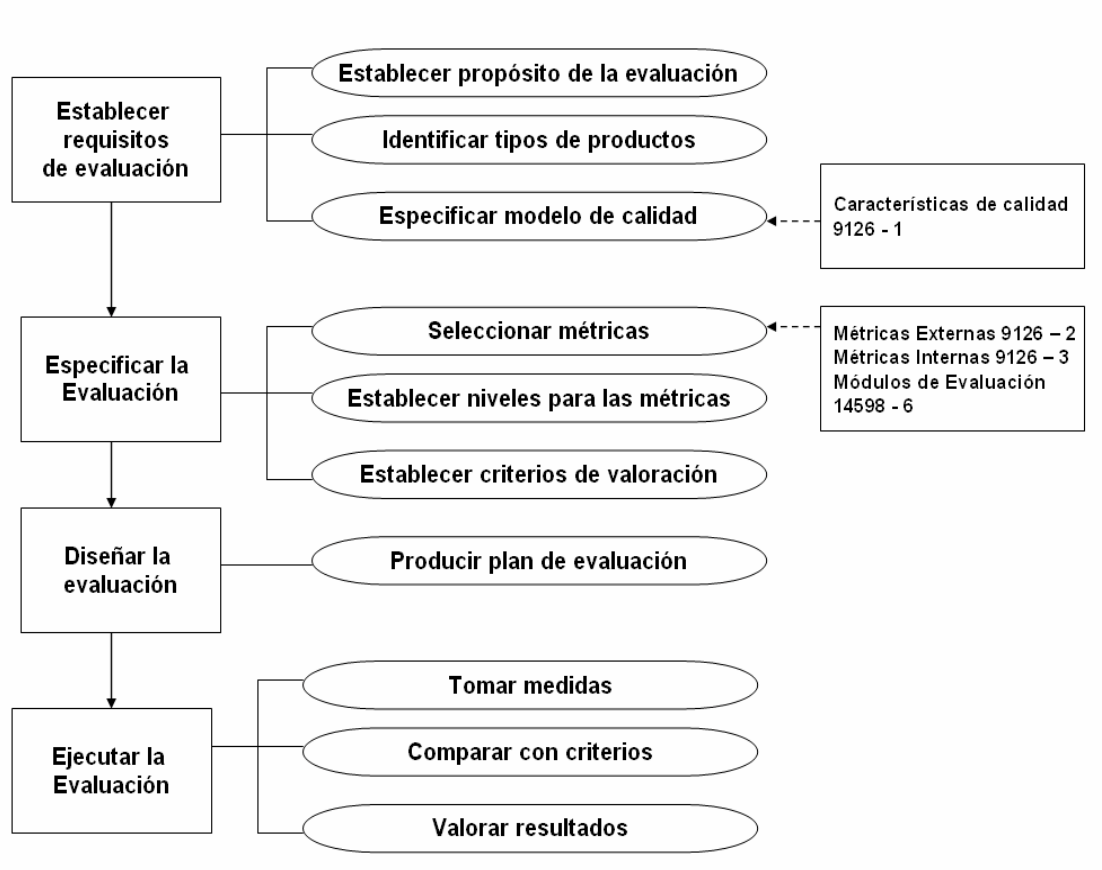


Figura 1.1 Proceso de Evaluación

Fuente: ISO/IEC 14598-1

1.1.1.1.1. Establecer requisitos de Evaluación

Establecer propósitos de la evaluación

El propósito de calidad del software es apoyar la producción y la adquisición de software que satisfaga directamente las necesidades del usuario.

El propósito de la evaluación se divide en dos grupos:

1. Evaluación de la calidad de un producto intermedio.
2. Evaluación de la calidad de un producto final.

El propósito de evaluación de calidad del producto intermedio es:

- Decidir sobre la aceptación de un producto intermedio de un subcontratista.
- Decidir sobre la realización de un proceso y cuándo enviar los productos al siguiente proceso.
- Predecir o estimar la calidad de producto final.
- Recolectar la información sobre los productos intermedios con el objetivo de controlar y administrar el proceso.

El propósito de evaluación de la calidad del producto final es:

- Decidir sobre la aceptación de los productos.
- Decidir cuándo liberar los productos.
- Comparar el producto con los productos competitivos.
- Seleccionar un producto de entre los productos alternativos.
- Evaluar el aspecto positivo y negativo de la entidad designada cuando es usada.
- Decidir cuándo mejorar o reemplazar el producto.

Identificar tipos de productos

Estos tipos de productos no significan aplicaciones de software, más bien se preocupa por la fase alcanzada en el ciclo de vida del producto, el cual determina si se ha procesado la evaluación de la calidad interna, calidad externa o calidad en uso.

El objetivo es que cuando el producto este realmente en uso, satisfaga las necesidades implícitas y explicitas del usuario, teniendo así *calidad en uso*.
La *calidad externa* puede ser evaluada solo por un sistema completo de hardware/software de la cual el producto de software es una parte. Las características externas son aquellas que se evalúan al ejecutar el software.
El Software que se ejecuta satisfactoriamente en un ambiente, pero puede mostrar defectos de calidad en otro.
Las medidas externas son a menudo sólo indicadores de la calidad real en uso.

Para el propósito de desarrollo, los requerimientos de *calidad interna* son definidos como aquellos que permiten verificar la calidad de los productos intermedios. Las características internas son aquellas que se evalúan observando los rasgos internos del software.

Las medidas externas de un sistema de computación pueden ser usadas también como una medida indirecta de la calidad interna del software.

Especificar un modelo de calidad

Para evaluar el software es necesario seleccionar las características de calidad relevantes, para lo cual se puede usar el estándar ISO/IEC 9126-1 como una guía, el cual define seis categorías de calidad de software: funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad.
Cabe recalcar que las características y subcarecterísticas pertinentes dependerán del propósito de la evaluación y deberán identificarse previo estudio de los requisitos de la calidad.

1.1.1.1.2. Especificar la evaluación

Se involucra tres pasos para la especificación de la evaluación, estos son:

- Selección de métricas.
- Establecer niveles para las métricas.
- Establecer criterios de valoración.

Selección de métricas

Cada característica cuantificable del software y cada interacción cuantificable con su ambiente que se correlaciona con una característica, puede ser establecida como una métrica.

Las métricas pueden diferir dependiendo del ambiente y la fase del proceso de desarrollo en el cual se encuentren.

Establecer niveles para métricas

Las características cuantificables pueden ser medidas cuantitativamente usando métricas de calidad.

El valor medido es colocado en una escala de la siguiente manera:

- Dividiendo la escala en dos categorías: poco satisfactorio y satisfactorio.
- Dividiendo la escala en cuatro categorías: limitado por el nivel actual, un nivel existente (o valor medido), el peor nivel del caso, nivel planeado.

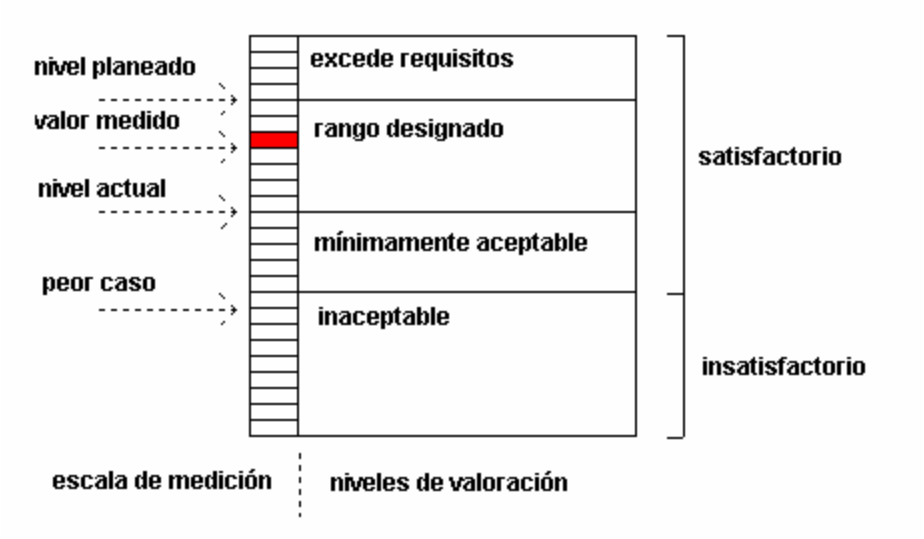


Figura 1.2 Niveles de valoración para las métricas

Fuente: ISO/IEC 14598-1

Establecer criterios de valoración

Para evaluar la calidad del producto, los resultados de las evaluaciones de las diferentes características tienen que ser sumarizados. El evaluador es el encargado de preparar un procedimiento, el cual involucra criterios separados para las características de calidad diferentes, donde cada una de ellas pueden estar expresadas en términos de subcaracterísticas individuales, o una combinación ponderada de ellas. Generalmente, el procedimiento esta compuesto por otros aspectos como tiempo y costo que contribuyen a la valoración de calidad de un producto del software en un ambiente determinado.

1.1.1.1.3. Diseñar la evaluación

Producir un plan de evaluación

El plan de evaluación describe los métodos de evaluación y el cronograma de acciones del evaluador (ISO/IEC 14598-5). Un estudio más amplio de esta parte se la realiza en la norma ISO/IEC 14598-2.

1.1.1.1.4. Ejecutar la evaluación

Toma de medidas

Al aplicar las métricas seleccionadas al producto de software se obtienen las medidas. Los resultados de las medidas son valorados sobre la escala de métricas.

Criterios de comparación

Los criterios de comparación pueden ser tomados desde los niveles de valoración (ver Figura 1.2), definidos por los valores medidos y ser comparados entre ellos.

Evaluar los resultados

La evaluación de los resultados es el último paso del proceso de evaluación del software, donde un conjunto de niveles valorados son sumarizados. Dando como resultado una perspectiva de hasta que punto el producto del software reúne los requisitos de calidad.

1.1.1.2. Planificación y Administración (ISO/IEC 14598-2)

Esta parte de la norma contiene requisitos y guías para las funciones de soporte, como es la planificación y administración para la evaluación de productos de software.

Para las funciones de soporte existe un departamento designado para ello, el cual provee la tecnología necesaria para la evaluación del producto de software.

Es importante que este departamento capacite a su personal para realizar las actividades de evaluación de manera óptima, generando métodos, normas y documentos de evaluación.
De esta manera podrá ayudar a las organizaciones en los proyectos de desarrollo de software y a los que deseen realizar evolución de los mismos, así como brindar criterios para la adquisición del software.

Las principales funciones del departamento de soporte son:

- Adquisición de estándares nacionales e internacionales, información técnica y soporte de expertos.
- Desarrollo de estándares internos y herramientas, en base a los requisitos de organizaciones o proyectos desarrollados.
- Desarrollar criterios para la evaluación.
- Revisar la efectividad y calidad de adquisiciones o desarrollos de software.
- Analizar los resultados de la evaluación dentro de la organización.

Las organizaciones deben crear políticas y planes para las actividades de evaluación. Las actividades de evaluación definen las responsabilidades de cada departamento dentro de la organización.

Un plan para mejorar la evaluación de software incluye:

- Definición de los objetivos de la organización.
- Definición de políticas.
- Identificación de las técnicas a ser utilizadas.
- Asignación de responsabilidades para los administradores de evaluación de procesos.
- Analizar los resultados obtenidos con el objetivo de mejorar los futuros procesos de evaluación del software.

El departamento de soporte debe supervisar que las fases del proyecto de evaluación se estén cumpliendo dentro de los tiempos establecidos.

1.1.1.3. Proceso para Desarrolladores (ISO/IEC 14598-3)

ISO/IEC 14598-3 proporciona una guía para esclarecer los requisitos para la implementación y análisis de las medidas de la calidad de software.
Aquí se define las actividades necesarias para definir los requisitos, especificación, diseño y conclusiones de la evaluación de cualquier tipo de producto de software, brindando soporte al desarrollador al evaluar el producto durante el ciclo de vida de desarrollo, a través de la identificación de atributos de productos intermedios y el desarrollo de actividades para medir estos atributos.
La norma se enfoca en la selección de indicadores que son útiles para predecir la calidad del producto final a través de la calidad de productos intermedios.

El uso de indicadores de calidad permite al desarrollador identificar los posibles problemas de calidad tempranamente y realizar las acciones correctivas.

El proceso de evaluación para desarrolladores comprende un conjunto de actividades ejecutadas por el desarrollador y que son realizadas en base a los valores de mediciones obtenidas durante el proceso de desarrollo.

Estas actividades son:

- Establecer los requisitos de evaluación, la cual identifica las necesidades de los usuarios (requisitos de calidad general) y que deben estar de acuerdo al modelo de calidad seleccionado (ISO/IEC 9126).

- Especificación de la evaluación, consiste en la determinación de métricas externas y métricas internas.

- Diseño de la evaluación, consiste en planificar acciones para la recolección de datos.

- Ejecución de la evaluación, consiste en la recolección de valores de medidas internas, externas y compararlas con los valores objetivos (evaluación durante el desarrollo). Los valores de los atributos internos (indicadores de calidad) son usados para estimar la calidad del producto final.

- Retroalimentación de las organizaciones, se encarga de la revisión de los resultados de la evaluación.

1.1.1.4. Proceso para Adquisidores (ISO/IEC 14598-4)

ISO/IEC 14598-4 proporciona requisitos, recomendaciones, además de ser una guía para la evaluación y valoración de la calidad del producto de software en su adquisición.
El estándar ISO/IEC14598 clasifica a los productos de software en tres grupos:

- Productos de Software Comerciales.
- Productos del software desarrollados o adquiridos por otras organizaciones.
- Productos de Software Personalizados o productos de Software modificados.

1.1.1.5. Proceso para Evaluadores (ISO/IEC 14598-5)

El estándar ISO/IEC 14598-5 define los subprocesos necesarios para analizar los requisitos, especificaciones, diseños y ejecuciones de la evaluación, obteniendo así conclusiones y recomendaciones para cualquier tipo de software.

Este estándar se puede usar para:

- Evaluar productos existentes.

- Evaluar productos en desarrollo.

El proceso de evaluación consiste de un conjunto de tareas cooperativas e interacciones entre el solicitante (representado por un desarrollador, un usuario del software, un proveedor o adquisidor de software) y el evaluador (representado por un laboratorio u organización destinada a evaluar software). Al ejecutar los subprocesos, se generan documentos, especificaciones, y demás elementos que pueden ser utilizados como entradas a otras actividades, o ser el resultado del proceso de evaluación.

CARACTERÍSTICAS DEL PROCESO DE EVALUACIÓN

De acuerdo a la norma ISO/IEC 14598 las características esperadas en los Procesos de Evaluación del Software son:

- Repetible
- Reproducible
- Imparcial
- Objetiva

Repetible: La evaluación repetida de un mismo producto, realizada bajo la misma especificación y con el mismo evaluador, debe producir resultados que pueden aceptarse como idénticos.
Reproducible: La evaluación del mismo producto, con la misma especificación de evaluación, pero ejecutada por un evaluador diferente, debe producir resultados que pueden aceptarse como idénticos.
Imparcial: La evaluación no debe ser influenciada en comparación con cualquier otro resultado en particular.
Objetiva: Los resultados de la evaluación deben ser verdaderos, y no ser influenciados por las opiniones o sentimientos del evaluador.

El proceso de evaluación según el estándar ISO/IEC 14598 esta compuesto de cinco subprocesos, estos son:

a) Establecimiento de los Requisitos de Evaluación
b) Especificación de la Evaluación
c) Diseño de la Evaluación
d) Ejecución de la Evaluación
e) Conclusión de la Evaluación

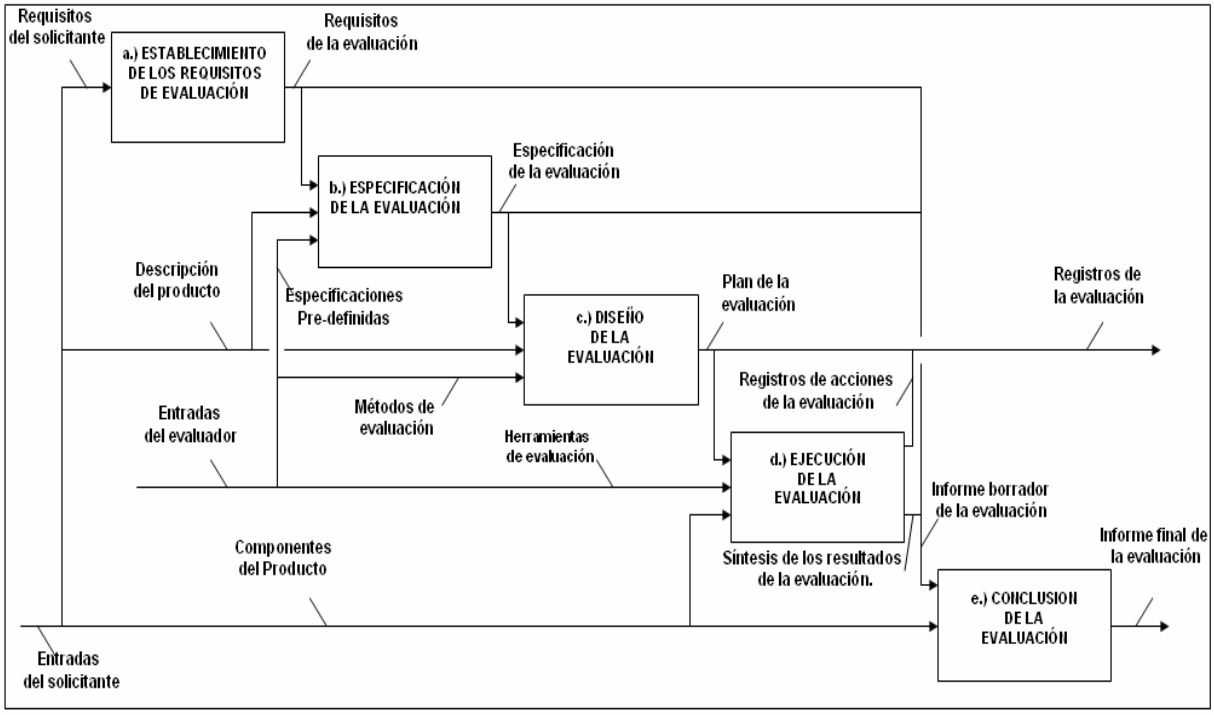


Figura 1.3 Proceso de Evaluación para Evaluadores

Fuente: ISO/IEC 14598-5

a) Establecimiento de los requisitos de evaluación

Este proceso describe los objetivos de la evaluación que se relacionan con el uso del producto de software.

El solicitante elabora un documento de requisitos donde se debe expresar la importancia de cada característica, todo esto en base al grado de cobertura y objetivos de la evaluación.

b) Especificación de la evaluación

Este proceso define el alcance de la evaluación y las mediciones a realizarse en el producto y componentes.
Al elaborar la especificación se debe analizar la descripción del producto, especificar mediciones y verificar las especificaciones en base a los requisitos de evaluación.
El documento de especificación de la evaluación debe contener el alcance de la evaluación, una relación entre la información para realizar la evaluación y los componentes del producto, una relación entre la especificación de las mediciones y verificaciones, y el documento de especificación de requisitos.

c) Diseño de la evaluación

Se encarga de documentar los métodos y procedimientos, optimizar el plan de evaluación y programar las actividades de acuerdo a los recursos disponibles, es así, que el evaluador genera el plan de la evaluación que describe los recursos necesarios (humanos, materiales, tecnológicos, etc.) y su adecuada distribución y asignación en las actividades.
Este documento se incluirá en los registros de evaluación y la documentación de los métodos de evaluación.

d) Ejecución de la evaluación
Se encarga de obtener los resultados al ejecutar las actividades programadas, conforme a los requisitos de evaluación.

En base a esta ejecución se generan dos documentos, el registro de evaluación y el borrador del informe de evaluación.

e) Conclusión de la evaluación

Consiste en la revisión del borrador entre las partes (solicitante y evaluador) y hacer disponibles los documentos finales.

Tanto el documento de requisitos, especificación, diseño, ejecución y conclusión de la evaluación generarán su respectivo borrador, el cual será revisado conjuntamente con el solicitante y el evaluador para su aprobación, e ir construyendo el informe final de evaluación.

1.1.1.6. Documentación de Módulos de Evaluación (ISO/IEC 14598-6)

En esta parte de la norma donde se define la estructura y el volumen de la documentación, es decir, se elige un formato para la documentación de un Módulo a evaluar. Los Módulos de Evaluación son usados en las normas ISO/IEC 9126 e ISO/IEC14598.

Un Módulo de evaluación: es un paquete de tecnología de la evaluación para medir características de la calidad del software, subcaracterísticas o atributos.[2]

El paquete incluye:

- Métodos y técnicas de evaluación.
- Entradas para la evaluación.
- Recolección de Datos a ser medidos.
- Procedimientos y herramientas de soporte.

[2] ISO/IEC 14598-6 Pág. 2

1.2. MÉTRICA DE COMPLEJIDAD CICLOMÁTICA

1.2.1. INTRODUCCIÓN

Métrica que se deriva de la teoría de grafos, propuesta por Thomas McCabe en 1976.

El número de ciclos fundamentales de conexión de los diagramas de flujo da paso al término ciclomática.

La complejidad ciclomática es una métrica del software para evaluar de manera cuantitativa la complejidad lógica de un programa, siendo utilizada como un indicador de la confiabilidad y mantenimiento máximo de un módulo o tamaño de un procedimiento.

La complejidad ciclomática es el valor que define el número de caminos independientes para la ejecución completa de un programa y proporciona un número límite para las pruebas donde cada sentencia se ejecute por lo menos una vez

Un camino independiente es un camino de ejecución del programa donde se incluye un nuevo conjunto de sentencias de proceso o una nueva condición. [3]

La medida resultante puede ser utilizada en el desarrollo, mantenimiento y reingeniería para estimar el riesgo, costo y estabilidad de los programas de software.

Algunos estudios experimentales indican 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 límite como se observa en la Tabla 1.1

[3] Presuman, R: Ingeniería del Software. Un enfoque Práctico, 2005

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

Tabla 1.1 Complejidad ciclomática vs. Evaluación de riesgo

Fuente: RIZZI-COMPLEJIDAD.PDF

1.2.2. ÁMBITO DE UTILIZACIÓN DE LA COMPLEJIDAD CICLOMÁTICA

La complejidad ciclomática puede ser aplicada en varias áreas como:

- **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 cuando se lo modifica. 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.

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 se encontró que un valor de 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.

1.2.3. DEFINICIÓN DE COMPLEJIDAD CICLOMÁTICA

La complejidad ciclomática de un grafo de flujo G (V(G)), se puede calcular de tres maneras:

$$V(\mathbf{G}) = A \quad \textbf{(1)}$$

Donde:

A: Número de áreas del grafo de flujo, incluyendo el área exterior del grafo como una región más.

$$V(\mathbf{G}) = E - N + 2 \quad \textbf{(2)}$$

Donde:

E: Número de aristas o enlaces entre nodos del grafo

N: Número de nodos del grafo

$$V(\mathbf{G}) = P + 1 \quad \textbf{(3)}$$

Donde:

P: Número de nodos predicado, es decir aquel nodo del cual emergen dos o más aristas

Entonces la complejidad ciclomática de la Figura 1.4 de acuerdo a las tres fórmulas anteriores sería:

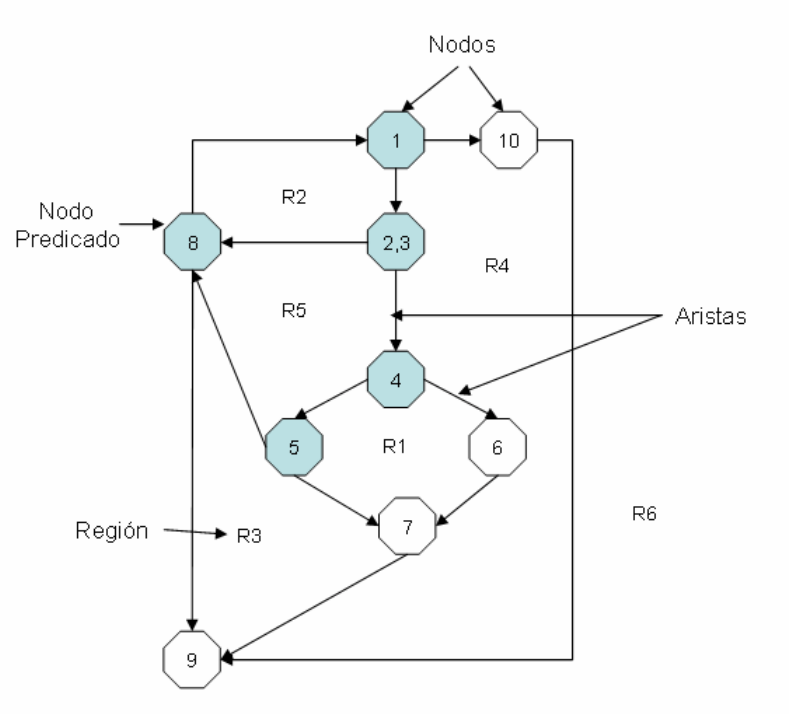


Figura 1.4 Grafo de Flujo

Fuente: El autor

1. V(**G**) = 6 regiones
2. V(**G**) = 13 aristas - 9 nodos + 2 = 6
3. V(**G**) = 5 nodos predicado +1 = 6

Entonces la complejidad ciclomática del grafo de flujo de la Figura 1.4 es 6.

Por ende este valor nos indica el número de pruebas que se deben diseñar para asegurar que se ejecuten todas sentencias del programa.

1.2.4. NOTACIÓN DE FLUJOS DE CONTROL DE UN PROGRAMA

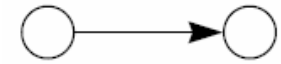
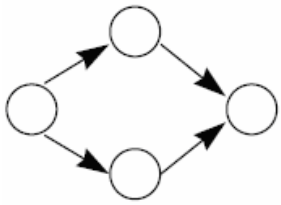
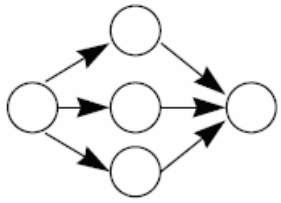
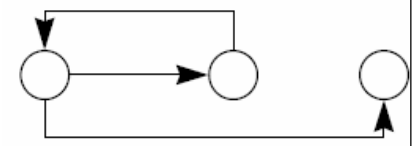
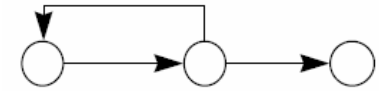
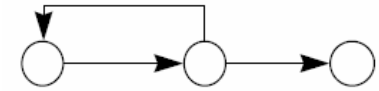
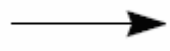
SENTENCIA	ESTRUCTURA EN FORMA DE GRAFO DE FLUJO	DEFINICIÓN
		Un paso simple de una instrucción a otra.
IF (Condición)		Donde se debe cumplir una condición para ejecutar un conjunto de sentencias y si no cumple con esa condición, ejecuta otro conjunto de sentencias.
CASE (Selección Múltiple)		Permite tener varias alternativas de selección.
WHILE (Bucle)		Especifica que mientras se cumple una condición establecida, se ejecuta una o varias sentencias
DO-WHILE (Bucle)		Especifica que se ejecute una o varias sentencias al menos una vez, mientras se cumpla la condición establecida.
FOR (Bucle)		Especifica que se ejecute cero o más las sentencias que se encuentran dentro del bucle, hasta que se cumpla la condición establecida.

Tabla 1.2 Indica la sentencia, estructura y definición de los flujos en un grafo

Sentencia de programa



Paso a la siguiente sentencia

1.3. METODOLOGÍA PARA EL DESARROLLO DE LA HERRAMIENTA

1.3.1. RUP (RATIONAL UNIFIED PROCESS)

Rational Unified Process esta basado en el trabajo integrado de tres especialistas, Ivar Jacobson, Grady Booch y James Rumbaugh. Estos especialistas con ayuda de una comunidad amplia de investigadores fueron congregando la Rational Corporation para formar una metodologia unificada, cohesiva y comprensiva para el desarrollo de sistemas de software.

RUP maneja una estructura bien definida, permite una aproximación a la programación orientada a objetos (POO) para su descripción. Su estructura está determinada por fases y flujos de trabajo, permitiendo una fácil navegación dentro de su arquitectura.

1.3.1.1. Arquitectura de RUP

RUP tiene un modelo orientado a objetos donde se utiliza fundamentalmente Unified Model Language (UML). La Figura 1.5 muestra la arquitectura global del Rational Unified Process. El proceso tiene dos dimensiones, estas son:

- *Dimensión horizontal:* representa tiempo y muestra aspectos dinámicos del ciclo de vida del proceso, es decir lo que se refiere a ciclos, fases, iteraciones e hitos.
- *Dimensión vertical:* representa los aspectos estáticos del proceso, el cual esta descrito por las actividades, artefactos, trabajadores y flujos de trabajo.

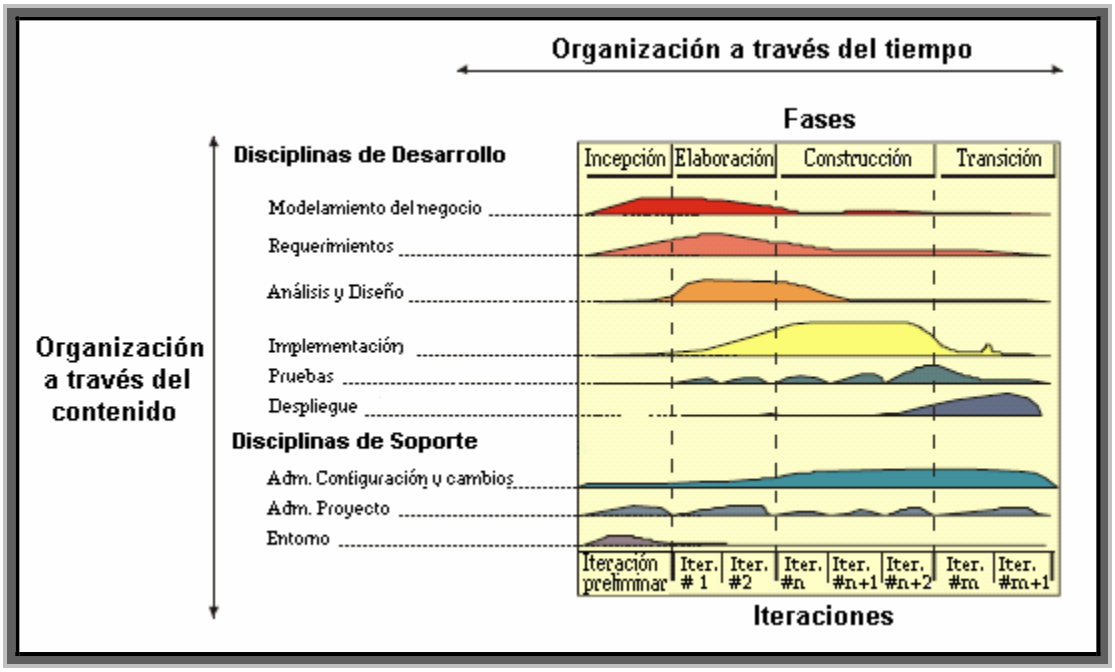


Figura 1.5Gráfico del Modelo iterativo, indica como el proceso se estructura a lo largo de dos dimensiones.

Fuente: rup_bestpractices.pdf

1.3.1.2. Las Mejores Prácticas de RUP

Rational Unified Process (RUP) captura muchas de las mejores prácticas de desarrollo de software moderno en una forma adecuada para una gama amplia de proyectos y organizaciones:

- Desarrollo de software iterativo
- Gestión de requerimientos
- Uso de arquitecturas basadas en componentes
- Modelo visual de software
- Verificación de la calidad de software
- Control de cambios de software

1.3.1.2.1. Desarrollo de Software Iterativo

Con los sistemas de software sofisticados de hoy es imposible definir secuencialmente primero el problema entero, el diseño de la solución, la construcción del software y finalmente probarlo. Es menester un acercamiento iterativo que permita una comprensión creciente del problema, a través de los refinamientos sucesivos, y para generar incrementalmente una solución eficaz sobre las múltiples iteraciones.

RUP soporta esta característica de acercamiento iterativo para el desarrollo, el cual se dirige a los ítems de alto riesgo en cada fase del ciclo de vida, reduciendo significativamente el perfil de riesgo del proyecto.

Este acercamiento iterativo ayuda a contrarrestar el riesgo, a través de la demostración del progreso frecuente, versiones ejecutables que permiten una continua relación con el usuario final y una retroalimentación, ya que cada iteración finaliza con una versión ejecutable, el equipo de desarrollo esta enfocado en producir soluciones, y frecuentemente verifican el estado, lo cual ayuda a asegurar que el proyecto cumpla con el cronograma establecido.

Debido al desarrollo iterativo se hace mucho más fácil adaptar cambios tácticos en los requerimientos, características y cronograma.

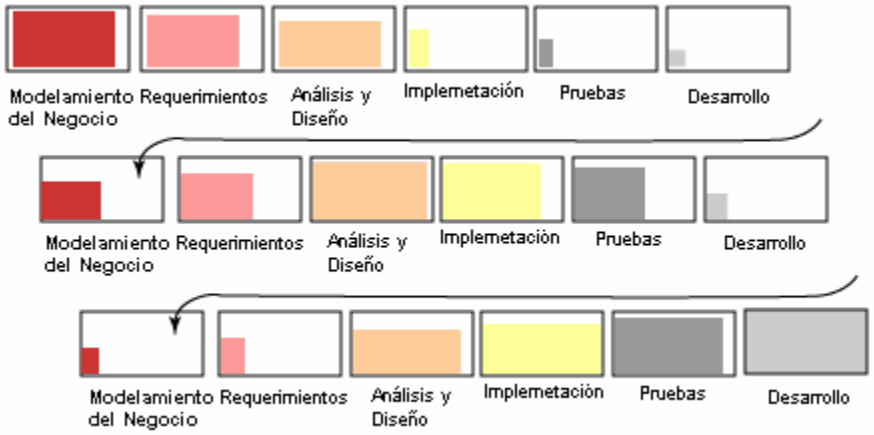


Figura 1.6 Proceso de Desarrollo Iterativo RUP

Fuente: rup_bestpractices.pdf

1.3.1.2.2. Gestión de Requerimientos

RUP describe sistemáticamente la forma de obtener, organizar y documentar funcionalidades requeridas y sus restricciones; además de capturar y comunicar fácilmente los requerimientos del negocio.

La intención de usar los casos de uso y escenarios descritos en el proceso, han demostrado ser una manera excelente para obtener los requisitos funcionales y asegurar que éstos manejan el diseño, aplicación y pruebas del software, de tal manera que el sistema final cumpla con todas las necesidades del usuario.

Los diagramas de casos de uso y escenarios facilitan el seguimiento en cuanto al desarrollo y a los entregables del sistema.

1.3.1.2.3. Uso de arquitecturas basadas en componentes

El proceso está enfocado en un desarrollo temprano, en una base de una arquitectura ejecutable y robusta, antes de comprometer los recursos para un desarrollo completo.

Describe como diseñar una arquitectura que sea flexible, que se acomode a cambios, que sea intuitivamente comprensible, y promueva la reutilización del software más eficazmente.

1.3.1.2.4. Modelamiento visual del software

Los modelos son simplificaciones de la realidad; ellos nos ayudan a entender el problema y formar una solución, además es útil para comprender sistemas grandes y complejos que nosotros no podríamos entenderlo como un solo conjunto.

Unified Modeling Language (UML) es un lenguaje grafico para visualización, especificación, construcción, y documentación de los artefactos de un sistema de software, además proporciona los medios necesarios para describir las características del software, cubre los ítems conceptuales, tales como, los procesos

del negocio y funciones del sistema, así como también ítems específicos como, clases escritas en un lenguaje de programación determinado, esquemas de bases de datos, y componentes del software reusables. UML provee un vocabulario para expresar los distintos modelos, mas no indica como desarrollar el software, es por ello que Rational desarrollo RUP, una guía para el uso eficaz de UML en el modelamiento, es decir describe los modelos que se necesitan, porque se necesitan, y como construirlos.

1.3.1.2.5. Verificación de la calidad de software

La calidad del software debería revisarse con respecto a los requerimientos basados en la fiabilidad, funcionalidad, ejecución de la aplicación y del sistema.
RUP ayuda en la planificación, diseño, implementación, ejecución, y evaluación de estos tipos de prueba, para verificar la calidad del software.
La evaluación de la calidad es construida dentro del proceso, en todas las actividades, involucrando a todos los participantes, usando medidas y criterios objetivos, y no tratándolo como una ejecución de una actividad totalmente separada.

1.3.1.2.6. Control de cambios de software

Todo proyecto de desarrollo de software iterativo, siempre exige modificaciones. RUP define métodos para controlar, cuantificar y monitorear estos cambios. RUP también define espacios de trabajo seguro, garantizando un sistema de ingeniería de software que no será afectada por cambios en otros sistemas.

1.3.1.3. Fases del Ciclo de vida del proyecto

Cada ciclo de vida del proyecto, trabaja sobre una nueva generación del producto. RUP divide un ciclo de desarrollo en cuatro fases:

- Inicio

- Elaboración
- Construcción
- Transición

Cada fase concluye con un punto bien definido (o hito), en el cual se deben tomar ciertas decisiones críticas a tiempo, y por consiguiente se lograrán las metas más importantes.

1.3.1.3.1. Fases de Inicio

En la fase de inicio se establece los casos del negocio para el sistema y se delimita el alcance del sistema. Para ello es necesario identificar las entidades externas con las cuales el sistema interactúa (actores) y definir la naturaleza de esta interacción a un alto nivel.

1.3.1.3.2. Fases de Elaboración

En esta fase se analiza el dominio del problema, establecer una arquitectura, desarrollar el plan del proyecto y eliminar los altos riesgos del mismo, además se prepara al personal y se presupuesta el costo de la totalidad del proyecto.

1.3.1.3.3. Fases de Construcción

En la fase de construcción, todos los componentes y características de la aplicación faltantes son desarrollados e integrados dentro del producto y son probadas en su totalidad.

Esta fase se enfoca en los procesos de fabricación donde se pone énfasis en los recursos administrativos y se controlan las operaciones para optimizar costos, horarios y calidad.

1.3.1.3.4. Fases de Transición

En la fase de transición, el producto de software es entregado al usuario final, y es aquí cuando los problemas aparecen, lo cual requiere el desarrollo de nuevas versiones, corrección de algunos problemas, o agregar características finales que fueron propuestas.

1.3.1.4. Disciplinas del ciclo de vida del proyecto

RUP esta basado en un conjunto de bloques, elementos, descripciones de lo que se va a producir, las habilidades requeridas y una explicación paso a paso de como lograr las metas de desarrollo especificadas.
El ciclo de vida de RUP organiza las tareas dentro de fases e iteraciones, en cada iteración las tareas se categorizar en nueve disciplinas.
Estas disciplinas son:

- Disciplinas de Desarrollo
    1. Modelamiento del negocio
    2. Requerimientos
    3. Análisis y Diseño
    4. Implementación
    5. Pruebas
    6. Despliegue

- Disciplinas de soporte
    7. Administración de configuración y cambios
    8. Administración del proyecto
    9. Entorno

1.3.1.4.1. Disciplinas de Desarrollo

Modelamiento del negocio

En el modelamiento del negocio se documenta los procesos del negocio usando los llamados casos de uso del negocio. El objetivo del modelamiento es establecer un entendimiento y comunicación entre la ingeniería del negocio y la ingeniería del software, manteniendo un conocimiento común entre el objetivo de la organización, los usuarios y los desarrolladores.

Requerimientos

La disciplina de requerimientos describe lo que el sistema debería hacer y permitir a los desarrolladores y clientes estar de acuerdo con esta descripción. Para lograr esto, se obtiene, organiza y documenta las funcionalidades y restricciones requeridas e intercambiando documentos y decisiones.

Se crea un documento de visión, se identifica actores, se representan los usuarios, y cualquier otro sistema que pueda interactuar recíprocamente con el sistema que se este desarrollando.

Cada caso del uso se describe en detalle. La descripción del caso de uso indica paso a paso cómo el sistema interactúa con los actores y lo que el sistema ejecuta.

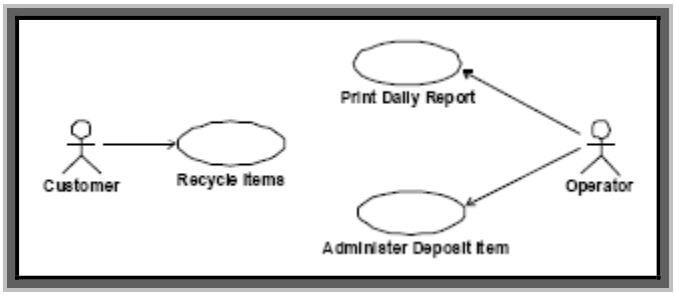


Figura 1.7 Ejemplo de modelo de casos, actores y casos de uso

Fuente: rup_bestpractices.pdf

Análisis y diseño

El objetivo de esta disciplina es indicar como el sistema será generado en la fase de implementación. Además se crea un modelo de diseño y opcionalmente un modelo de análisis. El modelo de diseño contiene clases estructuradas dentro de un paquete de diseño y un subsistema de diseño con interfaces bien definidas; representando lo que será los componentes de la aplicación; también contiene descripciones de cómo los objetos del diseño de clases, colaboran para interpretar los casos de uso.

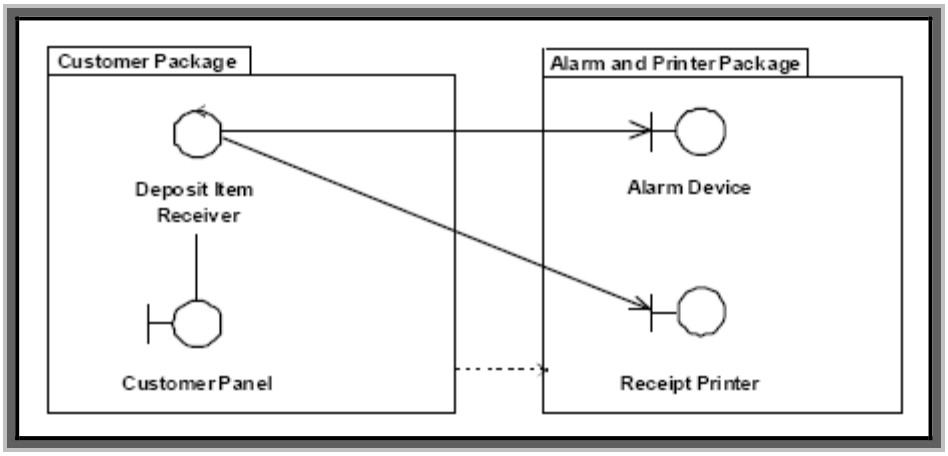


Figura 1.8 Parte de modelo de diseño con las clases del diseño que se comunican

Fuente: rup_bestpractices.pdf

Implementación

El propósito de la implementación es:

- Definir la organización del código, en términos de implementación de subsistemas estructurados en capas.
- Implementar clases y objetos en términos de componentes.
- Probar los componentes desarrollados como unidades.
- Integrar los resultados producidos por implementaciones individuales (o equipos) dentro de un sistema ejecutable.

RUP describe como rehusar los componentes existentes, o implementar nuevos componentes con responsabilidades bien definidas, haciendo al sistema fácil de mantener, e incrementando las posibilidades de reutilización de código.

Pruebas

El propósito de las pruebas es:

- Verificar la interacción entre objetos.
- Verificar la integración apropiada de todos los componentes del software.
- Verificar que todos los requerimientos hayan sido correctamente implementados.
- Asegurar que los errores se identifiquen y solucionen antes de entregar el software al usuario final.

RUP propone desarrollo iterativo, lo que significa que se hacen pruebas a lo largo del proyecto. Esto permite identificar defectos tan pronto como sea posible, lo que reduce los costos de reparación del defecto. Las pruebas son llevadas a cabo a lo largo de las tres dimensiones de calidad: confiabilidad, funcionalidad, rendimiento de la aplicación.

Despliegue

El objetivo de esta disciplina es la de generar versiones satisfactorias del producto y entregar el software a los usuarios finales, además cubre una gama amplia de actividades como:

- Producción de versiones externas del software
- Empaquetamiento, distribución e instalación del software
- Provee soporte y asistencia a usuarios

1.3.1.4.2. Disciplinas de Soporte

Administración de configuración y de cambios

El objetivo de esta disciplina describe como controlar los numerosos artefactos producidos por varias personas que trabajan sobre un proyecto en común. El control ayuda a evitar confusión a nivel de costos, y asegura que los artefactos resultantes no estén en conflicto por algunos de los siguientes problemas:

- Actualización simultánea: cuando dos o mas trabajadores realizan cambios separadamente sobre el mismo artefacto; el último en guardar los cambios destruye el trabajo del anterior.
- Notificación simultánea: cuando un problema es arreglado en artefactos compartidos por varios desarrolladores, y algunos de ellos no están notificando los cambios.
- Versiones múltiples: la mayoría de grandes programas son desarrollados en versiones que van evolucionando. Una versión podría estar en uso por algún cliente, mientras otra esta en prueba, y una tercera esta todavía en desarrollo.

Administración de proyecto

La administración de proyectos de software es el arte de equilibrar los objetivos, competentes, administrar los riesgos, superar las restricciones para entregar exitosamente un producto, que satisfaga las necesidades de los clientes (los que pagan el dinero) y usuarios.

Esta disciplina se enfoca principalmente en aspectos específicos de un proceso de desarrollo iterativo, proveyendo una arquitectura para la administración de proyectos de software intensivo, guías practicas para planificación, ejecución y monitoreo de proyectos y una arquitectura para la administración de riesgos.

Entorno

La disciplina de entorno provee un adecuado ambiente de desarrollo de software a la organización; además de los procesos y herramientas necesarias para ayudar al equipo de desarrollo.

El entorno se enfoca en las actividades para configurar los procesos dentro del contexto de un proyecto, a través del uso de pautas, plantillas y herramientas, para personalizar los procesos.

1.3.1.5. Los elementos del RUP

Actividades: Son los procesos que se llegan a determinar en cada iteración.
Trabajadores: Personas o entes involucrados en cada proceso.
Artefactos: Documentos, un modelo, o elemento de un modelo.

1.3.2. JUSTIFICACIÓN PARA EL USO DE LA METODOLOGÍA RUP

RUP es un framework del proyecto que describe una clase de procesos que son iterativos e incrementales como:

- Manejar proyectos a través de muchas iteraciones pequeñas, cada una de las cuales involucra más o menos el mismo tipo de actividades.
- Tiene cuatro hitos mayores los cuales marcan los límites entre las cuatro fases mayores.
- Genera software, y posiblemente otros entregables.
- Se estima y planea en base a la medida del progreso real.
- Es adaptable a cambios.

Ventajas

- Las actividades son concretas y repetibles.
- Se utiliza UML como herramienta de modelamiento para el análisis y diseño de la aplicación.
- Es comúnmente utilizada en proyectos de desarrollo de software.

Por las características y ventajas anteriormente descritas, se utilizará RUP para el desarrollo de la herramienta.

CAPÍTULO 2. DESARROLLO DE LA HERRAMIENTA PARA MEDIR LA COMPLEJIDAD CICLOMÁTICA

2.1. ANÁLISIS

2.1.1. DEFINICIÓN DEL PROBLEMA

El problema de generar programas sin una estructura óptima, afecta en gran medida a los sistemas computacionales, por el alto consumo de sus recursos, como la memoria, el espacio en disco, etc.

Esto produce insatisfacción por parte del usuario al utilizar estos programas, debido a su tiempo de respuesta tardía.

Una solución satisfactoria a este problema, sería la disponer de una herramienta que permita evaluar la calidad del código fuente, indicando el nivel de complejidad y su estructura lógica (la cual define el número de caminos independientes para la ejecución completa de un programa), para así saber de que manera se le puede dar un buen mantenimiento mejorando su eficiencia.

2.1.2. VISIÓN GENERAL DE LA HERRAMIENTA

Desarrollar una herramienta de software, que con la ayuda de la métrica de complejidad ciclomática se pueda determinar la calidad del código fuente, mediante el uso de grafos de flujo (Ver Figura 1.4) y un valor numérico, indicadores de la calidad del código fuente.

Además la herramienta también permitirá crear, abrir, editar, guardar el programa (archivo); abrir, guardar los reportes.

2.1.3. LEVANTAMIENTO DE REQUERIMIENTOS

Identificar cuales son las necesidades de los usuarios, para definir los requerimientos que se desarrollarán en la aplicación, y de esta manera proveer un alto nivel satisfacción en el manejo de la herramienta.

2.1.3.1. Requisitos de la aplicación

Para la herramienta se han establecido los siguientes requisitos:

Administrar Archivo

- Los usuarios podrán abrir, crear, editar, guardar los archivos.
- Para abrir los archivos se debe verificar que el archivo tenga extensión .c.
- Permitirá ejecutar la evaluación de la calidad del código fuente de los archivos, para ello es necesario que primero se realice un análisis léxico-sintáctico del código fuente, para verificar que este sea correcto.
- Una vez ejecutada la evaluación, se presentará un resultado, el cual consiste de un grafo de flujo de las sentencias control para conocer su estructura dentro del archivo, un valor de la complejidad ciclomática que indica el nivel de calidad del código fuente, la sentencia de control seguido de la secuencia de nodos que la conforman y el nombre de la función donde se encuentran declaradas, estos resultados formarán parte del reporte.

Administrar Reporte

1. Guardar los reportes en archivos .pdf, Este reporte esta compuesto del grafo de flujo, valor de la complejidad ciclomática, nombre de las funciones con sus sentencias de control y los respectivos nodos, fecha de evaluación, y nombre del archivo que se evaluó.

Estos requerimientos se describen con más detalle en el documento de especificación de requerimientos (Documento de Visión según RUP) Anexo A, además de otros aspectos.

2.1.4. MODELO DE CASOS DE USO

En el proceso unificado es indispensable definir los casos de uso, los actores, e identificar como están relacionados.

2.1.4.1. Diagrama de casos de uso para la Aplicación

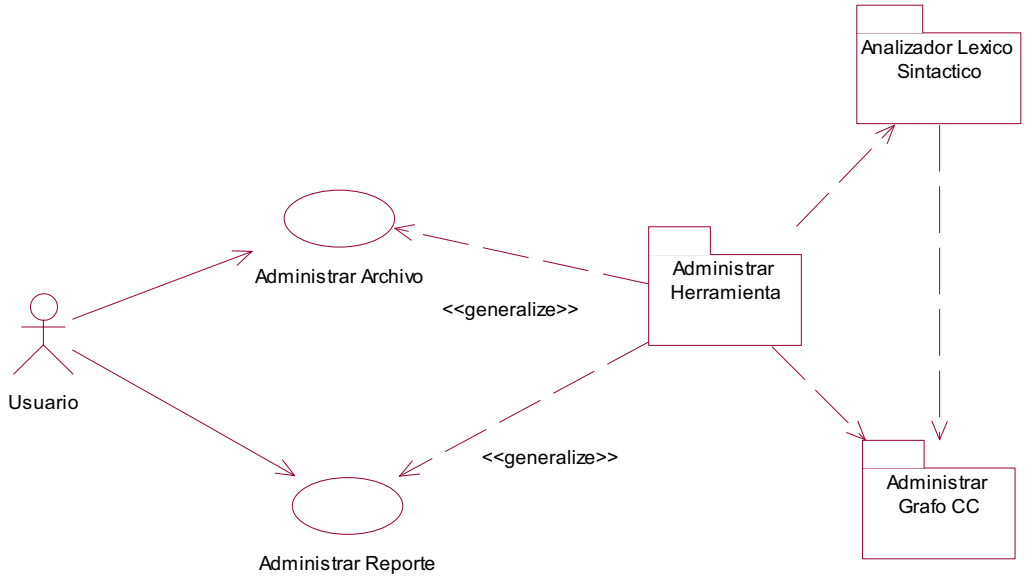


Figura 2.1 Diagrama de casos de uso para la aplicación

Fuente: La autora

Actor	Descripción
Usuario	Persona que manejará la herramienta EVAC

Tabla 2.1 Descripción de actores

Fuente: La autora

Caso de uso	Descripción
Administrar Archivo	- Abrir, editar, guardar un programa en C ANSI para ser evaluado. - Realizar un análisis léxico- sintáctico del archivo (C) antes de ejecutar la evaluación de la calidad del código fuente. - Evaluar la calidad del código fuente generando: - Grafo del flujo. - Valor numérico de la complejidad ciclomática. - Sentencia y secuencia nodos que la conforman.
Administrar Reporte	Guardar reporte.

Tabla 2.2 Casos de Uso de la Aplicación

Fuente: La autora

Paquete de casos de uso	Descripción
Administrar Herramienta	Agrupa los casos de uso relacionados con la Gestión del Archivo, Gestión del Reporte y filtro de archivos.
Analizador Léxico – Sintáctico	Agrupa los casos de uso relacionados con el analizador léxico-sintáctico para el archivo C.
Administrar Grafo CC	Agrupa los casos de uso relacionados con la administración del grafo, su estructuración y el flujo o direccionamiento de aristas.

Tabla 2.3 Descripción de paquetes de casos de uso

Fuente: La autora

2.1.4.2. Especificación de Casos de Uso

En el Anexo B, se describen con más detalle los casos de uso que forman parte de los casos de uso generales de Administrar Archivo y Administrar Reporte.

2.1.5. ARQUITECTURA DEL SOFTWARE A NIVEL DE ANÁLISIS

2.1.5.1. Representación Arquitectónica

La herramienta de evaluación está conformada por tres componentes principales que se muestran en la Figura 2.2.

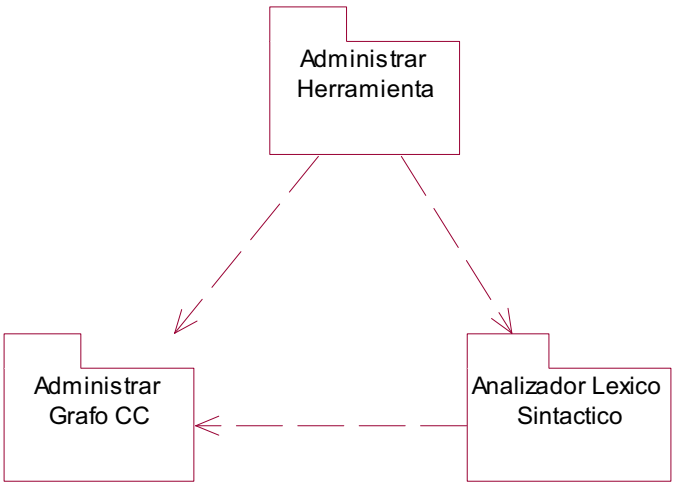


Figura 2.2 Apreciación General de la Arquitectura de componentes de la aplicación EVAC

Fuente: La autora

Vista arquitectónica	Descripción
Vista de requerimientos	Contiene un listado de varios mecanismos de análisis.
Vista lógica	Describe el modelo de diseño a nivel de análisis y contiene la realización de casos de uso.

Tabla 2.4 Vistas arquitectónicas a nivel de análisis

Fuente: La autora

2.1.5.2. Vista de requerimientos: Mecanismos de análisis

En base al Documento de Especificación de Requerimientos (Anexo A), surgen algunos aspectos que forman la arquitectura del software, por lo que se determinan los siguientes mecanismos de análisis:

Nombre	Descripción
Tipo de aplicación	Especifica la modalidad como se va a ejecutar la aplicación o herramienta.
Métrica de complejidad ciclomática	Se refiere a la métrica que se va a utilizar para obtener el nivel de calidad del código fuente.

Tabla 2.5 Mecanismos de análisis

Fuente: La autora

2.1.5.3. Vista lógica: Modelo de diseño

Ver Anexo B.

2.2. DISEÑO

2.2.1. ARQUITECTURA DEL SOFTWARE A NIVEL DE DISEÑO

2.2.1.1. Vista de Requerimientos: Mecanismos de diseño e implementación

En base a los mecanismos de análisis descritos en la
Tabla 2.6 se identifican los siguientes mecanismos de diseño e implementación

Mecanismos de análisis	Mecanismos de diseño	Mecanismos de implementación
Tipo de aplicación	-Arquitectura de sistema orientado a objetos.	-El sistema cuenta con componentes con sus respectivas interfaces.
Métrica de complejidad ciclomática	El sistema carga el archivo o programa C y aplica un algoritmo para obtener la complejidad ciclomática.	La métrica de complejidad ciclomática se calcula en base a : -Número de aristas (E) menos el número de nodos (N) más dos. (E – N + 2)

Tabla 2.6 Mecanismos de diseño e implementación

Fuente: La autora

2.2.1.2. Vista lógica: Modelo de diseño

En base la realización de los casos de uso del Anexo B se clasifican las responsabilidades de la herramienta EVAC en dos grupos que se presentan en la Tabla 2.7 Responsabilidades de EVAC.Tabla 2.7.

Grupo	Responsabilidades
Responsabilidades asociadas al usuario	• Iniciar la herramienta • Finalizar o Salir de la herramienta.
Responsabilidades asociadas a la evaluación de la calidad del código fuente.	• Abrir, editar, guardar los archivos escritos en C. • Antes de la evaluación, verificar que el código fuente sea correcto, a través del uso del analizador léxico-sintáctico. • Evaluación de la calidad del código fuente. • Generación de resultados (Grafo de flujo y valor de la métrica de complejidad ciclomática) y demás detalles que serán colocados en el reporte.

Tabla 2.7 Responsabilidades de EVAC.

Fuente: La autora

2.2.1.2.1. Clases de diseño

Diagrama de clases para la aplicación:

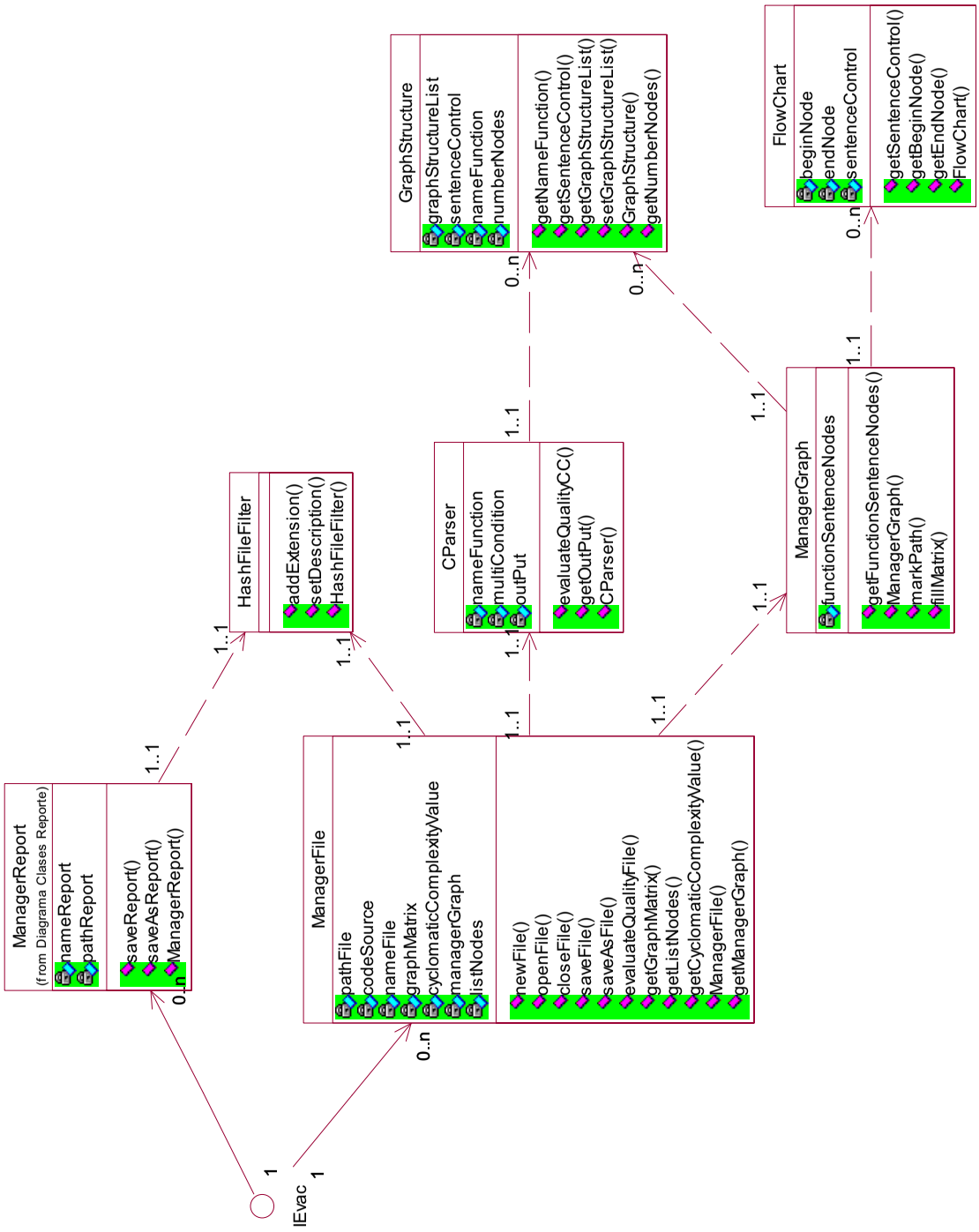


Figura 2.3 Diagrama de Clases

Fuente: La autora

En la

Tabla 2.8 se describen los elementos que conforman el paquete manager- Tool:

Nombre	**Descripción**
IEvac	Interfaz de la Herramienta de Evaluación del código fuente C ANSI basada en los grupos de responsabilidades de la Tabla 2.7
ManagerFile	Ejecuta tareas de gestión de los archivos C.
ManagerReport	Ejecuta tareas de gestión de reportes.
HashFileFilter	Filtra los archivos .C cuando se quiere guardar o abrir un programa en C y filtra los archivos .PDF cuando se quiere guardar los reportes.

Tabla 2.8 Elementos del paquete managerTool.

Fuente: La autora

En la Tabla 2.9 se presenta la descripción de las propiedades de las clases del paquete managerTool.

En la Tabla 2.10 se describen los métodos de las clases del paquete managerTool:

Clase	**Propiedad**	**Descripción**
ManagerFile	nameFile	Nombre del archivo. (String).
	codeSource	Es el texto por el cual está compuesto el archivo.(String).
	pathFile	Dirección donde se encuentra almacenado el archivo. (String).
	graphMatrix	Contiene la matriz “gráfica” necesaria para dibujar la imagen del grafo de flujo. (int [ ][ ]).
	cyclomaticComplexity-Value	Valor de la Complejidad Ciclomática del archivo evaluado. (int).
	managerGraph	Objeto ManagerGraph. (ManagerGraph).
	listNodes	Almacena la sentencia de control y los nodos que la conforman. (ArrayList).
ManagerReport	nameReport	Nombre del reporte. (String).

Clase	Propiedad	Descripción
	pathReport	Dirección donde se encuentra almacenado el reporte. (String).

Tabla 2.9 Propiedades de los elementos del paquete managerTool

Fuente: La autora

Clase	Método	Descripción
ManagerFile	openFile	Abrir un código fuente generado en C ANSI.
	newFile	Abrir una nueva ventana de la herramienta, que representa un archivo nuevo.
	closeFile	Cerrar un archivo abierto.
	saveFile	Guardar el archivo.
	saveAsFile	Seleccionar la ubicación (path) para guardar un archivo.
	evaluateQualityFile	Evaluar la calidad del código a través del uso de la métrica de complejidad ciclomática. Una vez evaluado el archivo se genera el grafo de flujo que nos indica la estructura de las sentencias de control dentro del archivo C, el valor de la complejidad ciclomática, sentencias de control y sus nodos.
	getGraphMatrix	Devuelve la matriz gráfica (int [ ][ ]) obtenida desde ManagerGraph.
	getListNodes	Devuelve el ArrayList de las sentencias de control y sus respectivos nodos.
	getCyclomaticComplexity-Value	Devuelve el valor de la complejidad ciclomática obtenida a través de la siguiente formula: Número de Aristas – Número de Nodos +2
	getManagerGraph	Devuelve el objeto ManagerGraph. Utilizado para retornar un ArrayList con el nombre de la función con sus sentencias de control y nodos declarado en la instancia ManagerGraph.

Clase	Método	Descripción
	ManagerFile	Contructor de la instancia ManagerFile.
ManagerReport	saveReport	Guardar un reporte.
	saveAsReport	Seleccionar la ubicación (path) para almacenar un reporte.
	ManagerReport	Constructor de la instancia ManagerReport.
HashFileFilter	addExtension	Añade el tipo de extensión por el cual se debe filtrar los archivos en el JFileChooser.
	setDescription	Coloca una descripción a desplegar en el JFileChooser.
	HashFileFilter	Constructor de la instancia HashFileFilter

Tabla 2.10 Métodos de los elementos del paquete managerTool.

Fuente: La autora

En la Tabla 2.11 describen los elementos que conforman el paquete graphCyclomaticComplexity:

Nombre	Descripción
ManagerGraph	Contiene las funciones que determinan: - El direccionamiento de las aristas dentro del grafo, es decir desde donde parte una arista y a que nodo llega. - Llenar (la variable int [ ][ ]) la matriz gráfica con números, para que cuando encuentre ceros la imagen se pinte de blanco y donde este un número distinto de cero se pinte el cuadrado de blanco con un Oval encima y con el número del nodo adentro.
FlowChart	Clase que almacena el flujo del grafo, es decir, sus nodos iniciales y finales con sus sentencias de control. Clase que conforma la estructura del ArrayList para almacenar el flujo del grafo.
GraphStructure	Define una estructura de datos a manera de un árbol jerárquico (anidándolas) de las sentencias de control, número de nodos y nombre de la función a las que pertenece dentro del código fuente. Se utiliza un ArrayList recursivo.

Tabla 2.11 Elementos del paquete graphCyclomaticComplexity

Fuente: La autora

En la Tabla 2.12 se presenta la descripción de las propiedades de las clases del paquete graphCyclomaticComplexity.

En la Tabla 2.13 se describen los métodos de las clases del paquete graphCyclomaticComplexity:

Clase	Propiedad	Descripción
ManagerGraph	functionSentence-Nodes	Almacena el nombre de función, sentencias de control con sus respectivos nodos que la conforman. (ArrayList)
FlowChart	beginNode	Nodo inicial desde donde parte una arista.
	endNode	Nodo final hasta la cual llega la arista.
	sentenceControl	Sentencia de control.
GraphStructure	sentenceControl	Almacena la sentencia de control que se está leyendo con el parser.
	nameFunction	Nombre de la función donde se encuentran sentencias de control.
	numberNodes	Número de nodos por cada sentencia de control. Ejemplo: IF: 4 nodos WHILE: 3 nodos
	graphStructureList	ArrayList (recursivo) de la instancia GraphStructure.

Tabla 2.12 Propiedades de los elementos del paquete graphCyclomaticComplexity

Fuente: La autora

Clase	Método	Descripción
ManagerGraph	getFunctionSentence-Nodes	Variable ArrayList que almacena el nombre de la función, sentencias de control y sus nodos.
	markPath	A través del uso de los datos que se encuentran en el ArrayList graphStructureList se define el camino que sigue el grafo, es decir, define la sentencia de control, el nodo inicial y el nodo final, insertando estos datos en el

Clase	Método	Descripción
		ArrayList FlowChart (ArrayList).
	fillMatrix	A través del uso de los datos que se encuentran en el ArrayList graphStructureList se llena la variable tipo matriz, con la cual se dibuja el grafo de flujo del programa.
	getCountNodes	Función que retorna el número total de nodos.
	ManagerGraph	Constructor de cada instancia.
FlowChart	getSentenceControl	Función que obtiene las sentencias de control almacenadas.
	getBeginNode	Obtiene el punto Inicial (nodo inicial).
	getEndNode	Obtiene el punto Final (nodo final).
	FlowChart	Constructor de la instancia FlowChart.
GraphStructure	getNameFunction	Obtener el nombre de cada función.
	getSentenceControl	Obtiene cada sentencia de control.
	getGraphStructureList	Obtiene cada graphStructure almacenado en el ArrayList graphStructureList.
	setGraphStructureList	Almacena un graphStructure dentro del ArrayList graphStructureList, para hacerlo recursivo.
	getNumberNodes	Obtiene el número de nodos, para irlos sumando recursivamente.
	GraphStructure	Constructor de la instancia GraphStructure.

Tabla 2.13 Métodos de los elementos del paquete graphCyclomaticComplexity

Fuente: La autora

En la Tabla 2.14 describen los elementos que conforman el paquete CParser:
Estas clases fueron creadas compilando el archivo C.jj (Reutilización de código) en el generador de Parsers JavaCC.

Nombre	Descripción
CParser	Realiza las funciones del parser, es decir chequea que las declaraciones de funciones, sentencias de control, operadores lógicos, operadores matemáticos, etc., estén correctamente estructurados. Esta es la instancia principal, donde se agrega el nombre de las funciones, sentencias de control, número de nodos y un ArrayList graphStructure.
CParserConstants	Asignación de constantes a las diferentes palabras claves del lenguaje C.
CParserTokenManager	Realiza el manejo de chequeo de los Tokens.
ParseException	Presenta mensajes cuando se existe un error en la estructura sintáctica del analizador.
SimpleCharStream	Asume que el flujo está compuesto solo por caracteres ASCII
Token	Divide la cadena del código fuente en Tokens.
TokenMgrError	Presenta mensajes cuando se presenta un error en la estructura léxica (o de tokens) del analizador.

Tabla 2.14 Elementos del paquete CParser

Fuente: La autora

2.3. IMPLEMENTACIÓN Y PRUEBAS

El modelo de implementación se genera mediante elementos del modelo de diseño, indicando la distribución de los componentes relacionados con la estructura y lenguajes de programación.

Los componentes deben ser probados e integrados en ejecutables para posteriormente efectuar las pruebas de aplicación.

2.3.1. MODELO DE IMPLEMENTACIÓN

Para determinar el modelo de implementación para la herramienta, es indispensable definir el modelo de componentes y las herramientas de desarrollo de la interfaz de usuario.

La aplicación será desarrollada con las siguientes herramientas de desarrollo:

2.3.1.1. Plataforma de programación

JAVA: una tecnología desarrollada por Sun Microsystems para aplicaciones de software independientemente de la plataforma, que engloba:

- **Lenguaje de programación Java**, un lenguaje de programación de alto nivel, orientado a objetos
- **API Java** provista por los creadores del lenguaje Java, y que da a los programadores un ambiente de desarrollo completo así como una infraestructura.
- **Máquina Virtual de Java (JVM)**, la máquina virtual que ejecuta bytecode de Java. También se refiere a la parte de la plataforma Java que se ejecuta en el PC, el entorno en tiempo de ejecución de Java (JRE, Java Runtime Environment).

2.3.1.2. Entorno de desarrollo integrado (IDE)

NetBeans: Es un IDE open-source de Sun Microsystems escrito enteramente en Java. Sun ONE Studio llamado inicialmente Forte for Java se basó en este proyecto y es la versión comercial.

2.3.2. DESCRIPCIÓN DE HERRAMIENTAS Y PLATAFORMA DE DESARROLLO

EVAC está desarrollado completamente sobre Java ya que permite una fácil integración de diferentes tecnologías afines. En la Tabla 2.15 se presenta la descripción del software utilizado.

Software	Descripción	Versión	Sitio Web
NetBeans	Herramienta de desarrollo	5.5	www.netbeans.org
JAVA	Plataforma de programación	1.6	java.sun.com

Tabla 2.15 Descripción de herramientas y plataforma de desarrollo

Fuente: La autora

2.3.2.1. NetBeans

NetBeans es un IDE (Integrated Development Environment) de código fuente abierto, para desarrolladores de software. Este IDE se ejecuta sobre varias plataformas incluyendo Windows, Linux, Solaris, y MacOS. Es de fácil uso e instalación. El NetBeans IDE provee todas las herramientas que necesitan los desarrolladores para crear aplicaciones de escritorio, aplicaciones profesionales, empresariales, móviles y de Web.

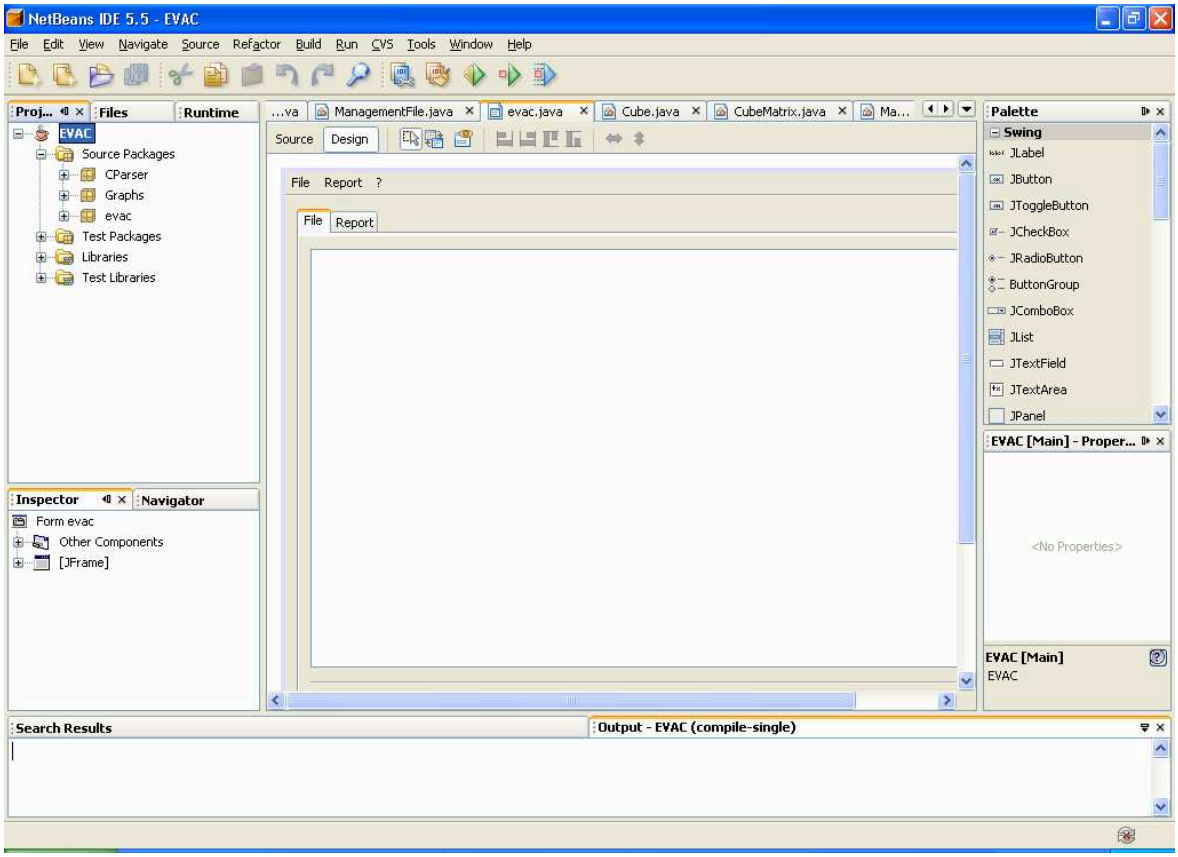


Figura 2.4 NetBeans IDE

Fuente: La autora

2.3.2.2. JAVA

La plataforma Java es el nombre de un entorno o plataforma de computación originaria de Sun Microsystems, capaz de ejecutar aplicaciones desarrolladas usando el Lenguaje de programación Java y un conjunto de herramientas de desarrollo. En este caso, la plataforma no es un hardware específico o un sistema operativo, sino más bien una máquina virtual encargada de la ejecución, y un conjunto de librerías estándar que ofrecen funcionalidad común.

2.3.3. ESTRUCTURA DE PAQUETES JAVA

En esta estructura se define tres paquetes que contiene todas las clases relacionadas con la aplicación.

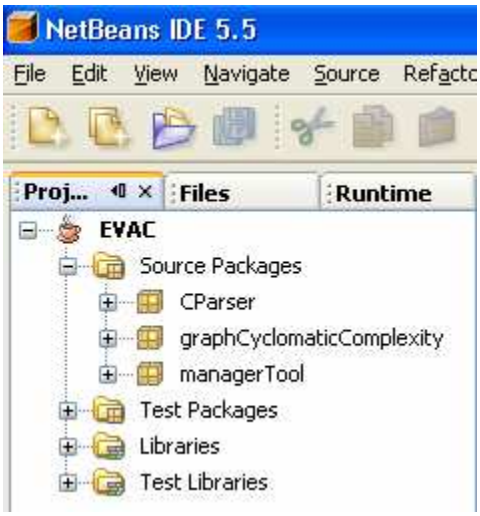


Figura 2.5 Estructura de paquetes Java.

Fuente: La autora

Paquete Java	Paquete de diseño
managerTool	managerTool
graphCyclomaticComplexity	graphCyclomaticComplexity
CParser	CParser

Tabla 2.16 Descripción de paquetes Java

Fuente: La autora

Paquete	Archivo	Clase de diseño / Descripción
managerTool	IEvac	IEvac
	ManagerFile.java	ManagerFile
	ManagerReport.java	ManagerReport
	HashFileFilter.java	HashFileFilter
CParser	CParser.java	Clases generadas automáticamente por JavaCC en base al archivo C.jj
	CParserConstants.java	
	CParserTokenManager.java	
	CParserException.java	
	SimpleCharStream.java	
	Token.java	
	TokenMgrError.java	
graphCyclomaticComplexity	ManagerGraph.java	ManagerGraph
	GraphStructure.java	GraphStructure
	FlowChart.java	FlowChart

Tabla 2.17 Estructura de paquetes Java

Fuente: La autora

2.3.4. PRUEBAS

2.3.4.1. Pruebas de Unidad

En el software orientado a objetos la menor unidad a considerar para realizar una prueba es la clase. Esta prueba está fundamentalmente dirigida a las operaciones encapsuladas por la clase, así como al estado y comportamiento del objeto que se implementa en ella. El énfasis de la prueba de unidad es verificar que esta pequeña unidad trabaje correctamente en forma aislada, antes de proceder a integrarla en el sistema.

2.3.4.2. Pruebas de Integración

Cuando se aplican pruebas de integración al software orientado a objetos, se pretende demostrar que las unidades que ya han sido sometidas a un proceso de prueba y funcionan correctamente, lo hacen de igual forma cuando interactúan y se integran con otras unidades del sistema. Prácticamente, el trabajo de esta prueba se concentra en la interacción de métodos en diferentes unidades.

2.3.4.3. Pruebas en base a los Casos de Uso

Por cada caso de uso que se encuentra descrito en el Anexo B se ejecuta un caso de prueba para verificar que el código realice la funcionalidad especificada.

2.3.4.4. Pruebas del Sistema

La prueba del sistema, realmente, está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas.

En el Anexo C se encuentran la realización del sistema y en el Anexo J (CD) se encuentran las pruebas de unidad, integración, de casos de uso.

CAPÍTULO 3. EVALUACIÓN DE CALIDAD PARA CASOS DE ESTUDIO

En este capítulo se aplicará los procesos de evaluación definidos por la norma ISO/IEC 14598 (Ver Capítulo 1) para evaluar la calidad del código fuente de la herramienta (EVAC) desarrollada en este proyecto de titulación.

3.1. DESCRIPCIÓN DE LOS CASOS DE ESTUDIO

Nombre de la Empresa: Escuela Politécnica Nacional

Misión de la Empresa: Promover el desarrollo de herramientas de software que permitan evaluar la calidad del código fuente y que estas puedan ser utilizadas en el área de la Calidad de Software.

Nombre del Producto: Herramienta para la evaluación de la calidad del código fuente generado en C ANSI (EVAC).

Misión del producto: Analizar la calidad del código fuente desarrollado en C ANSI en base a la utilización de la métrica de complejidad ciclomática.

Metodología de Desarrollo: Orientado a Objetos

Sistema Operativo: Windows XP Professional

Lenguaje de programación: JAVA

Hardware Utilizado:

- Procesador Pentium IV 2.40 GHz
- Memoria RAM de 512 MB
- Espacio Requerido 3 MB

- Monitor Super VGA

Módulos de la Herramienta:

- Archivo
- Reporte

Módulo de Archivo:

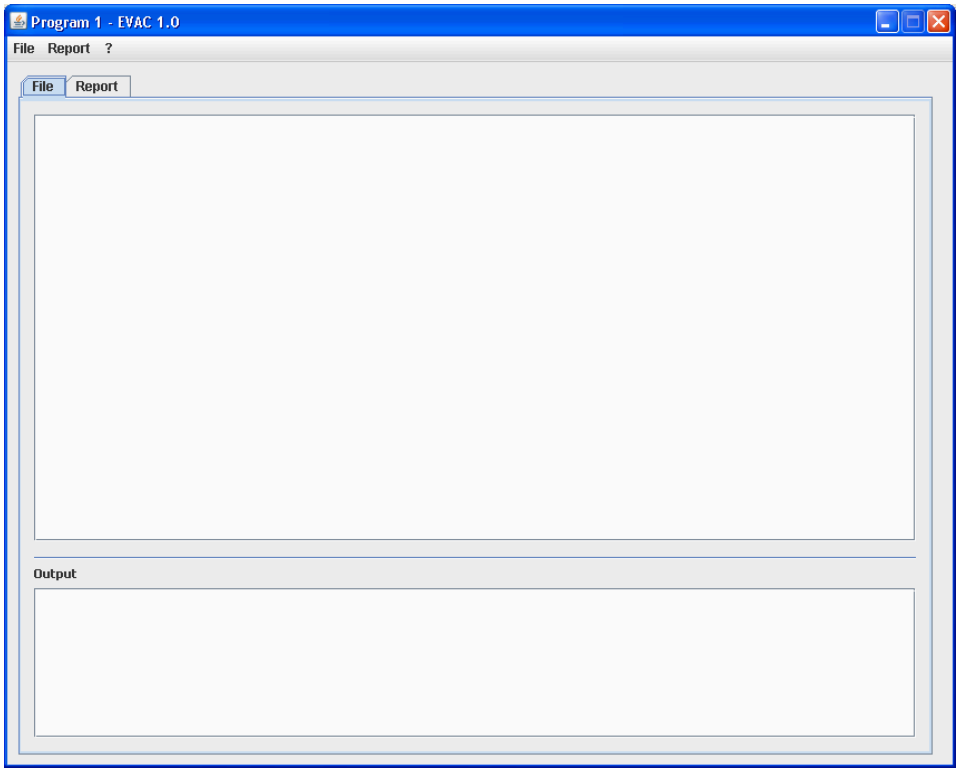


Figura 3.1 Módulo Archivo

Fuente: La autora

Módulo de Reporte:

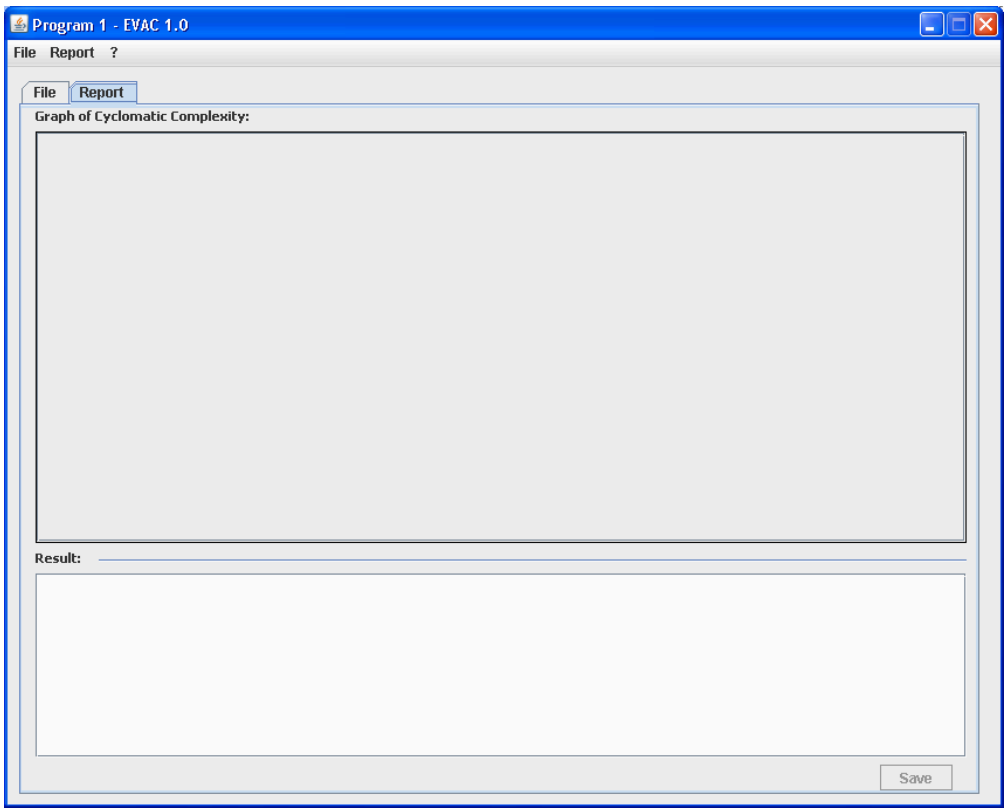


Figura 3.2 Módulo Reporte

Fuente: La autora

3.2. PROCESO DE EVALUACIÓN

3.2.1. ESTABLECIMIENTO DE LOS REQUISITOS

3.2.1.1. Establecer el propósito de la Evaluación

El propósito de la evaluación es determinar la calidad del código fuente de la herramienta EVAC usando los procesos de evaluación del software que se definen el la norma ISO/IEC 14598.

En la Tabla 3.1 y Tabla 3.2 se muestran las funciones que maneja el módulo de Archivo y Reporte respectivamente.

MODULO ARCHIVO EVAC	
FUNCIONES	**DESCRIPCION**
Administrar Archivo	• Abre una nueva aplicación de EVAC. • Busca un archivo C ANSI y lo abre. • Cierra el archivo C ANSI abierto. • Guarda el archivo C en la misma dirección (path) desde el cual fue abierto o guardado anteriormente. • Busca un path para almacenar el archivo C ANSI. • Evalúa la calidad de los archivos C ANSI abiertos, genera un grafo de flujo e información de las sentencias de control y el valor de la complejidad ciclomática.

Tabla 3.1 Módulo Archivo EVAC

Fuente: EVAC

MODULO REPORTE EVAC	
FUNCIONES	**DESCRIPCION**
Administrar Reporte	• Guarda la información obtenida de la evaluación de un archivos C ANSI en un archivo .PDF en una dirección en la cual se almacenó previamente. • Busca un path para almacenar el reporte .PDF de la evaluación de los archivos C ANSI.

Tabla 3.2 Módulo Reporte EVAC

Fuente: EVAC

Para ejecutar la evaluación es necesario definir los niveles de riesgos. Según el Anexo H, se determina que la evaluación es de nivel D en los aspectos de salvamento, económico, seguridad y medioambiental.

3.2.1.2. Identificar el tipo de producto

En base al ciclo de vida del Software de la norma ISO/IEC 14598-1 se establece que la aplicación a evaluar es un producto final.

3.2.1.3. Especificar el modelo de calidad

El modelo de calidad que se utilizará para realizar la evaluación es el Modelo de Calidad preescrito en la norma ISO/IEC 9126-1. Las características que componen el modelo son: Funcionalidad, Fiabilidad, Usabilidad, Eficiencia, Mantenibilidad, Portabilidad.

3.2.2. ESPECIFICACIÓN DE LA EVALUACIÓN

3.2.2.1. Selección de Métricas

Siendo el propósito de la evaluación evaluar la calidad del código fuente, las métricas seleccionadas son las métricas internas definidas en el modelo de calidad ISO/IEC 9126-3.

Las métricas escogidas para los casos de estudio son:

CARACTERISTICA	SUBCARACTERISTICA	METRICA
FUNCIONALIDAD	Cumplimiento de la Funcionalidad	Cumplimiento de la Funcionalidad
FIABILIDAD	Tolerancia a Fallas	Anulación de Operación Incorrecta
USABILIDAD	Capacidad para ser entendido	Funciones Evidentes
	Operabilidad	Claridad del mensaje
		Recuperabilidad de error operacional
PORTABILIDAD	Adaptabilidad	Adaptabilidad de la estructura de datos

Tabla 3.3 Métricas para Código Fuente

Fuente: La autora

Las métricas referidas a la característica de Mantenibilidad no son seleccionadas porque se trata de un producto de software que posee una sola versión, es decir, no ha sido modificado, y las métricas referentes a Eficiencia tampoco se las toma en cuenta porque no se obtiene resultado alguno, pero se analizó el rendimiento de la herramienta, la cual se detalla en las pruebas del sistema en el Anexo C. A continuación, en la Tabla 3.4 se presenta la especificación formalizada de las métricas que va a ser utilizadas:

Característica: Funcionalidad **Subcaracterística:** Cumplimiento de la Funcionalidad

Métrica interna de Cumplimiento de la Funcionalidad									
Nombre de la métrica	**Propósito de la métrica**	**Método de aplicación**	**Medición, formula y cálculo de datos**	**Interpretación de los valores medidos**	**Tipo de escala de métrica**	**Tipo de medida**	**Entradas para medición**	**Referente ISO/IEC 12207 SLCP**	**Usuarios seleccionados**
Cumplimiento de la Funcionalidad	¿Cuán dócil es la funcionalidad del producto al aplicar regulaciones, estándares y convenciones?	Contar el número de detalles que se han reunido y que requieren cumplimiento y comparar con el número de detalles que requieren cumplimiento como en la especificación	X = A / B A = número de ítems implementados correctamente relacionados con el cumplimiento de funcionalidad confirmado en la evaluación B = número total de ítems de cumplimiento.	0<= X <=1 El más cercano a 1. Es el mejor	Absoluto	X = contable/ contable A = contable B = contable	Especificación de cumplimiento y relación de estándares, convenciones o regulaciones. Diseño Código Fuente Reporte de Revisión.	Verificación Revisión colectiva	Analistas Desarrolladores

Característica: Fiabilidad **Subcaracterística:** Tolerancia a fallas

Métrica interna de Tolerancia a fallas									
Nombre de la métrica	**Propósito de la métrica**	**Método de aplicación**	**Medición, formula y cálculo de datos**	**Interpretación de los valores medidos**	**Tipo de escala de métrica**	**Tipo de medida**	**Entradas para medición**	**Referente ISO/IEC 12207 SLCP**	**Usuarios seleccionados**
Anulación de operaciones incorrecta	¿Cuántas funciones son implementadas con capacidad de anular operaciones incorrectas?	Contar el número de funciones implementadas que evitan críticas y serias fallas causadas por operaciones incorrectas y comparar éste al número de modelo de operaciones incorrectas a ser consideradas.	X = A / B A = número de funciones implementadas para anular operaciones incorrectas B = número de operaciones incorrectas del modelo a ser consideradas.	0<= X Donde X es mayor a 0, siendo X la mejor anulación de operaciones incorrectas	Absoluto	X = contable/ contable A = contable B = contable	El valor A viene del reporte de revisión. El valor B viene del documento de especificación de requerimientos	Verificación Validación. Revisión colectiva Resolución del problema	Desarrolladores Analistas Soporte

Característica: Usabilidad **Subcaracterística:** Capacidad para ser entendido

Métrica interna de Capacidad para ser entendido									
Nombre de la métrica	**Propósito de la métrica**	**Método de aplicación**	**Medición, formula y cálculo de datos**	**Interpretación de los valores medidos**	**Tipo de escala de métrica**	**Tipo de medida**	**Entradas para medición**	**Referente ISO/IEC 12207 SLCP**	**Usuarios seleccionados**
Funciones Evidentes	¿Qué proporción de las funciones del producto son evidentes al usuario?	Contar el número de funciones que son evidentes al usuario y comprar con el número total de funciones.	X = A / B A = número de funciones (o tipo de funciones) evidentes al usuario B = número total de funciones (o tipo de funciones).	0<= X <= 1 El límite a 1 es el mejor.	Absoluto	X = contable/ contable A = contable B = contable	Especificación de requerimientos Diseño Reporte de revisión	Verificación Revisión colectiva	Desarrolladores Analistas

Característica: Usabilidad **Subcaracterística:** Operabilidad

Métrica interna de Operabilidad									
Nombre de la métrica	**Propósito de la métrica**	**Método de aplicación**	**Medición, formula y cálculo de datos**	**Interpretación de los valores medidos**	**Tipo de escala de métrica**	**Tipo de medida**	**Entradas para medición**	**Referente ISO/IEC 12207 SLCP**	**Usuarios seleccionados**
Claridad del mensaje	¿Qué proporción del mensaje es auto explicativo?	Contar el número de mensajes implementados con explicaciones claras y comparar con el número total de mensajes implementados.	X=A/B A=Número de mensajes llevados a cabo con explicaciones claras. B= Número de mensajes llevados a cabo	0 <= X <= 1 El más cercano a 1, el más claro.	Absoluto	X=contable / contable A= contable B= contable	La especificación de Requisitos Diseño Informe de revisión	Comprobación Revisión colectiva	Diseñadores Analistas

Métrica interna de Operabilidad									
Nombre de la métrica	**Propósito de la métrica**	**Método de aplicación**	**Medición, formula y cálculo de datos**	**Interpretación de los valores medidos**	**Tipo de escala de métrica**	**Tipo de medida**	**Entradas para medición**	**Referente ISO/IEC 12207 SLCP**	**Usuarios seleccionados**
Recuperabilidad de error operacional	¿Qué proporción de funciones pueden tolerar errores de usuario?	Contar el número de funciones implementadas que toleran errores de usuarios y comparar con el número total de funciones requeridas que tiene capacidad de tolerancia.	X=A/B A=Número de funciones implementadas con tolerancia de error de usuarios. B=Número total de funciones requeridas con capacidad de tolerancia.	0 <= X <= 1 El más cercano a 1, el más recuperable.	Absoluto	X=contable / contable A= contable B= contable	La especificación de Requisitos Diseño Informe de revisión	Comprobación Revisión colectiva	Diseñadores Analistas

Característica: Portabilidad **Subcaracterística:** Adaptabilidad

Métrica interna de Adaptabilidad									
Nombre de la métrica	**Propósito de la métrica**	**Método de aplicación**	**Medición, formula y cálculo de datos**	**Interpretación de los valores medidos**	**Tipo de escala de métrica**	**Tipo de medida**	**Entradas para medición**	**Referente ISO/IEC 12207 SLCP**	**Usuarios seleccionados**
Adaptabilidad de la estructura de datos	¿Cuán adaptable es el producto a los cambios de estructura de datos?	Contar el número de estructuras de datos que son operables y no tienen ninguna limitación después de la adaptación y comparar con el número total de estructuras de datos que requieren capacidad de adaptación.	X=A/B A=Número de estructuras de datos que son operables y no tienen ninguna limitación después de la adaptación, conformada la revisión B=Número total de estructuras de datos que requieren capacidad de adaptación	0 <= X <= 1 El más cercano a 1. Es el mejor	Absoluto	X=contable / contable A= contable B= contable	La especificación de Requisitos Diseño Informe de revisión	Comprobación Revisión colectiva	Diseñadores Analistas

Tabla 3.4 Especificación formalizada de métricas

Fuente: ISO / IEC 9126-3

Al final de la evaluación se tendrá la documentación del ***módulo de evaluación*** aplicado al caso de estudio, de acuerdo a la definición del estándar ISO/IEC 14598-6, (Ver Anexo G).

3.2.2.2. Establecer los niveles de puntuación

Siendo la norma ISO/IEC 14598 un estándar que es aplicable a cualquier tipo de producto de software sin importar la tecnología, lenguaje de programación, metodología de desarrollo y arquitectura, propone dos alternativas cualitativas: dividir la escala en dos categorías: satisfactoria e insatisfactoria, o dividir la escala en cuatro categorías: excede los requerimientos, rango objetivo, mínimamente aceptable e inaceptable. Cabe indicar que la norma no establece rangos cuantitativos por el mismo hecho de ser un proceso de evaluación general de productos de software.

Entonces la escala de los valores a ser medidos se fundamentan en el Modelo MOSCA (Modelo Sistémico para estimar la Calidad de los Sistemas de Software) desarrollado en la Universidad Simón Bolívar por LISI[4] (Laboratorio de investigación en Sistemas de información) de Caracas Venezuela, que utiliza el modelo de calidad de la Norma ISO/IEC 9126 y Dromey, y para la valoración de los procesos de software utiliza la Norma ISO/IEC 15504 conocida como SPICE, en el cual se determinan que los valores aceptables deben ser mayores o iguales al 0.75 (75%).
La escala definida se muestra en la Figura 3.3 :

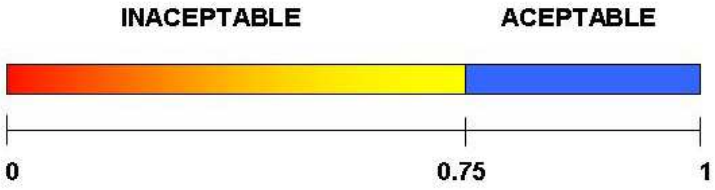


Figura 3.3 Escala de valores para Proceso de Evaluación

Fuente: Modelo MOSCA

[4] http://www.lisi.usb.ve

3.2.2.3. Establecer criterios para la valoración

Para este propósito se va a utilizar el modelo de calidad de la norma ISO/IEC 9126-1 y las definiciones de las métricas de calidad interna de la ISO/IEC 9126-3. Adicionalmente se utilizará la información de la Tabla 3.4 que hacen referencia a medición, formula, cálculo de datos e interpretación de los valores medidos.

3.2.3. DISEÑO DE LA EVALUACIÓN

3.2.3.1. Plan de la evaluación

3.2.3.1.1. Introducción

El propósito de realizar el Plan es el de definir cada una de las actividades a ejecutarse en la evaluación de la herramienta EVAC, especificar los objetivos para los cuales se realiza la evaluación, identificar las características de calidad a ser utilizadas, determinar la lista de prioridades en base a los componentes del producto de software, se definirá los recursos humanos, técnicos y administrativos a utilizarse para la evaluación de la calidad de la aplicación, realizar un cronograma de actividades (carta Gantt) en el cual se definirá el tiempo empleado para cumplir cada una de las actividades para ejecutar la evaluación, definir la técnica, procedimiento y herramientas a utilizar para llevar a cabo la evaluación, finalmente especificar las fórmulas para la sumarización de subcaracterísticas y características según la norma ISO/IEC 14598-6 para su posterior análisis.

3.2.3.1.2. Objetivos

Los objetivos del plan de evaluación son los siguientes:

- Asegurar que la evaluación sea llevada efectivamente.
- Documentar los métodos y procedimientos de evaluación.
- Determinar los recursos necesarios para realizar la evaluación.
- Programar las actividades conforme a los recursos disponibles.
- Asegurar que los resultados de la evaluación puedan ser cuantificados, claramente presentados e identificados.
- Asegurar la disponibilidad que las recomendaciones para futuras actividades de evaluación estén disponibles.

3.2.3.1.3. Características de calidad aplicables

Siendo el caso de estudio: “Evaluación de la Calidad del Código Fuente” basados en la norma ISO / IEC 9126-3 las características a utilizar son las definidas en la Tabla 3.3.

3.2.3.1.4. Lista de prioridades

En la Tabla 3.5 se especifica la prioridad de los componentes del producto de Software con su correspondiente nivel de importancia para realizar la evaluación como.

ITEM	PRIORIDAD
Disponibilidad de Código Fuente	Alta
Disponibilidad de documentación de desarrollo	Alta
Estándares (Diseño – Código Fuente)	Alta
Disponibilidad de Ejecutables	Media
Disponibilidad de Manuales del Sistema	Media
Ayudas del Sistema	Baja

Tabla 3.5 Lista de prioridades

Fuente: La autora

3.2.3.1.5. Objetivos de la calidad

Los Objetivos de Calidad son los siguientes:

- Contar el número de detalles que se han reunido y que requieren cumplimiento de Funcionalidad.
- Contar el número de funciones implementadas que evitan críticas y serias fallas causadas por operaciones incorrectas.
- Contar el número de funciones que son evidentes al usuario.
- Contar el número de mensajes implementados con explicaciones claras.

- Contar el número de funciones implementadas que toleran errores de usuarios.
- Contar el número de estructuras de datos que son operables y no tienen ninguna limitación.

3.2.3.1.6. Recursos

Para realizar el plan de la evaluación se utiliza tres tipos de recursos: Humano, Técnico y Administrativo, detallados en el Anexo D. A continuación en la Tabla 3.6 se muestra el resumen de los costos del plan en la que se indica los valores de todos los recursos utilizados y el costo final de la evaluación de la aplicación.

COSTO TOTAL DEL PROYECTO

RECURSO	**VALOR**
HUMANO	**800**
TECNICO	**31.6**
ADMINISTRATVO	**264.75**
TOTAL	**1096.35**
3 % IMPREVISTOS	**32.89**
15 % UTILIDAD	**164.45**
COSTO TOTAL	**1293.69 USD**

Tabla 3.6 Costo Total del Proyecto

Fuente: La autora

3.2.3.1.7. Cronograma

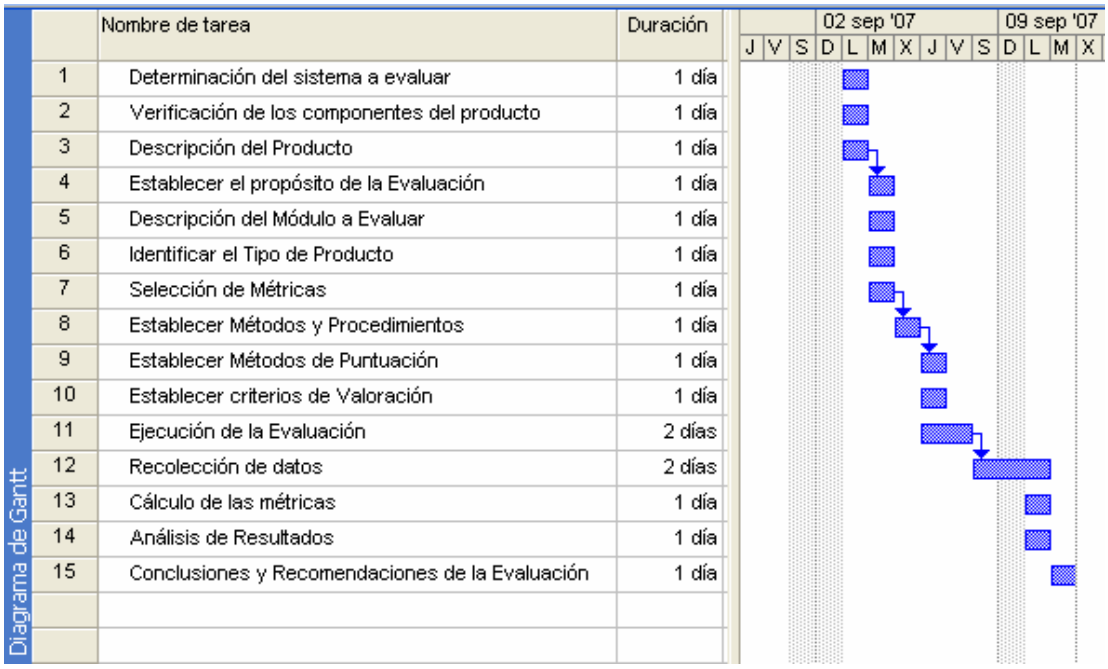


Figura 3.4 Cronograma de Actividades del Proceso de Evaluación

Fuente: La Autora

3.2.3.1.8. Definición de responsabilidades

La definición de responsabilidades para cada una de las tareas a ser realizadas se muestra en la Tabla 3.7, en la que se tiene el rol de Evaluador (Amy Calahorrano N.).

TAREA	RESPONSABLE
Documentación	Evaluador
Determinación del Sistema a Evaluar	Evaluador
Verificación de los componentes del producto	Evaluador
Descripción del Producto	Evaluador
Establecer el propósito de la Evaluación	Evaluador
Descripción del Módulo a Evaluar	Evaluador
Identificar el Tipo de Producto	Evaluador

TAREA	RESPONSABLE
Selección de Métricas	Evaluador
Establecer Métodos y Procedimientos	Evaluador
Establecer Métodos de Puntuación	Evaluador
Establecer criterios de Valoración	Evaluador
Ejecución de la Evaluación	Evaluador
Recolección de Datos	Evaluador
Cálculo de Métricas	Evaluador
Análisis de Resultados	Evaluador
Conclusiones y Recomendaciones de la Evaluación	Evaluador

Tabla 3.7 Definición de Responsabilidades para el Proceso de Evaluación

Fuente: La autora

3.2.3.1.9. Técnica, Procedimiento y Herramientas de Medición

Para realizar las mediciones de las métricas establecidas en la Tabla 3.3 se utilizará la técnica de Inspección y como procedimiento la revisión de código. Las herramientas utilizadas para realizar las medidas son propias de NetBeans 5.5 y son las siguientes:

- Examinador de Objetos
- Vista de Clases
- Ir a referencia
- Esquematización (Colapsar rutina o clase)
- Comando Buscar

3.2.3.1.10. Uso y Análisis de Datos

Los datos medidos serán calculados de acuerdo a las formulas de las métricas establecidas en la Tabla 3.4, luego serán trasladados a una herramienta de representación gráfica (Microsoft Excel 2003) para su interpretación y análisis.

Las fórmulas a utilizarse para la sumarización de subcaracterísticas y características según la norma ISO/IEC 14598-6 son las siguientes:

$$Vsc = \frac{\sum m}{n}$$; donde: Vsc=Valor de subcaracterística, m= Valor de Métrica y n= número de métricas.

$$Vc = \frac{\sum Vsc}{nsc}$$; donde: Vc= Valor de característica, Vsc=Valor de subcaracterística, nsc= número de subcaraterísticas.

3.2.4. EJECUCIÓN DE LA EVALUACIÓN

3.2.4.1. Tomar Medidas

Conforme al Plan de Evaluación se tomará medidas para cada una de las funciones que componen los módulos de la herramienta EVAC descritas en la Tabla 3.1 y Tabla 3.2.

3.2.4.1.1. Medidas definidas en base al código fuente

A continuación se muestra un ejemplo de las medidas realizadas de la Característica FIABILIDAD de la Función Administrar Archivo utilizando el Código fuente de la aplicación.

FUNCION: ADMINISTRAR ARCHIVO

CARACTERÍSTICA: FIABILIDAD

Subcaracterística: tolerancia a fallas

Métrica: anulación de operaciones incorrectas

Siendo A: funciones implementadas, B: funciones especificadas en los requisitos y X: A/B se lo determina en la Tabla 3.4.

El código fuente (funciones implementadas) que se utilizó para realizar el cálculo de las diferentes métricas seleccionadas, se presenta a continuación:

NUEVO

```
public void newFile(final int x,final int y)
```

ABRIR

```
public String openFile()
//Si no existe el archivo .C que se desea abrir
if (extFile.equals(".C")||extFile.equals(".c")) {
if (!selected.exists()) {
                JOptionPane.showMessageDialog(frameEvac,
                    "File " +nameCurrentFile+" does not exist",
                    "Information",
                    JOptionPane.ERROR_MESSAGE); }
else {
//Si desea abrir un archivo que no sea  .C
            JOptionPane.showMessageDialog(frameEvac,
                "Only allows to open files written in C ",
                "Information",
                JOptionPane.ERROR_MESSAGE);
        }
```

CERRAR

```
public int exitFile(String codeSourceB, String codeSourceE, String pathCloseF, String nameCloseF)
```

GUARDAR

```
public boolean saveFile(String nameSaveF, String pathSaveF, String codeSourceES)
try {…
    }
catch (IOException e) {
    JOptionPane.showMessageDialog(frameEvac,"It did not save the changes of the file "+nameSaveF,
                "Error",JOptionPane.ERROR_MESSAGE);}
```

GUARDAR COMO

```
public boolean saveAsFile(String pathSaveAsF, String codeSourceSave)
try {…
    }
catch (SecurityException e) {
```

JOptionPane.showMessageDialog(frameEvac,"Error of access to the directory "+currentLocation,
"Error",JOptionPane.ERROR_MESSAGE);}

EVALUAR CALIDAD

Parser

public int evaluateQualityCC(String path, GraphStructure graphStructure)

this.codErrParser= parser.evaluateQualityCC(path,graphStructure);

Registra Flujo

public int markPath(GraphStructure graphStructure,ArrayList listNode, int node) **//No Cumple//**

managerGraph.markPath(graphStructure,listNodes,1);

Registra Diagrama

public int[] fillMatrix(GraphStructure graphStructure,int[][] matrix, int node, int x, int y2, int m, boolean incremental, int space) **//No Cumple//**

managerGraph.fillMatrix(graphStructure,matrix,0,0,0,lenthMatrix*3,true,5);

	NUEVO	ABRIR	CERRAR	GUARDAR	GUARDAR COMO	EVALUAR CALIDAD	TOTAL
B	1	1	1	1	1	3	8
A	1	1	1	1	1	1	6

A	B	X
6	8	0,75

Por lo extenso del documento la medición de las funciones restantes se encuentran en el Anexo E.

3.2.4.1.2. Resultado de las medidas

FUNCION: ADMINISTRAR ARCHIVO

FUNCIONALIDAD

SUBCARACTERISTICA	METRICA	RESULTADO
Cumplimiento de la Funcionalidad	Cumplimiento de la Funcionalidad	1

FIABILIDAD

SUBCARACTERISTICA	METRICA	RESULTADO
Tolerancia a fallas	Anulación de operación incorrecta	0.75

USABILIDAD

SUBCARACTERISTICA	METRICA	RESULTADO
Capacidad para ser entendido	Funciones Evidentes	1
Operabilidad	Claridad del mensaje	1
	Recuperabilidad de error operacional	1

PORTABILIDAD

SUBCARACTERISTICA	METRICA	RESULTADO
Adaptabilidad	Adaptabilidad de la estructura de datos	1

FUNCION: ADMINISTRAR REPORTE

FUNCIONALIDAD

SUBCARACTERISTICA	METRICA	RESULTADO
Cumplimiento de la Funcionalidad	Cumplimiento de la Funcionalidad	1

FIABILIDAD

SUBCARACTERISTICA	METRICA	RESULTADO
Tolerancia a fallas	Anulación de operación incorrecta	1

USABILIDAD

SUBCARACTERISTICA	METRICA	RESULTADO
Capacidad para ser entendido	Funciones Evidentes	1
Operabilidad	Claridad del mensaje	1
	Recuperabilidad de error operacional	1

PORTABILIDAD

SUBCARACTERISTICA	METRICA	RESULTADO
Adaptabilidad	Adaptabilidad de la estructura de datos	1

3.2.4.2. Comparar con criterios

El valor de la métrica es comparado con el criterio que se muestra en la Figura 3.3, en el cual los valores aceptables son mayores o iguales a 0.75 (75%). A continuación se muestra los datos obtenidos de la función Administrar Archivo y Administrar Reporte.

FUNCION: ADMINISTRAR ARCHIVO

CARACTERISTICA: FUNCIONALIDAD

SUBCARACTERISTICA: Cumplimiento de la Funcionalidad

METRICA: Cumplimiento de la Funcionalidad

	DISEÑO	CODIGO FUENTE	RELACION	TOTAL
B	6	2	2	10
A	6	2	2	10

A	B	X	CRITERIO
10	10	1,00	ACEPTABLE

CARACTERISTICA: FIABILIDAD

SUBCARACTERISTICA: Tolerancia a fallas

METRICA: Anulación de operación incorrecta

	NUEVO	ABRIR	CERRAR	GUARDAR	GUARDAR COMO	EVALUAR CALIDAD	TOTAL
B	1	1	1	1	1	3	8
A	1	1	1	1	1	1	6

A	B	X	CRITERIO
6	8	0,75	ACEPTABLE

CARACTERISTICA: USABILIDAD

SUBCARACTERISTICA: Capacidad para ser entendido

METRICA: Funciones Evidentes

	NUEVO	ABRIR	CERRAR	GUARDAR	GUARDAR COMO	EVALUAR CALIDAD	TOTAL
B	1	1	1	1	1	1	6
A	1	1	1	1	1	1	6

A	B	X	CRITERIO
6	6	1,00	ACEPTABLE

SUBCARACTERISTICA: Operabilidad

METRICA: Claridad del mensaje

A	B	X	CRITERIO
7	7	1,00	ACEPTABLE

METRICA: Recuperabilidad de error operacional (Excepciones)

A	B	X	CRITERIO
3	3	1,00	ACEPTABLE

CARACTERISTICA: PORTABILIDAD

SUBCARACTERISTICA: Adaptabilidad

METRICA: Adaptabilidad de la estructura de datos

A	B	X	CRITERIO
2	2	1.00	ACEPTABLE

FUNCION: ADMINISTRAR REPORTE

CARACTERISTICA: FUNCIONALIDAD

SUBCARACTERISTICA: Cumplimiento de la Funcionalidad

METRICA: Cumplimiento de la Funcionalidad

	DISEÑO	CODIGO FUENTE	RELACION	TOTAL
B	2	2	2	6
A	2	2	2	6

A	B	X	CRITERIO
6	6	1,00	ACEPTABLE

CARACTERISTICA: FIABILIDAD

SUBCARACTERISTICA: Tolerancia a fallas

METRICA: Anulación de operación incorrecta

	GUARDAR	GUARDAR COMO	TOTAL
B	1	1	2
A	1	1	2

A	B	X	CRITERIO
2	2	1,00	ACEPTABLE

CARACTERISTICA: USABILIDAD

SUBCARACTERISTICA: Capacidad para ser entendido

METRICA: Funciones Evidentes

	GUARDAR	GUARDAR COMO	TOTAL
B	1	1	2
A	1	1	2

A	B	X	CRITERIO
2	2	1,00	ACEPTABLE

SUBCARACTERISTICA: Operabilidad

METRICA: Claridad del mensaje

A	B	X	CRITERIO
3	3	1,00	ACEPTABLE

METRICA: Recuperabilidad de error operacional (Excepciones)

A	B	X	CRITERIO
2	2	1,00	ACEPTABLE

CARACTERISTICA: PORTABILIDAD

SUBCARACTERISTICA: Adaptabilidad

METRICA: Adaptabilidad de la estructura de datos

A	B	X	CRITERIO
2	2	1.00	ACEPTABLE

3.2.4.3. Valorar Resultados

La Tabla 3.8 contiene la sumarización de los datos obtenidos de las métricas aplicadas a la herramienta EVAC.

CARACTERISTICA	SUBCARACTERISTICA	METRICA	m	Vsc	Vc
FUNCIONALIDAD	Cumplimiento de la Funcionalidad	Cumplimiento de la Funcionalidad	1.00	1.00	**1.00**
FIABILIDAD	Tolerancia a Fallas	Anulación de Operaciones Incorrectas	0.87	0.87	**0.87**
USABILIDAD	Capacidad para ser Entendido	Funciones Evidentes	1.00	1.00	**1.00**
	Operabilidad	Claridad del mensaje	1.00	1.00	

CARACTERISTICA	SUBCARACTERISTICA	METRICA	m	Vsc	Vc
		Recuperabilidad de error operacional	1.00		
PORTABILIDAD	Adaptabilidad	Adaptabilidad de la Estructura de Datos	1	1	**1**

Tabla 3.8 Sumarización de los datos obtenidos de las métricas aplicadas a la herramienta EVAC
m=Valor de métrica Vsc= Valor de Subcaracterística Vc= Valor de Característica
Fuente: La autora

3.2.5. CONCLUSIÓN DE LA EVALUACIÓN

3.2.5.1. Análisis de Resultados

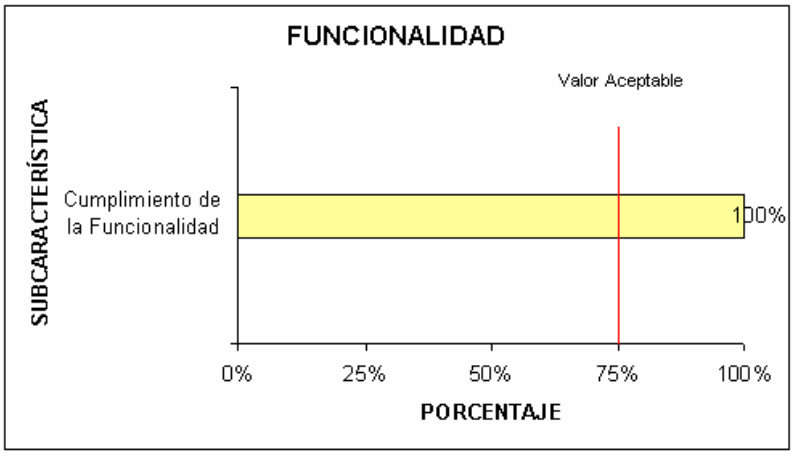


Figura 3.5 Análisis de Resultados, característica Funcionalidad
Fuente: La autora

La subcaracterística de **Funcionalidad** del código fuente de la aplicación EVAC, sobrepasan el valor aceptable de la calidad, lo cual indica que el software cumple con todas funcionalidades especificadas en su documento de análisis y diseño.

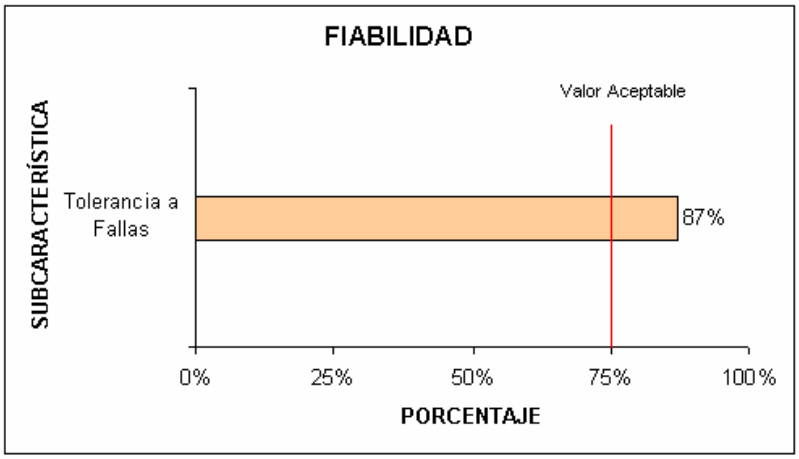


Figura 3.6 Análisis de Resultados, característica Fiabilidad

Fuente: La autora

La subcaracterística de Tolerancia a fallas de la **Fiabilidad** en el gráfico sobrepasa el nivel de aceptación de calidad, la misma que permite mantener un nivel adecuado de rendimiento en caso de un error operacional.

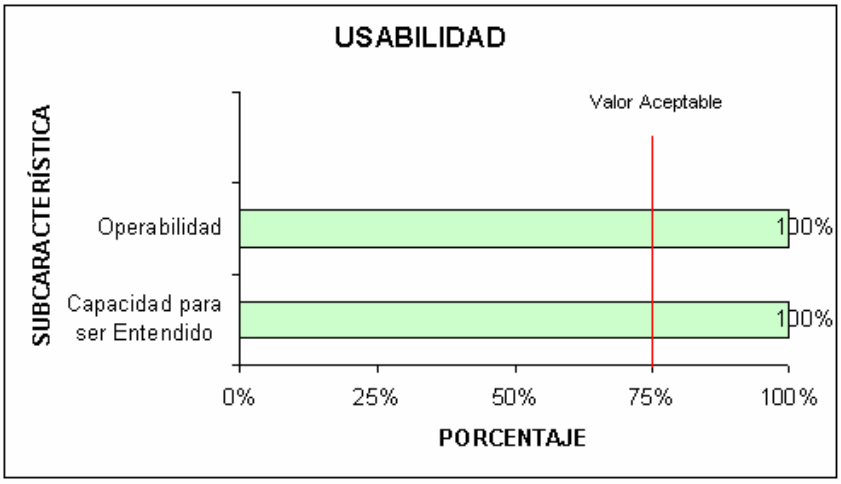


Figura 3.7 Análisis de Resultados, característica Usabilidad

Fuente: La autora

Las subcaracterísticas de **Usabilidad** sobrepasan el nivel aceptable de calidad, lo que quiere decir que los usuarios pueden operar, controlar y entender el producto de software con facilidad.

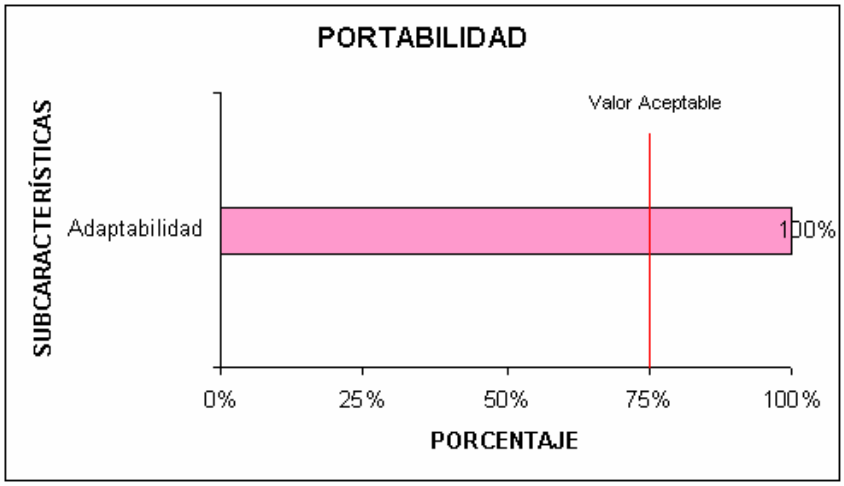


Figura 3.8 Análisis de Resultados, característica Portabilidad

Fuente: La autora

Como se puede apreciar la subcaracterística de **Portabilidad** referida a la Adaptabilidad sobrepasa el valor aceptable de calidad, es decir, el producto de software es flexible al momento de adaptarse a ambientes diferentes (Back-End, Front-End) por parte de la organización.

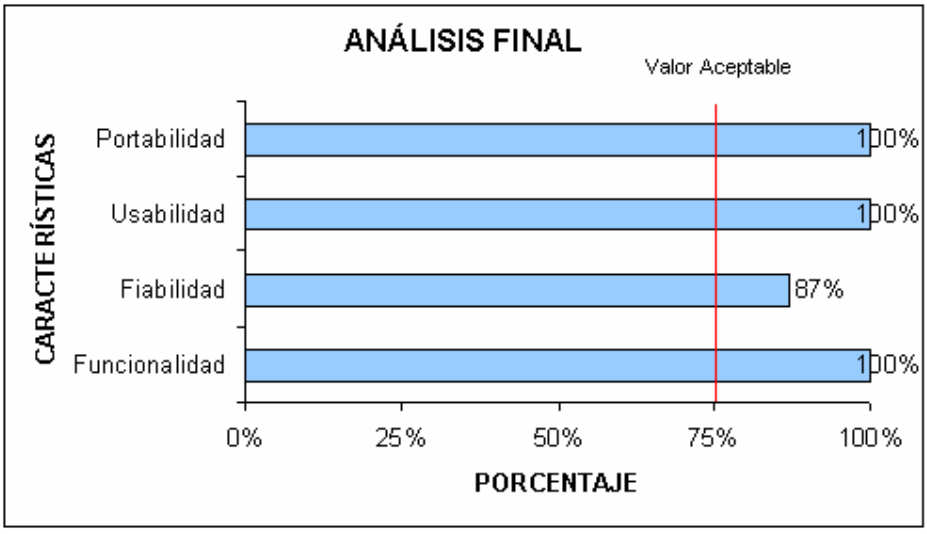


Figura 3.9 Análisis Final de Resultados

Fuente: La autora

El gráfico de Análisis final representa la sumarización de todos los datos obtenidos durante el proceso de evaluación del código fuente basado en el modelo de calidad interna de la norma ISO/IEC 9126. Se puede observar que las características de Funcionalidad, Fiabilidad, Usabilidad y Portabilidad sobrepasan el valor aceptable de calidad. Por lo tanto el código fuente del producto de software satisface los requisitos funcionales (con respecto a la Funcionalidad), mantiene un nivel de rendimiento aceptable al momento de anular operaciones incorrectas (con respecto a la Fiabilidad), es fácil de ser entendido y operado (con respecto a la Usabilidad) y es adaptable a diferentes ambientes bajo la *plataforma Windows* (con respecto a la Portabilidad).

En general la calidad del código fuente de la aplicación evaluada cumple con el 100% de las características de la calidad seleccionadas, ubicándose dentro del nivel de aceptabilidad, lo cual satisface los requisitos de calidad.

CAPÍTULO 4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- El proceso de evaluación de un producto de software descrito en la norma ISO/IEC 14598 necesita de un modelo de calidad y de la selección de métricas adecuadas para su correcta ejecución.

- El valor de complejidad ciclomática puede ser utilizada en el desarrollo, mantenimiento y reingeniería para estimar el riesgo, costo y estabilidad de los programas de software.

- RUP es una metodología orientada a objetos, permite un desarrollo iterativo del Software, mide el riesgo en cada una de las fases del ciclo de vida del Producto, lo cual reduce significativamente el riego total del Proyecto.

- Las normas ISO/IEC 9126 e ISO/IEC 14598 son estándares aplicables a cualquier tipo de producto de software, independientemente de la tecnología, lenguaje de programación, motor de base de datos, metodología de desarrollo y arquitectura que utilice.

- Por lo general cuando el código fuente de un producto de software aumente, su complejidad también lo hará porque se tiende a utilizar más sentencias de control, por ende será más propenso a contener errores y difícil de mantener.

- El desarrollo de la herramienta (EVAC) para la evaluación de la calidad del producto de software, permite automatizar el proceso de realizar el grafo de flujo y obtener el valor de la complejidad ciclomática el cual nos da un índice de la calidad del código fuente C ANSI.

- El valor de la CC es independiente del número de LDC del código fuente analizado.

- Entre mayor sea el valor de la CC del código fuente mayor será la utilización de recursos de Hardware, especialmente de la memoria RAM.

4.2. RECOMENDACIONES

- Las empresas que se dedican al desarrollo de productos de software deben establecer normas y procesos de calidad en cada fase del ciclo de vida del desarrollo del software lo cual de como resultados productos de alta calidad con un consumo de costos y tiempos bajos.

- Para asegurar los resultados de la evaluación de la calidad de un producto de software es recomendable cumplir con todas las fases del proceso de evaluación de la norma ISO/IEC 14598 y ejecutar el plan producido.

- Es recomendable aplicar las normas ISO/IEC 9126 e ISO/IEC 14598 cuando se desee evaluar productos de software en general, es decir, independientemente de la tecnología, lenguaje de programación, motor de base de datos, metodología de desarrollo y arquitectura seleccionada.

- Utilizar RUP para el desarrollo de sistemas de software porque es una metodología que permite una aproximación a la programación orientada a objetos (POO), además soporta características de acercamiento iterativo para el desarrollo, el cual se dirige a los ítems de alto riesgo en cada fase del ciclo de vida, reduciendo significativamente el perfil de riesgo del proyecto.

- Si se desea realizar una versión de la herramienta para el análisis de calidad de otro tipo de leguaje de programación se recomienda solo cambiar el componente de parseo o analizador sintáctico.

REFERENCIAS BIBLIOGRÁFICAS

LIBROS Y MANUALES

[1] PRESSMAN, Roger. INGENIERÍA DEL SOFTWARE. Un enfoque práctico. Quinta edición. Editorial McGraw-Hill Interamericana. España. 2002

[2] RACINES, CHIRIBOGA. Agente Inteligente De Recomendación De Productos O Servicios Basado En Ontologías. Escuela Politécnica Nacional. Facultad de Ingeniería de sistemas. 2006

[3] CARRILLO, BRITO. Evaluación De La Calidad Del Código Fuente En Base A Las Normas Iso/Iec 9126, Iso/Iec 14598. Escuela Politécnica Nacional. Facultad de Ingeniería de sistemas. 2005

[4] RUIZ, Erika. Aplicación de un modelo sistemático para la evaluación de la calidad el software. Escuela Politécnica Nacional. Facultad de Ingeniería de sistemas. 2004

[5] ROMERO, Alberto. Desarrollo de un sistema de manejo de pacientes de un consultorio pediatrico. Escuela Politécnica Nacional. Facultad de Ingeniería de sistemas. 2003

DIRECCIONES ELECTRÓNICAS

[6] Metodología de Desarrollo RUP.
www.ibm.com/.../rational/library/content/RationalEdge/jan01/WhatIstheRationalUnif iedProcessJan01.pdf, 2006.
www.cs.nmt.edu/~cs328/reading/rup_bestpractices.pdf, 2006.

[7] Pruebas de Software.
tdi.eui.upm.es/ls/descargas/Tema5_PruebasSoftware.PDF, 2007.

www.cs.man.ac.uk/~velascop/publ/Tesis.pdf, 2006.
www.it.uc3m.es/ttrd/material/05-pruebas-de-programas.pdf, 2007.
http://www.sel.unsl.edu.ar/licenciatura/ingsoft1/Apuntes/2007/teoria7.pdf, 2007.

[8] Modelos de Calidad de Software.
http://www.arisa.se/files/LL-06.pdf, 2006.
www.di.uminho.pt/~joostvisser/publications/HeitlagerKuipersVisser-Quatic2007.pdf, 2007.

[9] Calidad del Software.
http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad_software.PDF, 2006.

[10] Métrica de Complejidad Ciclomática.
www.itba.edu.ar/capis/rtis/articulosdeloscuadernosetapaprevia/RIZZI-COMPLEJIDAD.pdf, 2006.
www.cis.strath.ac.uk/teaching/ug/classes/52.429/MetricsandMeasurement.pdf, 2006.
http://www.lsi.us.es/docencia/get.php?id=2271, 2007.

[11] ISO 9126, 14598.
http://www.psmsc.com/UG2003/Presentations/07IshigakiFinal.pdf, 2003
www.inf.unisi.ch/faculty/lanza/QAOOSE2006/QAOOSE2006Proc.pdf , 2006
www.dsi.unifi.it/~nesi/life-cycle-assessment-metrics-software-quality-v1-0.pdf, 2006

[12] Generador de Analizador Léxico - Sintáctico.
https://javacc.dev.java.net , 2006.

[13] Librerías iText para JDK.
http://sourceforge.net/project/showfiles.php?group_id=15255, 2007.

[14] Código fuente del Parser C
www.koders.com , 2006

ANEXO A. DOCUMENTO DE ESPECIFICACIÓN DE REQUERIMIENTOS

DOCUMENTO DE ESPECIFICACIÓN DE REQUERIMIENTOS

INTRODUCCIÓN

Es indispensable disponer de un documento de especificación de requerimientos para un mejor entendimiento de la aplicación a desarrollarse y a su vez para poder controlar la evolución de la misma.

El objetivo básico del análisis es el de construir una especificación detallada en función de los requisitos que deberá cumplir el producto de software en su fase de explotación, para así poder:

- Ser la base del diseño para la construcción de la herramienta.
- Definir un conjunto de requisitos que serán validados una vez construida la herramienta.
- Proveer información a los usuarios.
- Ser un medio de comunicación entre las partes que intervienen en el sistema.

PROPÓSITO

El establecimiento de la Especificación de Requerimientos es para identificar y describir los procesos y funciones que la herramienta va a ejecutar, con el propósito de proporcionar al técnico y los usuarios los medios necesarios para evaluar la calidad de la herramienta (aplicación) y a su vez realizar un control de la evolución de la misma.

Además este documento servirá de base para las fases de análisis y diseño de software.

ALCANCE

El desarrollo de la herramienta para evaluar la calidad del código fuente surge de la necesidad de mejorar el procesamiento de los programas (generados en C ANSI) o evitar que sean propensos a errores.
La herramienta evaluará el código fuente a través del uso de la métrica de complejidad ciclomática, dando como resultado un grafo de flujo, el valor numérico de la complejidad ciclomática, la sentencia y su secuencia nodos que la conforman.

Para evitar ambigüedades al momento de la evaluación del código fuente es necesario que la herramienta cumpla las funciones de un analizador léxico-sintáctico, para verificar que el código este correctamente estructurado y sin errores, así se evitará resultados ficticios.

Cabe recalcar que el analizador (o parser) es solo a nivel léxico y sintáctico, mas no semántico. Esto quiere decir, por ejemplo, que el analizador no se encarga de verificar que las funciones que estén escritas (o desarrolladas), se hayan declarado al inicio, o que siempre exista la función main, que existan librerías, etc.

Definiciones, acrónimos y abreviaturas

- **EVAC**: Herramienta para evaluar la calidad del código fuente generado en C ANSI.
- **Usuario**: es la persona que interactúa con la herramienta EVAC.

SITUACIÓN

DECLARACIÓN DE PROBLEMA

El problema de	Generar programas sin una estructura lógica óptima.
Afecta	En gran medida a los sistemas computacionales, por el alto consumo de sus recursos, como la memoria, el espacio en disco, etc; y a los programas en si por no ser de fácil mantenimiento.
De lo cual el impacto es	La insatisfacción por parte del usuario en el uso de estos programas, debido a su tiempo de respuesta tardía y difícil mantenimiento.
Una solución satisfactoria sería	La creación de una herramienta que permita evaluar la calidad del código fuente, indicando el nivel de complejidad y su estructura lógica, para saber de que manera se puede mejorar el programa.

DECLARACIÓN DE SITUACIÓN DEL PRODUCTO

Para	Personas que intervienen en el desarrollo de programas en C ANSI.
Quienes	Quieren conocer que tan bueno es el código fuente de los programas que están generando o utilizando.
La herramienta para evaluar la calidad del código fuente generado en C ANSI	Es un producto de software
Que	Provee la funcionalidad para analizar el código fuente, desplegando un grafo de flujo, un valor cuantitativo de la complejidad del programa y las sentencias de control que la conforman.
Al contrario de	Los compiladores de programas en C ANSI, que se limitan a verificar que los programas estén correctamente escrito y estructurados, mas no a evaluar la calidad de los mismos.
Este producto	Tiene como principal función presentar al usuario el grafo de flujo del programa, un valor cuantitativo y sentencias de control de la métrica de complejidad ciclomática, además de otros detalles que cosntarán en el reporte (.PDF). Permite abrir archivos o programas generados en C ANSI, editarlos, guardar los cambios; generar, abrir y guardar reportes. Actúa como un analizador léxico-sintáctico y desplegará los errores en caso de que los hubiesen.

DESCRIPCIÓN DE REPRESENTANTES Y USUARIOS

RESUMEN DE LOS REPRESENTANTES

Nombre	Descripción	Responsabilidades
Administrador del Proyecto	Este es el representante primario, debido a que guía el desarrollo de la herramienta	Asignar los recursos, define las prioridades, coordina las iteraciones con los clientes y usuarios. Además establece un conjunto de prácticas que asegura la integridad y calidad de los artefactos del proyecto.
Especialista de requerimientos	Este representante interactúa con el analista, para trasladar correctamente los requisitos y necesidades, dentro de los requerimientos a ser utilizados en el diseño.	Capturar la especificación de una parte de la funcionalidad de la herramienta describiendo el aspecto de requisitos de uno o algunos casos de uso. También es responsable para empaquetar los casos de uso, y mantener la integridad de esos paquetes.
Analista del	Este es un representante	Guiar, definir, coordinar el modelamiento de

sistema	recoge las necesidades de los usuarios de la herramienta	los casos de uso del negocio, actores y como interactúan entre ellos. Elaboración del Modelo de Análisis y Diseño. Colaboración en la elaboración de las pruebas funcionales y el modelo de datos.
Diseñador		Define las responsabilidades, funciones, atributos y relaciones de una o más clases y determina como pueden adaptarse al ambiente de implementación. También crea prototipos de interfaces de la herramienta en base a los requerimientos de los usuarios finales y sus clientes
Desarrollador	Este representante interactúa con el diseñador, para emprender el desarrollo de la herramienta	Construcción de prototipos. Colaboración en la elaboración de las pruebas funcionales, modelo de datos y en las validaciones con el usuario
Verificador	Este representante es el encargado de llevar a cabo las pruebas funcionales de la herramienta.	Definir un conjunto de pruebas y las ejecutarla en un ambiente controlado para reproducir los posibles errores y reportarlos.

RESUMEN DEL USUARIO

Nombre	Descripción	Responsabilidades	Representante
Usuario	Usuario final de la herramienta	Abrir o guardar un programa; evaluar la calidad del código fuente; abrir, guardar reportes	Representado por el mismo

PERFILES DE LOS REPRESENTANTES

Usuario

Descripción	Un usuario que utiliza la herramienta para evaluar la calidad del código fuente.
Tipo	Principal usuario, con conocimientos del lenguaje C ANSI.
Responsabilidades	Obtener el nivel de calidad del código fuente que se esta utilizando o generando, para que así mejorarlo.
Criterio de éxito	El éxito esta definido por el uso continuo de la herramienta.
Involucrar	Ayudara a evaluar el diseño, marcando resultados que guiarán a la visión.
Entregables	Ninguno
Cometarios	Ninguno

NECESIDADES DEL USUARIOS

Necesidad	Prioridad	Involucra	Solución actual	Soluciones propuestas
Verificar que el código sea correcto.	Media	Chequea la sintaxis y el léxico del programa.	Ninguna	Implementar únicamente un analizador léxico-sintáctico para los programas escritos en C ANSI.
Observar mediante un gráfico la estructura lógica del programa.	Alta	El número de ciclos fundamentales de conexión de los diagramas de flujo.	Ninguna	Generación del grafo de flujo.
Obtener valor cuantitativo del nivel de calidad del código.	Alta	Un número que indique el nivel de calidad del código fuente.	Ninguna	Obtener el valor de la métrica de complejidad ciclomática a través del uso del grafo de flujo.
Obtener las sentencias de control que se encuentran en el programa.	Alta	La sentencia de control y secuencia de nodos que la conforman.	Ninguna	Obtener las sentencias de control seguida de la secuencia de nodos que la conforman.
Generar un reporte con los resultados obtenidos.	Alta	Generar un reporte.	Ninguna	Colocar en un archivo .PDF los resultados obtenidos de la evaluación.

APRECIACIÓN GLOBAL DEL PRODUCTO

PERSPECTIVA DEL PRODUCTO

La herramienta esta planificada para funcionar sobre cualquier plataforma con soporte para JAVA.

La herramienta no es componente o parte de otro sistema de software.

La arquitectura de la herramienta es la siguiente:

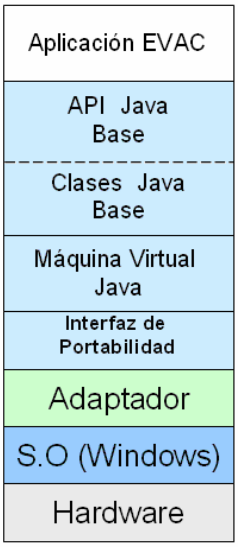


Figura1. Apreciación General de la Arquitectura de la aplicación EVAC

Fuente:

ASUNCIONES Y DEPENDENCIAS

- La herramienta debe estar orientada a ser de manejo simple, por lo cual el diseño estará compuesto de características tales que garanticen el fácil aprendizaje.
- El buen rendimiento de la herramienta dependen de los requerimientos mínimos especificados se cumplan.
- Es indispensable disponer de un sistema operativo que soporte la ejecución de sentencias JAVA.
- La herramienta no se encarga del análisis semántico del código fuente.

CARACTERÍSTICAS DEL PRODUCTO

- Permite editar nuevos archivos o programas para después evaluar su calidad.
- Abrir y seleccionar archivos generados en C ANSI.

- Verificar que el archivo que va a ser analizado esté desarrollado en C ANSI (extensión del archivo debe ser .C).
- Realizar un análisis léxico-sintáctico del archivo antes de su evaluación.
- Analizar el archivo con el uso de la métrica de complejidad ciclomática, para la evaluar la calidad del código fuente.
- Editar el archivo para bajar su nivel de complejidad y mejorar la calidad del código fuente.
- Guardar los cambios realizados en el archivo.
- Una vez finalizada la evaluación, se desplegará un número, el cual indica el nivel de complejidad ciclomática, un grafo que muestra la estructura lógica del programa, las sentencias de control y secuencias de nodos que las conforman, fecha de evaluación, nombre de archivo evaluado.
- Generar reportes con el resultado obtenido de la evaluación.
- Guardar el reporte en una ubicación especificada.
- El reporte será almacenado en un archivo .pdf.

RESTRICCIONES

- Analiza únicamente programas desarrollados en C ANSI (extensión .c).
- Disponer de un sistema operativo que soporte la ejecución de sentencias JAVA.
- La herramienta no mejora de manera automática la complejidad del software

ANEXO B. ESPECIFICACIÓN DE CASOS DE USO MODELO DE DISEÑO A NIVEL DE ANÁLISIS

ESPECIFICACIÓN DE CASOS DE USO

Los casos de uso Administrar Archivo y Administrar Reporte están conformados por los casos de uso que se detallan a continuación.

CASO DE USO NUEVO ARCHIVO

DESCRIPCIÓN

El caso de uso Nuevo Archivo Figura 1, permite al usuario crear un nuevo archivo o programa en C ANSI para posteriormente ser evaluado, siempre se abrirá en una nueva ventana, para así manejar sus procesos de manera independiente.

Figura 1. Diagrama de casos de uso Nuevo Archivo

Fuente: La autora

FLUJO DE EVENTOS

Flujo básico

- Dar clic en Archivo.
- Elegir la opción Nuevo.

Flujos alternativos

- No existe flujo alternativo.

PRECONDICIONES

- No existen precondiciones.

POSCONDICIONES

- No existen poscondiciones.

CASO DE USO ABRIR ARCHIVO

DESCRIPCIÓN

El caso de uso Abrir Archivo (Figura 2), permite buscar, seleccionar y abrir un archivo generado en C ANSI, la herramienta debe tener una funcionalidad implícita de verificar que el programa este generado exclusivamente en ese lenguaje de programación.

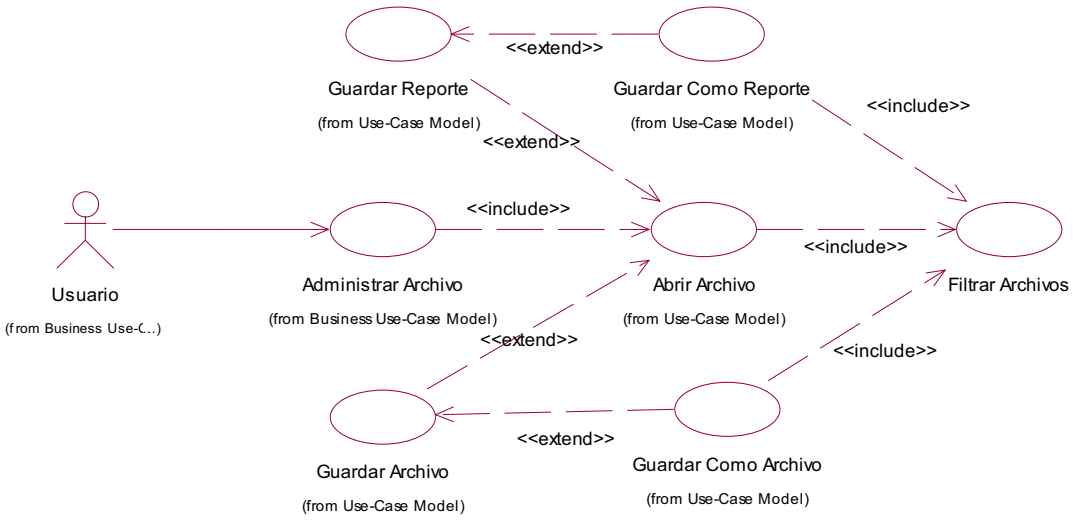


Figura 2. Diagrama de casos de uso Abrir Archivo

Fuente: La autora

FLUJO DE EVENTOS

Flujo básico

- Dar clic en Archivo.
- Elegir la opción Abrir.
- En la nueva ventana buscamos, seleccionamos el archivo y damos clic en el botón “Abrir”.

Flujos alternativos

Primer flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Abrir.
- Clic en el botón “Si”, para guardar los cambios del archivo abierto

modificado.

- Seleccionar el path y dar clic en botón “Abrir”.

Segundo flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Abrir.
- Clic en el botón “No”, para no guardar los cambios del archivo abierto modificado.
- Seleccionar un archivo, dar clic en botón “Abrir”.

Tercer flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Abrir.
- Clic en el botón “Si”, para guardar el nuevo archivo.
- Seleccionar un path, darle un nombre al archivo, clic en el botón “Guardar”.
- Seleccionar el archivo, dar clic en botón “Abrir”.

Cuarto flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Abrir.
- Clic en el botón “No”, para no guardar el nuevo archivo.
- Seleccionar el archivo, dar clic en botón “Abrir”.

Quinto flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Abrir.
- Clic en el botón “Si”, para guardar los cambios del reporte.
- Seleccionar el archivo, dar clic en botón “Abrir”.

Sexto flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Abrir.
- Clic en el botón “No”, para no guardar los cambios del reporte.
- Seleccionar el archivo, dar clic en botón “Abrir”.

Séptimo flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Abrir.

- Clic en el botón “Si”, para guardar el nuevo reporte.
- Seleccionar un path para el reporte, dar clic en el botón “Guardar”.
- Seleccionar el archivo, dar clic en botón “Abrir”.

Octavo flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Abrir.
- Clic en el botón “No”, para no guardar el nuevo reporte.
- Seleccionar el archivo, dar clic en botón “Abrir”.

PRECONDICIONES

- Ningún archivo abierto anteriormente.
- Tener un archivo abierto (tiene path) sin modificaciones.
- Tener un archivo abierto (tiene path) con modificaciones y si guardar los cambios.
- Tener un archivo abierto (tiene path) con modificaciones y no guardar los cambios.
- Tener un archivo nuevo editado (no tiene path) y si guardar los cambios.
- Tener un archivo nuevo editado (no tiene path) y no guardar los cambios.
- Tener un reporte que ya tiene path y ha sido actualizado (regenerado porque su código fuente fue evaluado de nuevo) y si guardar los cambios.
- Tener un reporte que ya tiene path y ha sido actualizado (regenerado porque su código fuente fue evaluado de nuevo) y no guardar los cambios.
- Tener un reporte nuevo (no tiene path) y si guardarlo.
- Tener un reporte nuevo (no tiene path) y no guardarlo.

POSCONDICIONES

- No existen poscondiciones.

CASO DE USO CERRAR ARCHIVO

DESCRIPCIÓN

El caso de uso Cerrar Archivo (Figura 3), permite setear todos los campos de la herramienta, para así abrir o editar un nuevo archivo.

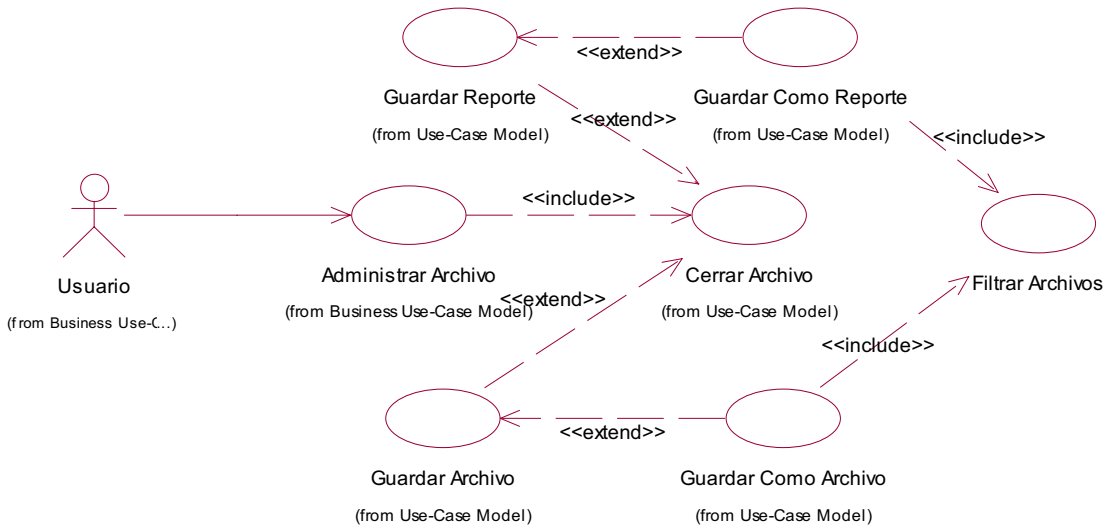


Figura 3. Diagrama de casos de uso Cerrar Archivo

Fuente: La autora

FLUJO DE EVENTOS

Flujo básico

- Dar clic en Archivo.
- Elegir la opción Cerrar.

Flujos alternativos

Primer flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Cerrar.
- Clic en el botón “Si”, para guardar los cambios del archivo abierto modificado.

Segundo flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Cerrar.

- Clic en el botón “No”, para no guardar los cambios del archivo abierto modificado.

Tercer flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Cerrar.
- Clic en el botón “Si”, para guardar el nuevo archivo.
- Seleccionar un path, darle un nombre al archivo, clic en el botón “Guardar”.

Cuarto flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Cerrar.
- Clic en el botón “No”, para no guardar el nuevo archivo.

Quinto flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Cerrar.
- Clic en el botón “Si”, para guardar los cambios del reporte.

Sexto flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Cerrar.
- Clic en el botón “No”, para no guardar los cambios del reporte.

Séptimo flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Cerrar.
- Clic en el botón “Si”, para guardar el nuevo reporte.
- Seleccionar un path para el reporte, dar clic en el botón “Guardar”.

Octavo flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Cerrar.
- Clic en el botón “No”, para no guardar el nuevo reporte.

PRECONDICIONES

- Archivo abierto (tiene path) sin modificaciones.
- Archivo abierto (tiene path) con modificaciones y si guardar los cambios.

- Archivo abierto (tiene path) con modificaciones y no guardar los cambios.
- Archivo nuevo editado (no tiene path) y si guardar los cambios.
- Archivo nuevo editado (no tiene path) y no guardar los cambios.
- Reporte que ya tiene path y ha sido actualizado (regenerado porque su código fuente fue evaluado de nuevo) y si guardar los cambios.
- Reporte que ya tiene path y ha sido actualizado (regenerado porque su código fuente fue evaluado de nuevo) y no guardar los cambios.
- Reporte nuevo (no tiene path) y si guardarlo.
- Reporte nuevo (no tiene path) y no guardarlo.

POSCONDICIONES

- No existen poscondiciones.

CASO DE USO GUARDAR ARCHIVO

DESCRIPCIÓN

El caso de uso Guardar Archivo (Figura 4), permite guardar los cambios directamente sobre la ubicación de la cual fue abierto, es decir, reemplazar el archivo original.

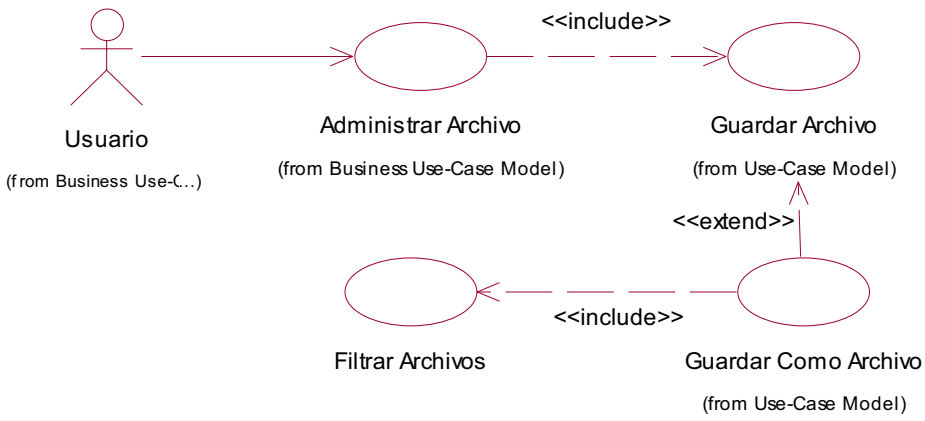


Figura 5. Diagrama de casos de uso Guardar Archivo

Fuente: La autora

FLUJO DE EVENTOS

Flujo básico

- Clic en Archivo.
- Elegir la opción Guardar.

Flujos alternativos

Primer flujo alternativo

- Clic en Archivo.
- Elegir la opción Guardar.
- Seleccionar path y darle un nombre al nuevo archivo, clic en el botón “Guardar”.

PRECONDICIONES

- Archivo abierto con o sin modificaciones
- Archivo nuevo editado o no.

POSCONDICIONES

- No existen poscondiciones.

CASO DE USO GUARDAR COMO ARCHIVO

DESCRIPCIÓN

El caso de uso Guardar Como Archivo (Figura 5), permite buscar y seleccionar una ubicación para almacenar el archivo.

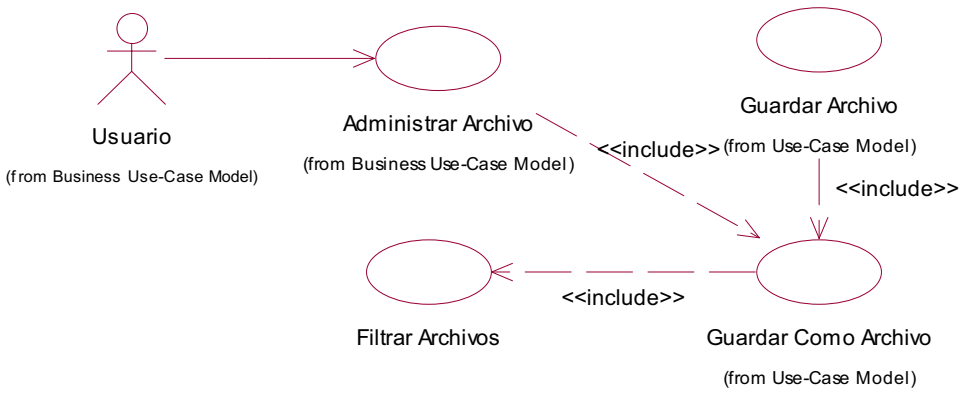


Figura 5. Diagrama de casos de uso Guardar Como Archivo

Fuente: La autora

FLUJO DE EVENTOS

Flujo básico

- Clic en Archivo.
- Elegir la opción Guardar Como.
- Seleccionar un path, darle un nombre y dar clic en el botón “Guardar”.

Flujos alternativos

- No existen flujos alternativos

PRECONDICIONES

- No existen precondiciones.

POSCONDICIONES

- No existen poscondiciones.

CASO DE USO EVALUAR CALIDAD ARCHIVO

DESCRIPCIÓN

El caso de uso Evaluar Calidad Archivo (Figura 6), realiza el análisis léxico-sintáctico del código fuente y se da inicio al proceso de evaluación de la calidad del código fuente, proporcionando los resultados que forman parte del reporte.

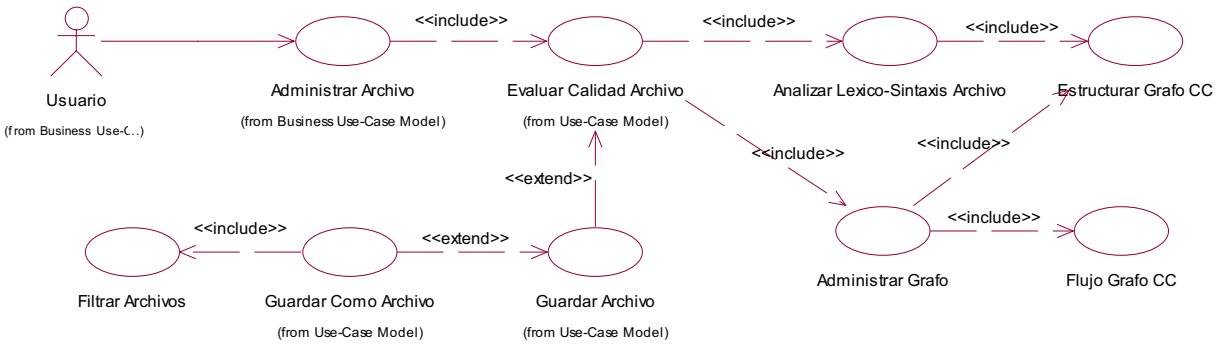


Figura 6. Diagrama de casos de uso Evaluar Calidad Archivo

Fuente: La autora

FLUJO DE EVENTOS

Flujo básico

- Dar clic en Archivo.
- Elegir la opción Evaluar Calidad.

Flujos alternativos

Primer flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Evaluar Calidad.
- Clic en el botón “Si”, para guardar los cambios del archivo abierto modificado.

Segundo flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Evaluar Calidad.
- Clic en el botón “No”, para no guardar los cambios del archivo abierto modificado.

Tercer flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Evaluar Calidad.
- Clic en el botón “Si”, para guardar el nuevo archivo.
- Seleccionar un path, darle un nombre al archivo, clic en el botón “Guardar”.

Cuarto flujo alternativo

- Dar clic en Archivo.
- Elegir la opción Evaluar Calidad.
- Clic en el botón “No”, para no guardar el nuevo archivo.

PRECONDICIONES

- Archivo abierto (tiene path) sin modificaciones.
- Archivo abierto (tiene path) con modificaciones y si guardar los cambios.
- Archivo abierto (tiene path) con modificaciones y no guardar los cambios.
- Archivo nuevo editado (no tiene path) y si guardar los cambios.
- Archivo nuevo editado (no tiene path) y no guardar los cambios.

POSCONDICIONES

- No existen poscondiciones.

CASO DE USO GUARDAR REPORTE

DESCRIPCIÓN

El caso de uso Guardar Reporte (Figura 7), permite guardar un reporte actualizado en la ubicación donde fue almacenada anteriormente.

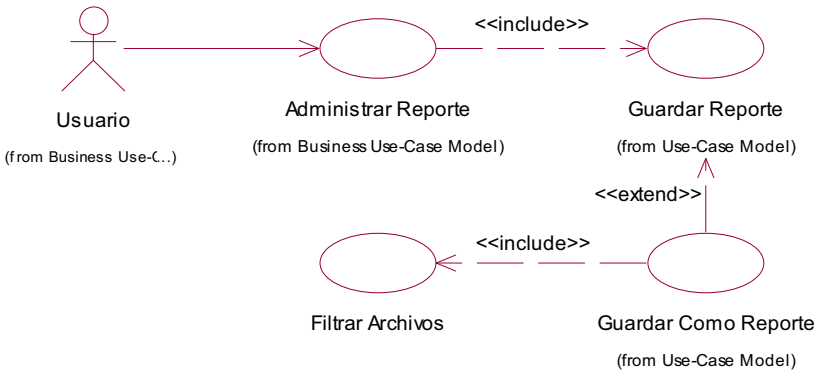


Figura 7. Diagrama de casos de uso Guardar Reporte

Fuente: La autora

FLUJO DE EVENTOS

Flujo básico

- Dar clic en Reporte.
- Elegir la opción Guardar.

Flujos alternativos

Primer flujo alternativo

- Clic en Reporte.
- Elegir la opción Guardar.
- Seleccionar un path, darle un nombre al nuevo reporte, clic en el botón “Guardar”.

PRECONDICIONES

- Reporte que ya tiene path, este o no modificado.
- Reporte nuevo (no tiene path).

POSCONDICIONES

- No existen poscondiciones.

CASO DE USO GUARDAR COMO REPORTE

DESCRIPCIÓN

El caso de uso Guardar Como Reporte (Figura 8), permite buscar y seleccionar una ubicación para almacenar el reporte.

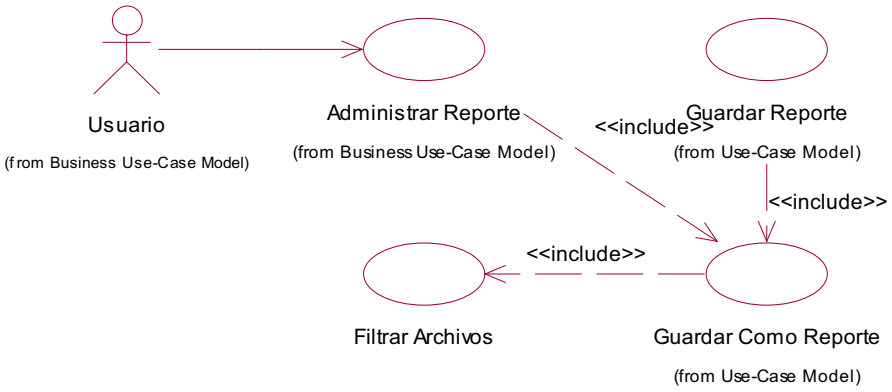


Figura 8. Diagrama de casos de uso Guardar Como Reporte

Fuente: La autora

FLUJO DE EVENTOS

Flujo básico

- Clic en Reporte.
- Elegir la opción Guardar Como.
- Seleccionar un path, darle un nombre al reporte y dar clic en el botón “Guardar”.

Flujos alternativos

- No existen flujos alternativos.

PRECONDICIONES

- Archivo correctamente evaluado.

POSCONDICIONES

- No existen poscondiciones.

VISTA LÓGICA: MODELO DE DISEÑO

En este modelo se definen las clases de análisis que sirven como base para describir la realización de casos de uso y demás elementos de la arquitectura.

En este parte se tomarán en cuenta los casos de uso generales como lo es Administrar Archivo y Administrar Reporte. Los estereotipos de las clases de análisis son:

Estereotipo	Descripción
<<entity>>	Las clases entidad son usadas para modelar la información y la conducta asociada que deben guardarse.
<<control>>	Las clases control son utilizadas para modelar conductas de control específicas de uno o varios casos de uso.
<<boundary>>	Las clases límite modelan la interacción entre los ambientes del sistema y sus funcionamientos internos.

Tabla 1. Estereotipos de clases de análisis

Fuente: La autora

ABSTRACCIONES CLAVE

En base a los requerimientos funcionales del Documento de Especificación de Requerimientos (Anexo A) se determinaron las siguientes clases entidad:

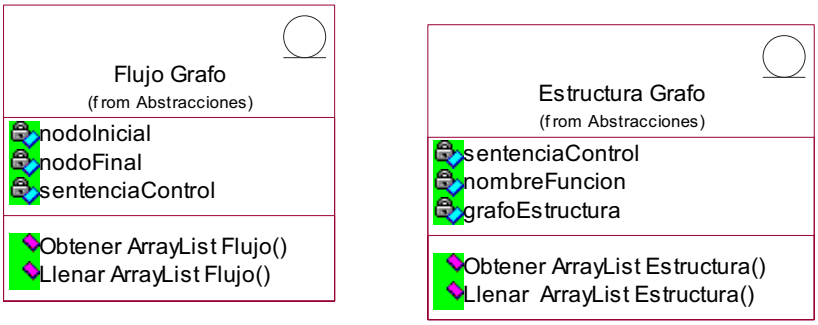


Figura 9. Abstracciones clave

Fuente: La autora

En la Tabla 2 se describe cada abstracción clave.

Clase	Descripción
Estructura Grafo	Estructura que contiene las sentencias de control y su anidación dentro de archivo, y el nombre de las funciones que las contiene.
Flujo Grafo	Define el flujo del grafo a través de los nodos iniciales y finales. Útil para poder graficar las aristas.

Tabla 2. Abstracciones clave

Fuente: La autora

ELEMENTOS DE ANÁLISIS

Para los casos de uso Administrar Archivo y Administrar Reporte se definen dos tipos de clases de análisis que corresponden a los estereotipos <<boundary>> y <<control>> (Tabla 3).

Caso de uso	<<boundary>>	<<control>>
Administrar Archivo	HerramientaUI	Gestión Archivo
		Filtro Archivos
		Gestión Grafo
		Analizador Léxico Sintáctico C
Administrar Reporte	HerramientaUI	Gestión Reporte
		Filtro Archivos

Tabla 3. Clases de análisis con estereotipo <<boundary>> y <<control>>

Fuente: La autora

REALIZACIÓN DE CASOS DE USO

En función de los actores y los tipos de clases de análisis mencionados anteriormente, se procede a hacer las realizaciones de los casos de uso especificados. La realización de casos de uso comprende dos tipos de vistas: dinámica y estática que son representadas con un diagrama de secuencia y un diagrama de clases respectivamente. El diagrama de secuencia representa el flujo de eventos del caso de uso, mientras que el diagrama de clases muestra las clases de análisis participantes en el caso de uso con sus atributos, métodos y relaciones. En la Tabla 4 se presentan las referencias de los diagramas mencionados.

Realización de Casos de Uso	**Participantes**	**Flujo de eventos**
Administrar Archivo	Anexo B – Figura 10	Anexo B – Figura 11
Administrar Reporte	Anexo B – Figura 12	Anexo B – Figura 13

Tabla 4. Participantes y flujo de eventos de la realización de los casos de uso.

Fuente: La autora

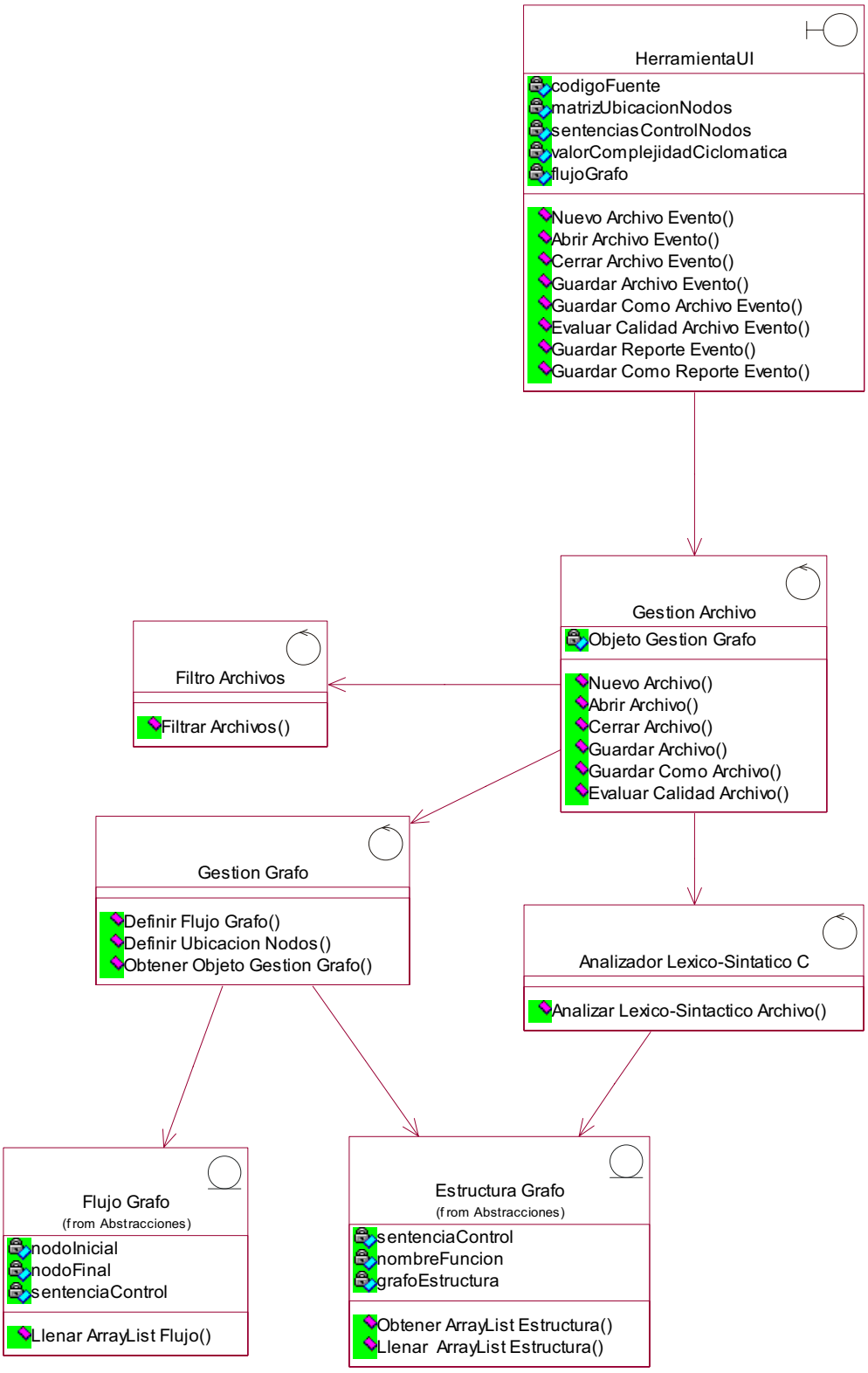
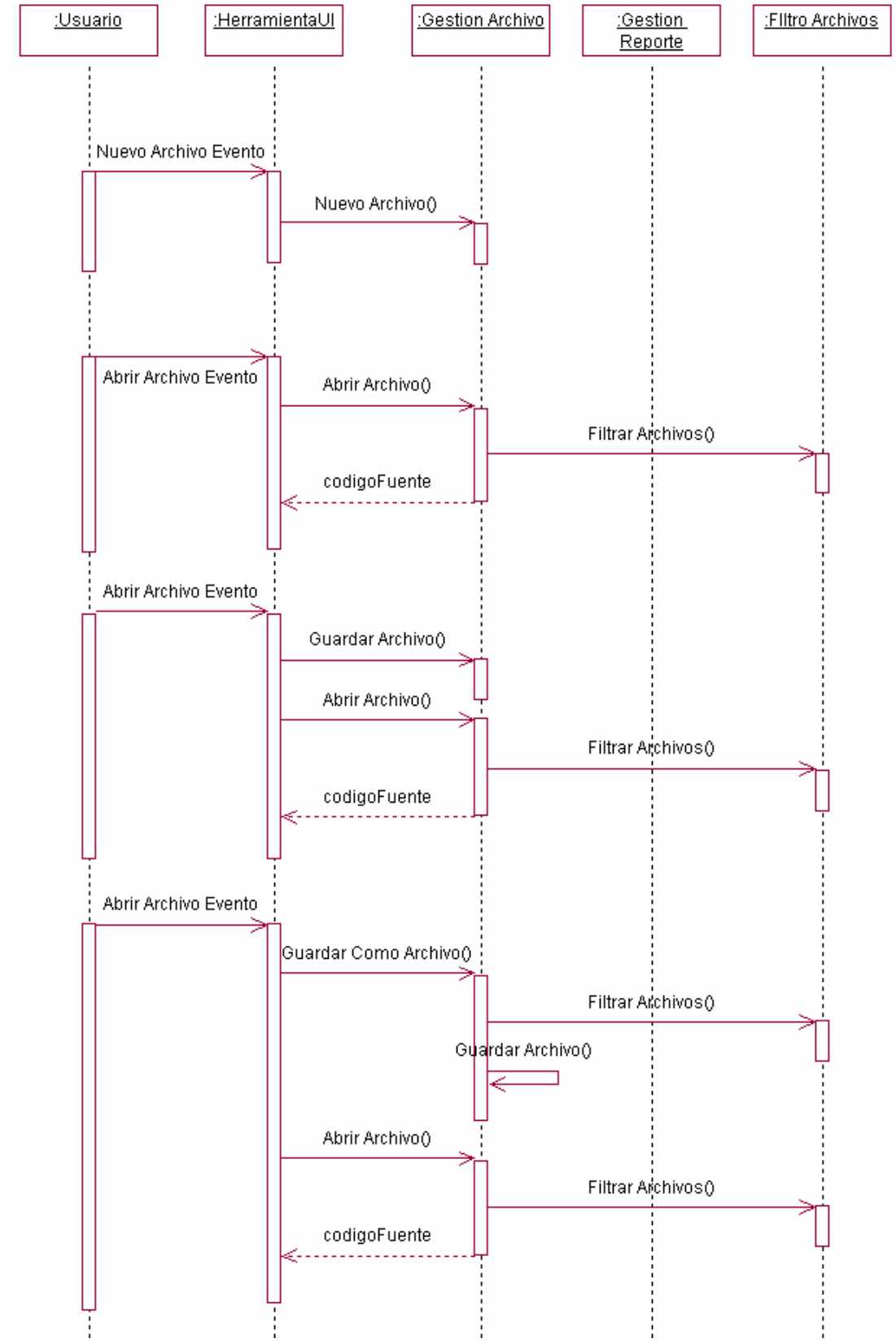
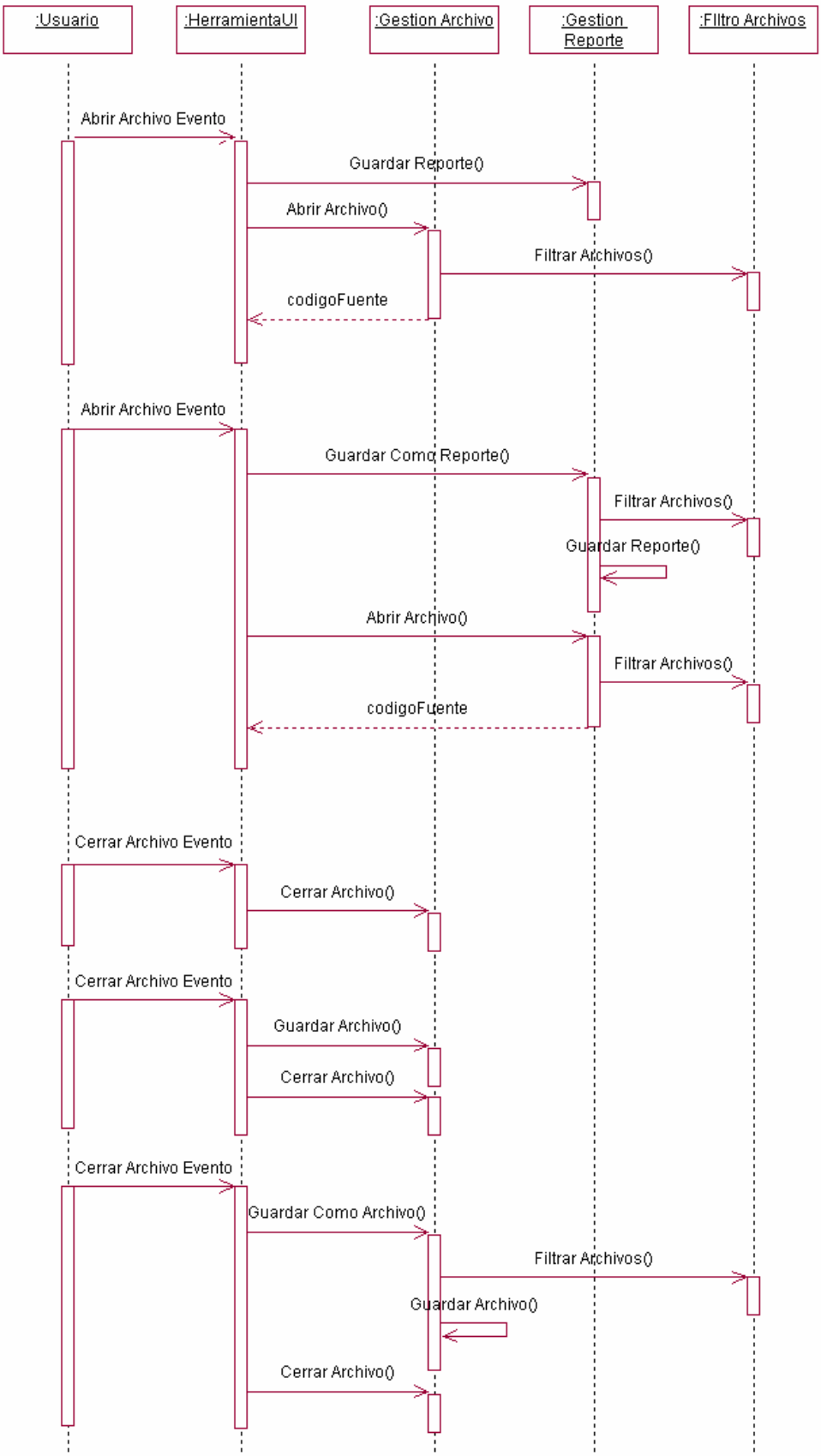
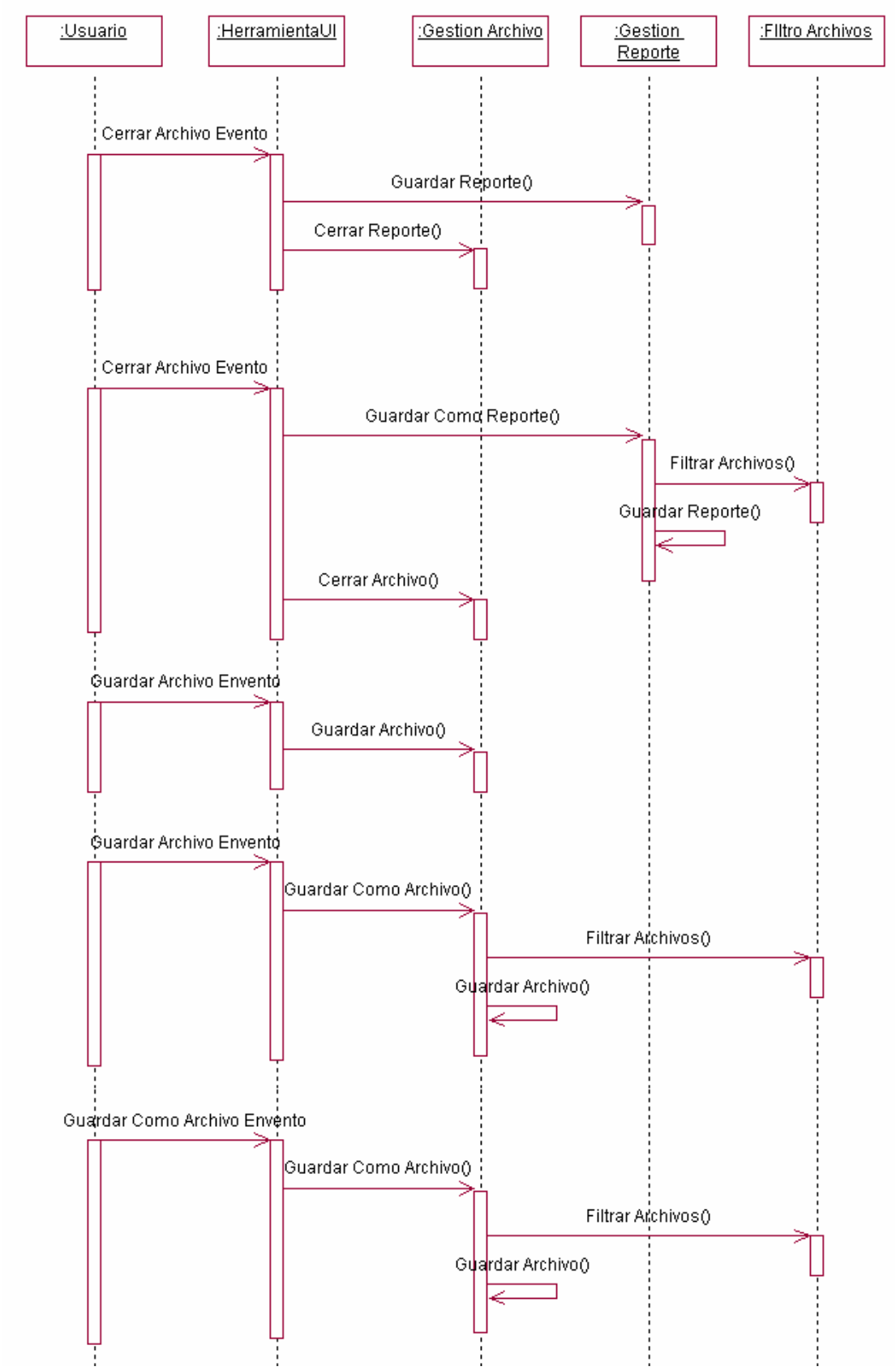


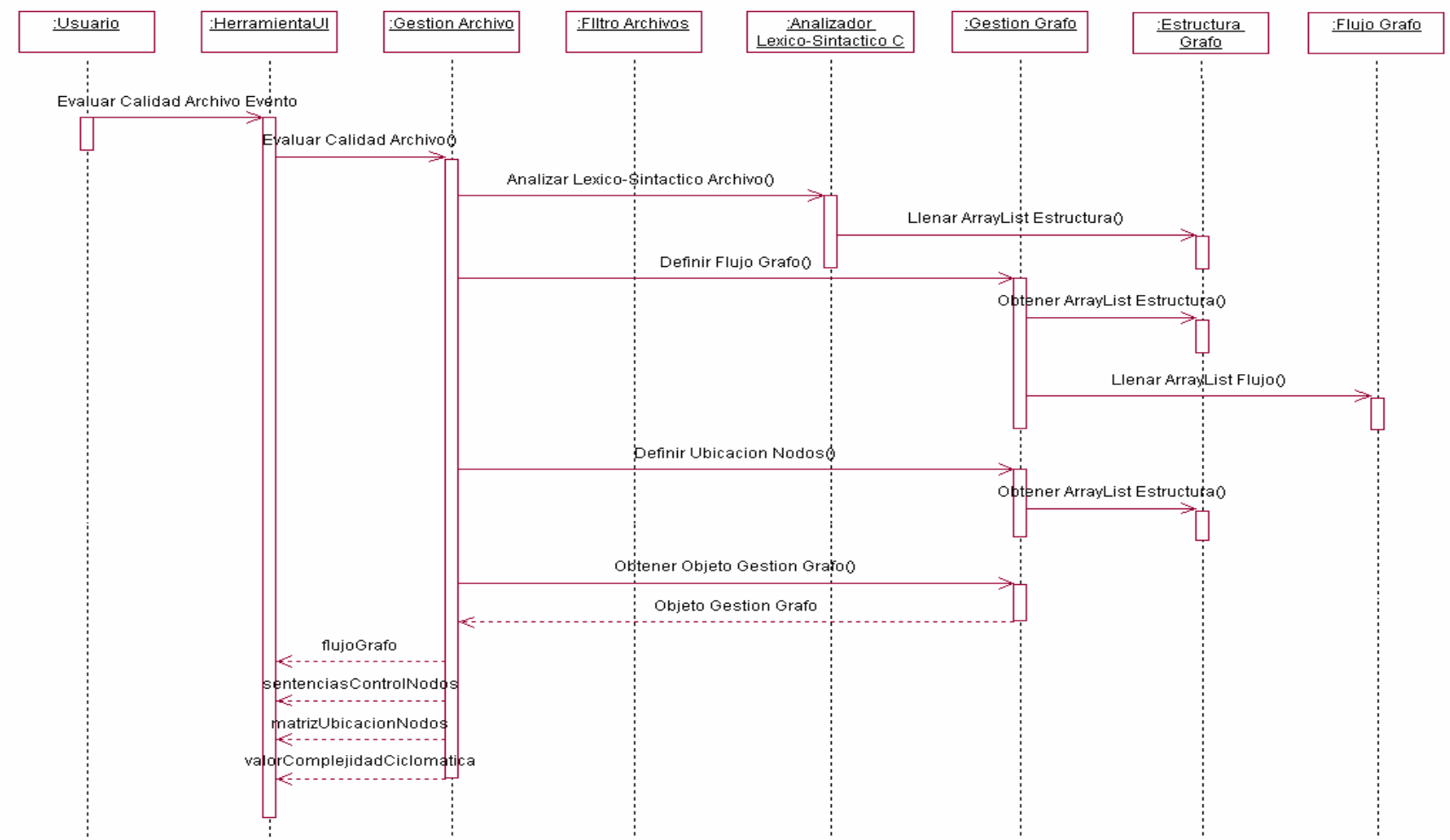
Figura 10. Participantes de Administrar Archivo

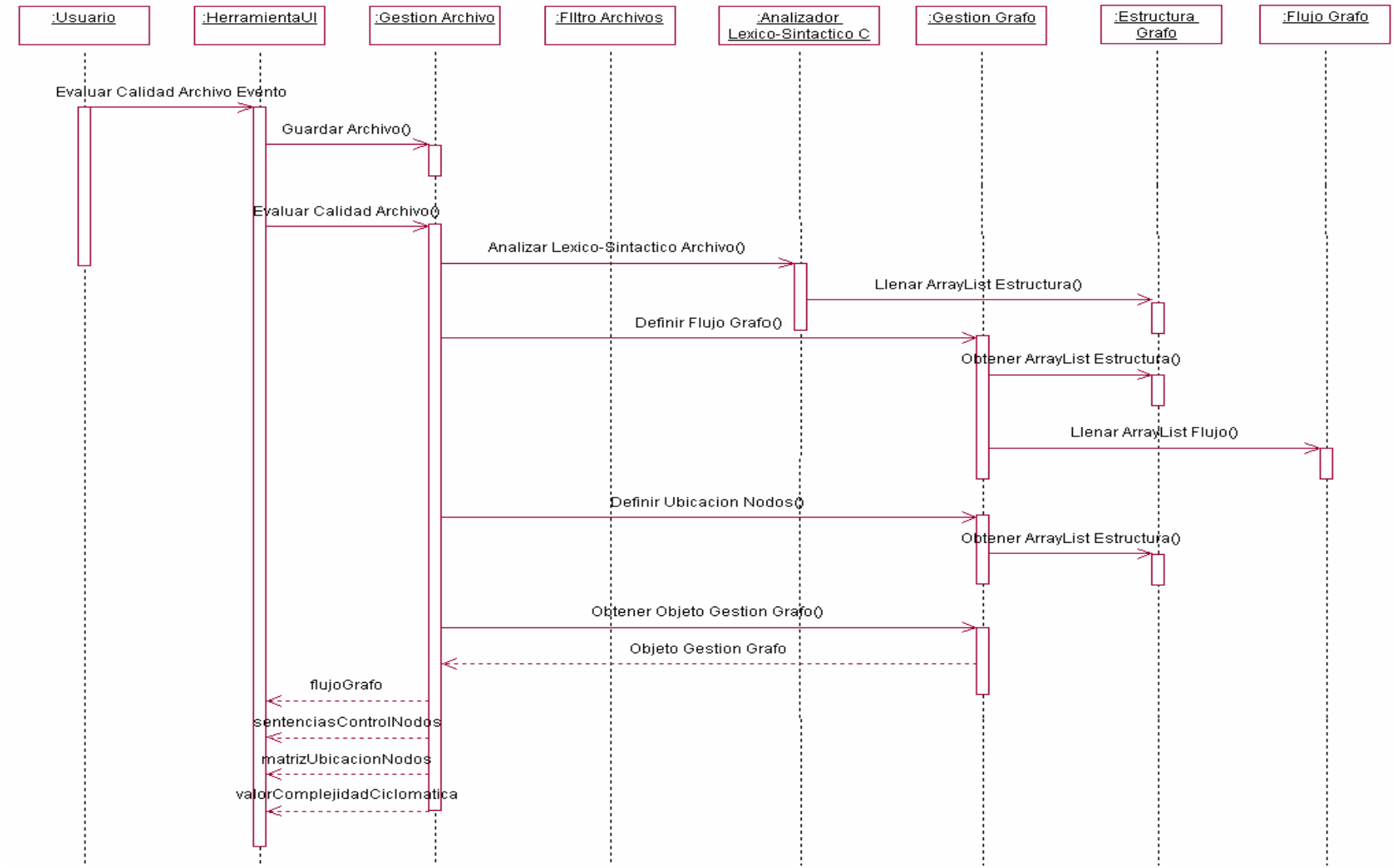
Fuente: La autora


:Usuario
:HerramientaUI
:Gestion Archivo
:Gestion Reporte
:FIltro Archivos
Nuevo Archivo Evento
Nuevo Archivo()
Abrir Archivo Evento
Abrir Archivo()
Filtrar Archivos()
codigoFuente
Abrir Archivo Evento
Guardar Archivo()
Abrir Archivo()
Filtrar Archivos()
codigoFuente
Abrir Archivo Evento
Guardar Como Archivo()
Filtrar Archivos()
Guardar Archivo()
Abrir Archivo()
Filtrar Archivos()
codigoFuente


:Usuario
:HerramientaUI
:Gestion Archivo
:Gestion Reporte
:FIltro Archivos
Abrir Archivo Evento
Guardar Reporte()
Abrir Archivo()
Filtrar Archivos()
codigoFuente
Abrir Archivo Evento
Guardar Como Reporte()
Filtrar Archivos()
Guardar Reporte()
Abrir Archivo()
Filtrar Archivos()
codigoFuente
Cerrar Archivo Evento
Cerrar Archivo()
Cerrar Archivo Evento
Guardar Archivo()
Cerrar Archivo()
Cerrar Archivo Evento
Guardar Como Archivo()
Filtrar Archivos()
Guardar Archivo()
Cerrar Archivo()


:Usuario
:HerramientaUI
:Gestion Archivo
:Gestion Reporte
:FIltro Archivos
Cerrar Archivo Evento
Guardar Reporte()
Cerrar Reporte()
Cerrar Archivo Evento
Guardar Como Reporte()
Filtrar Archivos()
Guardar Reporte()
Cerrar Archivo()
Guardar Archivo Envento
Guardar Archivo()
Guardar Archivo Envento
Guardar Como Archivo()
Filtrar Archivos()
Guardar Archivo()
Guardar Como Archivo Envento
Guardar Como Archivo()
Filtrar Archivos()
Guardar Archivo()


:Usuario
:HerramientaUI
:Gestion Archivo
:FIltro Archivos
:Analizador Lexico-Sintactico C
:Gestion Grafo
:Estructura Grafo
:Flujo Grafo
Evaluar Calidad Archivo Evento
Evaluar Calidad Archivo()
Analizar Lexico-Sintactico Archivo()
Llenar ArrayList Estructura()
Definir Flujo Grafo()
Obtener ArrayList Estructura()
Llenar ArrayList Flujo()
Definir Ubicacion Nodos()
Obtener ArrayList Estructura()
Obtener Objeto Gestion Grafo()
Objeto Gestion Grafo
flujoGrafo
sentenciasControlNodos
matrizUbicacionNodos
valorComplejidadCiclomatica


:Usuario
:HerramientaUI
:Gestion Archivo
:FIltro Archivos
:Analizador Lexico-Sintactico C
:Gestion Grafo
:Estructura Grafo
:Flujo Grafo
Evaluar Calidad Archivo Evento
Guardar Archivo()
Evaluar Calidad Archivo()
Analizar Lexico-Sintactico Archivo()
Llenar ArrayList Estructura()
Definir Flujo Grafo()
Obtener ArrayList Estructura()
Llenar ArrayList Flujo()
Definir Ubicacion Nodos()
Obtener ArrayList Estructura()
Obtener Objeto Gestion Grafo()
Objeto Gestion Grafo
flujoGrafo
sentenciasControlNodos
matrizUbicacionNodos
valorComplejidadCiclomatica

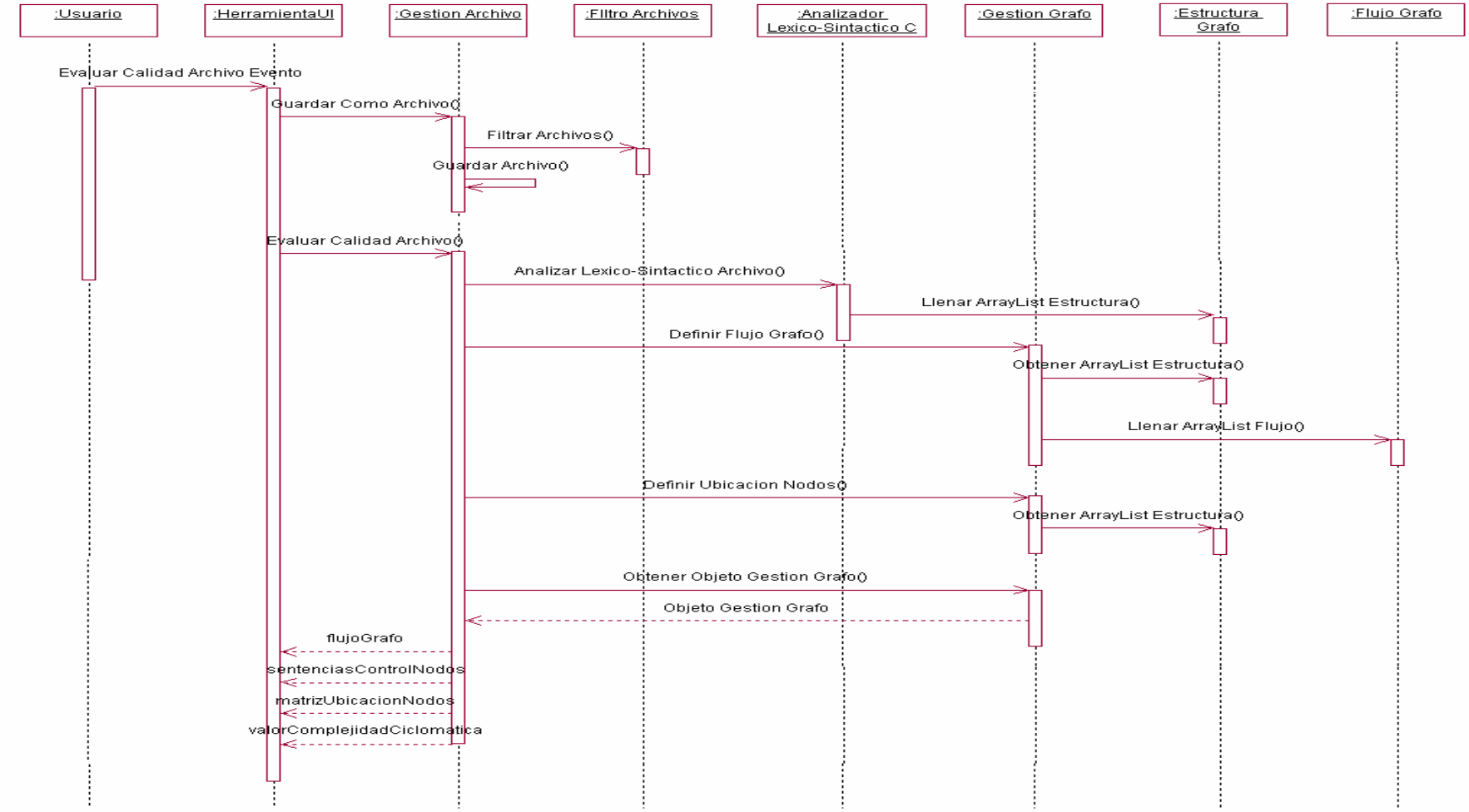


Figura 11. Flujo de eventos de Administrar Archivo

Fuente: La autora

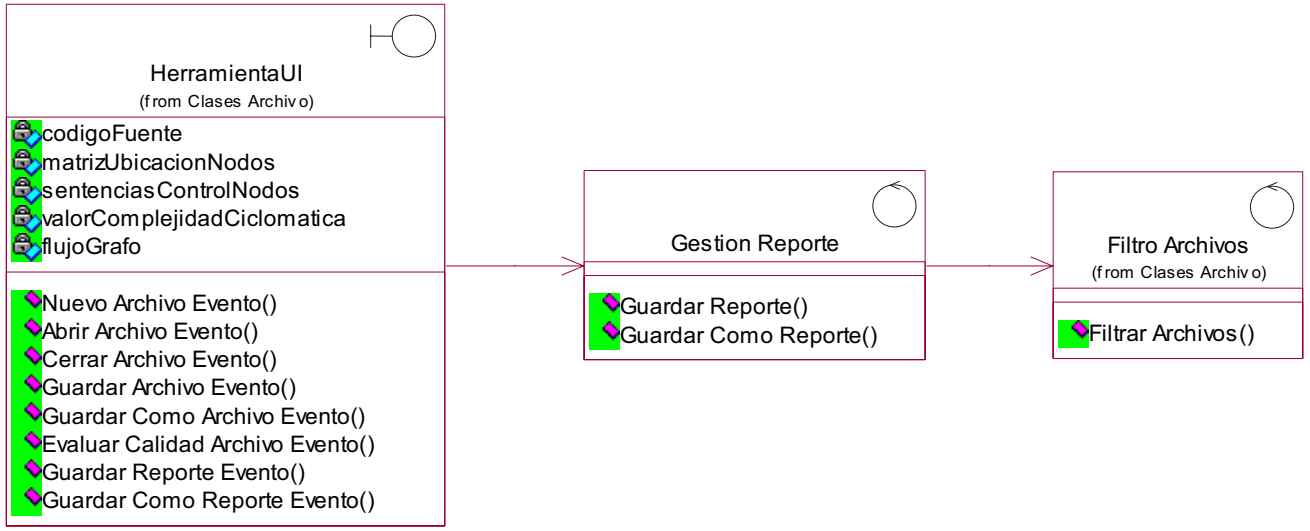


Figura 12. Participantes de Administrar Reporte

Fuente: La autora

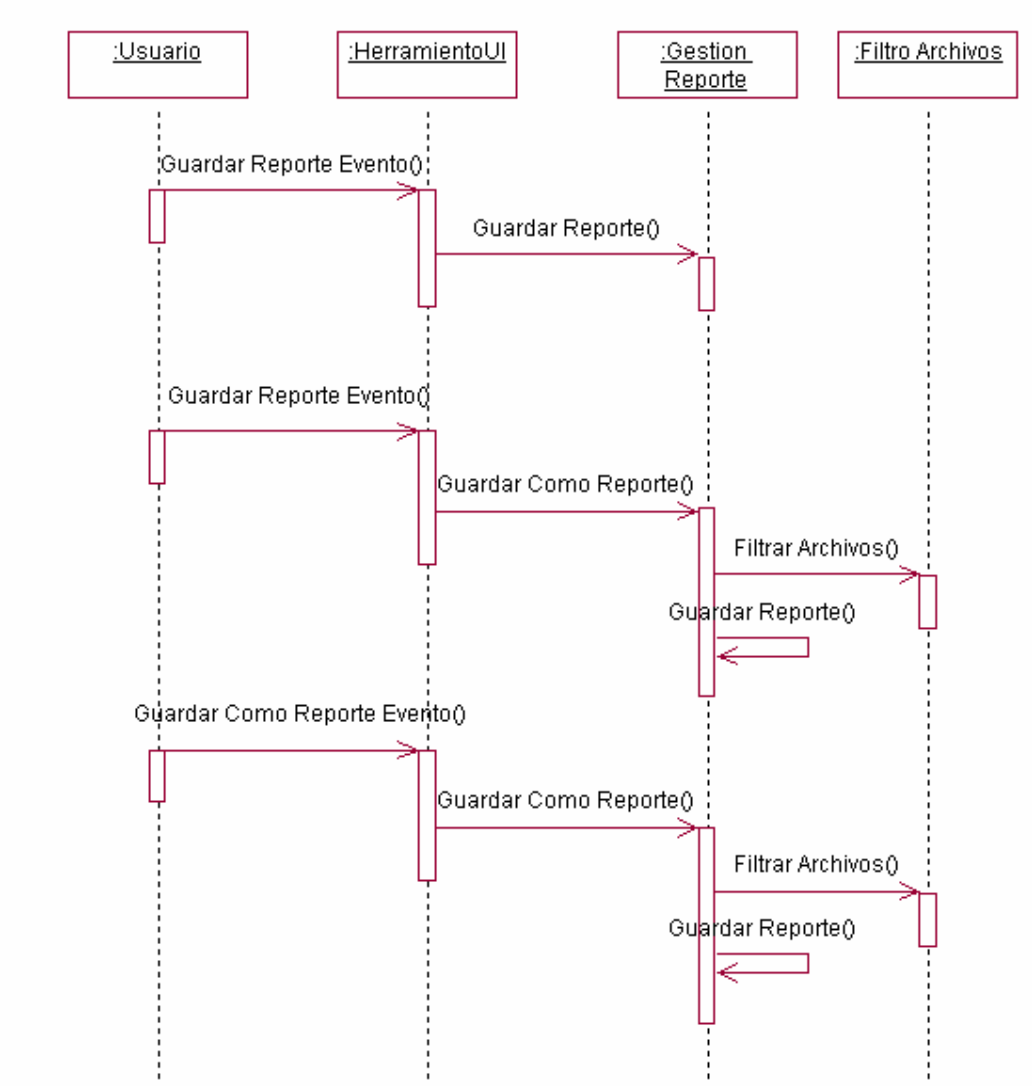


Figura 13. Flujo de eventos de Administrar Reporte

Fuente: La autora

ANEXO C. PRUEBAS DEL SISTEMA

PRUEBAS DEL SISTEMA

La herramienta utilizada para realizar estas pruebas y obtener el tiempo que se demora en evaluar la calidad del código fuente (C ANSI) fue JUnit que viene integrada en NetBeans (5.5).

HARDWARE

- Pentium IV 2.40 GHz
- 512 MB de memoria RAM

SOFTWARE

- Windows XP Home

ABREVIACIONES

CC = Complejidad Ciclomática

LDC = Líneas de código

PRUEBA DE RESISTENCIA

Las pruebas de resistencia están diseñadas para enfrentar a los programas con situaciones anormales.

Datos de Entrada

Nombre del Archivo	Líneas de Código	Complejidad Ciclomática
Brick Game.c	420	57

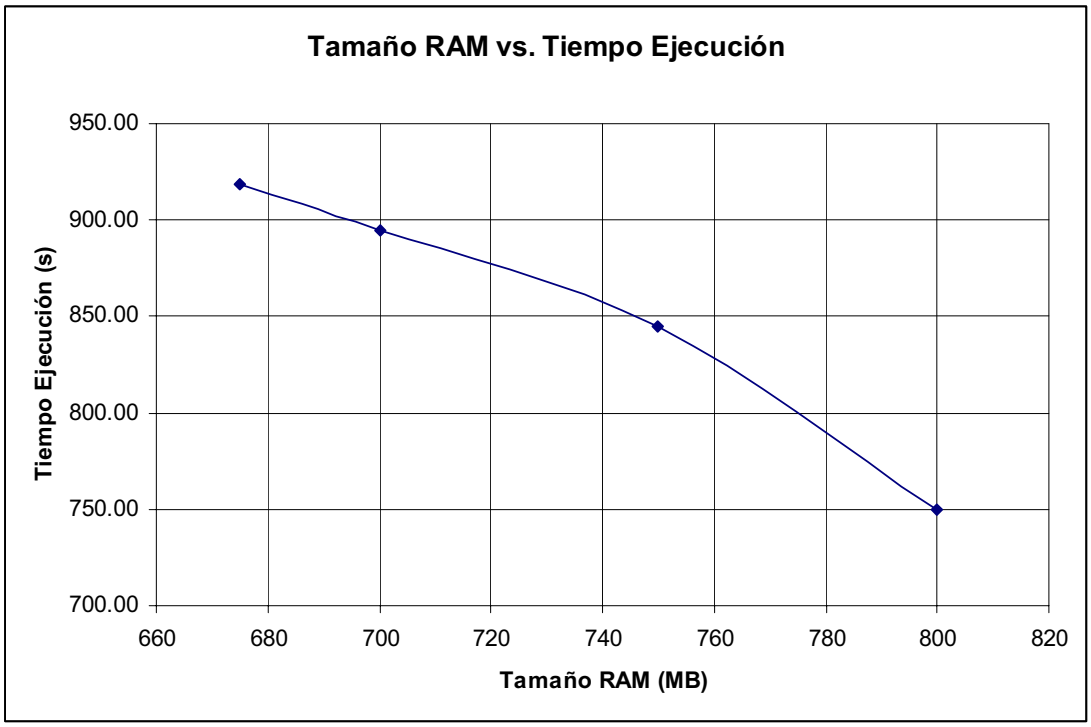
Análisis de resultados

Tiempo de Evaluación (seg)	Tamaño memoria RAM (MB)
749,427	800
844.965	750
894.756	700
918.536	675

Falló	650
Falló	600

Al realizar este caso de prueba con 512 MB de RAM la herramienta EVAC colapsó la memoria; configuramos el Run de la herramienta NetBeans para simular diferentes tamaños de memoria para detectar hasta que punto la herramienta funciona.

De la tabla de resultados podemos observar que para este caso de prueba la herramienta funciona hasta con 675 MB de memoria, menos de este tamaño la herramienta colapsa.



En el gráfico se aprecia que mientras aumenta el tamaño de memoria RAM el tiempo de ejecución disminuye.

Específicamente EVAC deja de funcionar cuando trata de asignar espacio en el Buffer de la memoria para dibujar el grafo de flujo.

PRUEBA DE RENDIMIENTO

La prueba de rendimiento está diseñada para probar el rendimiento del software

en tiempo de ejecución dentro del contexto de un sistema integrado.

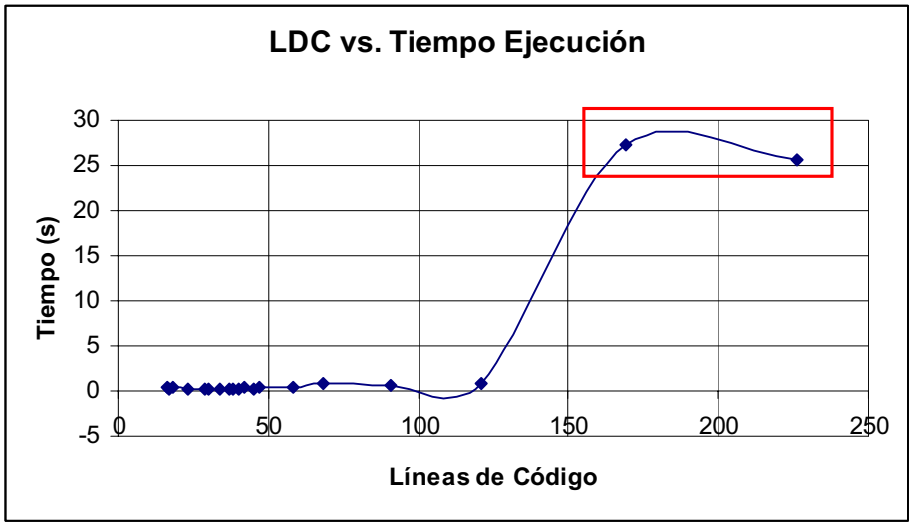
Datos de Entrada

Nombre del Archivo	Líneas de Código
recursive_power_function.c	16
program to reverse the input string.c	17
castle building.c	18
palindrome verification.c	23
program for finding the prime numbers.c	29
program to calculate the pascal triangle.c	30
program to calculate the sum of series.c	34
calculadora.c	37
car secletcing using switch.c	38
initialize the function pointer array.c	40
calculator.c	42
implementing arithmetic and array functions.c	45
tree manipulation and traversing.c	47
display a date.c	58
week calculation.c	68
buscador_de _caracteres.c	91
calculadora_mejorada.c	121
tower of hanoi.c	169
game.c	226

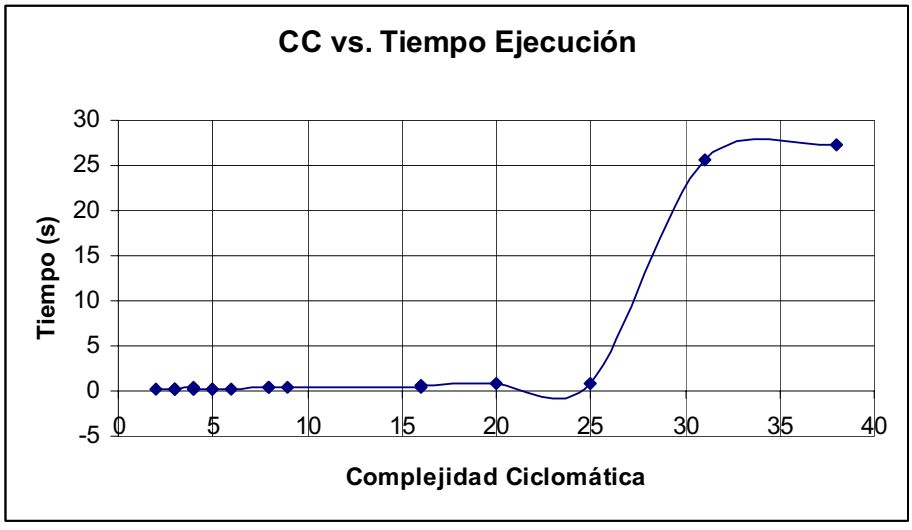
Análisis de resultados

Nombre del Archivo	Líneas de Código	Complejidad Ciclomática	Tiempo de Evaluación
recursive_power_function.C	16	4	0,381
Program to reverse the input string.c	17	2	0,21
Castle Building.c	18	8	0,32
Palindrome Verification.c	23	3	0,221
Program for finding the prime numbers.c	29	5	0,25
Program to Calculate the Pascal triangle.c	30	3	0,261
Program to calculate the sum of series.c	34	4	0,23
calculadora.c	37	6	0,25
car secletcing using switch.c	38	4	0,22
Initialize the function pointer array.c	40	3	0,241
calculator.c	42	8	0,33
Implementing arithmetic and array functions.c	45	5	0,26
Tree manipulation and traversing.c	47	9	0,36
Display a date.c	58	16	0,36
Week Calculation.c	68	20	0,861
BUSCADOR_DE _CARACTERES.C	91	16	0,681

CALCULADORA_MEJORADA.C	121	25	0,821
Tower of Hanoi.c	169	38	27,249
game.c	226	31	25,697

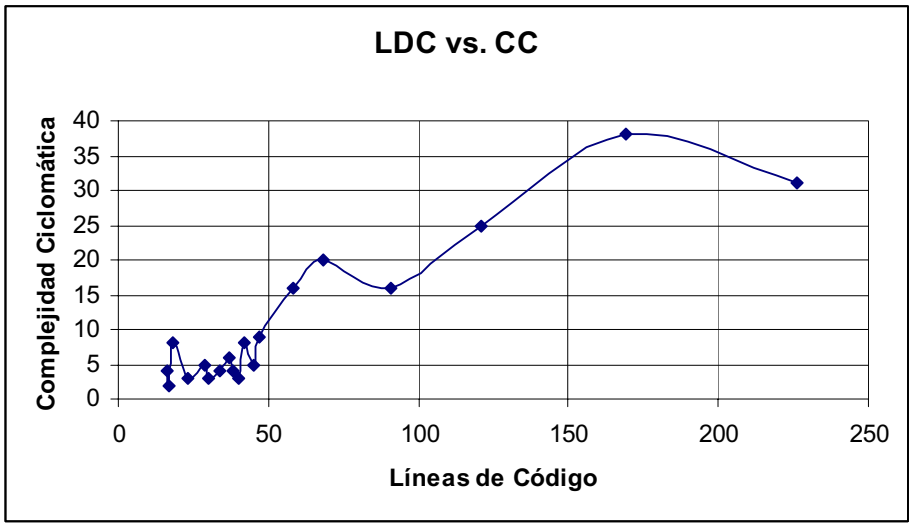


En el gráfico podemos observar que existe una tendencia a incrementar el uso de recursos de hardware (especialmente memoria) cuando el número de líneas de código aumenta, pero se pueden dar excepciones como la que se presenta en el recuadro rojo.



En este gráfico se observa que el número de complejidad ciclomática (CC) obtenido está directamente relacionado al tiempo de ejecución del proceso de

evaluación, es decir, entre mayor sea el número de CC mayor será el tiempo que se utilicen los recursos de hardware.



En este último gráfico se observa que el número de líneas de código es totalmente independiente del valor de la complejidad ciclomática, es decir, un programa puede tener 70 LDC con una CC de 20 y un programa de 90 LDC puede tener una CC de 15.

De las pruebas de resistencia y rendimiento podemos concluir que la herramienta funcionó con las siguientes especificaciones de hardware y software:

- Pentium IV 2.40 GHz
- 675 MB de memoria RAM
- Windows XP Home
- Archivo con 420 líneas de código y una complejidad ciclomática de 57.

ANEXO D. PLANIFICACION DE RECURSOS

Para realizar la evaluación se utiliza tres tipos de recursos: Humano, Técnico y Administrativo, cada uno representados en una tabla.

Para los recursos humanos se indica el rol, cantidad, su costo, tiempo y total. Para el caso de los recursos técnicos (hardware, software, comunicaciones) se tendrá la columna ítem, cantidad, descripción, costo total, tiempo de vida, el índice de prorrateo, y su valor luego del prorrateo. Se realizará una tabla resumen de los tres recursos técnicos. Para el caso de los recursos administrativos se tendrán las mismas columnas que en los recursos técnicos, de igual forma se realizará una tabla resumen de cada uno de los recursos.

Finalmente se tendrá una tabla resumen en la que se indica los valores de todos los recursos utilizados y el costo final del proyecto.

RECURSOS HUMANOS

ROL	CANTIDAD	COSTO/MES	TIEMPO TRABAJO (meses)	TOTAL(cantidad*costo/mes*tiempo de trabajo)
Evaluador	1	800	1 (100%)	800
TOTAL				**800 USD**

RECURSOS TECNICOS

HARDWARE Tiempo de la Evaluación: 1 mes

ITEM	CANTIDAD	DESCRIPCION	COSTO UNITARIO	TIEMPO VIDA (meses)	PRORRATEO (meses de proyecto / tiempo vida)	VALOR (cantidad*costo unitario *prorrateo)
Estación	1	Pentium IV 1.5GH 256 RAM 80 GB (HD)	400	24	0.04	16
Impresora (Inyección)	1	HP Injet	150	24	0.04	6
TOTAL						**22 USD**

SOFTWARE Tiempo de la Evaluación: 1 mes

ITEM	CANTIDAD	DESCRIPCION	COSTO UNITARIO	TIEMPO VIDA (meses)	PRORRATEO (meses de proyecto / tiempo vida)	VALOR (cantidad*costo unitario *prorrateo)
Sistema Operativo estación	1	Windows XP Professional	70	24	0.04	2.8
Office	2	Microsoft Office 2003	170	24	0.04	6.8
TOTAL						**9.6 USD**

RESUMEN RECURSOS TÉCNICOS

RECURSO	VALOR
Hardware	**22**
Software	**9.6**
TOTAL	**31.6 USD**

RECURSOS ADMINISTRATIVOS

SERVICIOS Tiempo de la Evaluación: 1 mes

ITEM	DESCRIPCION	COSTO/MES	COSTO (costo/mes*tiempo de proyecto)
Luz	Servicio Básico	10	10
Agua	Servicio Básico	12	12
Teléfono	Servicio Básico	35	35
Celular	Plan Cerrado	25	25
Internet	Ilimitado	35	35
TOTAL			**117 USD**

MOBILIARIO Tiempo de la Evaluación: 1 mes

ITEM	CANTIDAD	DESCRIPCION	COSTO UNITARIO	TIEMPO VIDA (meses)	PRORRATEO (meses de proyecto / tiempo vida)	VALOR (cantidad*costo unitario *prorrateo)
Mesa PC's	1	Mesa de madera	50	60	0.01	0.5
Sillas	1	Metálicas Girable	25	60	0.01	0.25
Cafetera	1	Oster (3 litros)	100	60	0.01	1
TOTAL						**1.75 USD**

SUMINISTROS **Tiempo de la Evaluación: 1 mes**

ITEM	CANTIDAD	DESCRIPCION	COSTO UNITARIO	VALOR (cantidad*costo unitario)
Resma Papel	1	Resma Láser A4	15	15
CD-R	2	Imation	0.7	1.4
Cartuchos	1	HP	35	35
Lápices	2	Staddler	0.25	0.5
Marcador	1	Staddler	0.80	0.8
Borradores	2	Rotring	0.15	0.3
TOTAL				**53 USD**

ASEO **Tiempo de la Evaluación: 1 mes**

ITEM	CANTIDAD	DESCRIPCION	COSTO UNITARIO	VALOR (cantidad*costo unitario)
Escoba	1	-	2	2
Desinfectante	1	-	1.5	1.5
Franela	1	-	0.5	0.5
Jabón Platos	1	-	0.75	0.75
Jabón Manos	1	-	0.5	0.5
TOTAL				**5.25 USD**

CAFETERIA **Tiempo de la Evaluación: 1 mes**

ITEM	CANTIDAD	DESCRIPCION	COSTO UNITARIO	VALOR (cantidad*costo unitario)
Vasos	1	Plásticos X 100	0.7	0.7
Agua	2	Botellón	0.90	1.80
Café	1	Nestcafé	0.75	0.75
Azúcar	1	San Carlos	0.90	0.90
Té	2	Varios sabores	0.50	1.00
Cucharas	1	Plásticas x 100	0.60	0.60
Servilletas	1	Paquetes x 100	0.50	0.50
Galletas	2	Ducales	0.75	1.50
TOTAL				**7.75 USD**

RESUMEN RECURSOS ADMINISTRATIVOS

RECURSO	VALOR
Oficina	**80**
Servicios	**117**
Mobiliario	**1.75**
Suministros	**53**
Aseo	**5.25**
Cafetería	**7.75**
TOTAL	**264.75 USD**

COSTO TOTAL DEL PROYECTO

RECURSO	VALOR
HUMANO	**800**
TECNICO	**31.6**
ADMINISTRATVO	**264.75**
TOTAL	**1096.35**
3 % IMPREVISTOS	**32.89**
15 % UTILIDAD	**164.45**
COSTO TOTAL	**1293.69 USD**

ANEXO E. MEDIDAS DEFINIDAS EN BASE AL CÓDIGO FUENTE

FUNCION: ADMINISTRAR ARCHIVO

CARACTERÍSTICA: FUNCIONALIDAD

Subcaracterística: cumplimiento de la funcionalidad
Métrica: cumplimiento de la funcionalidad

Siendo A: funciones implementadas, B: funciones especificadas en los requrimientos y X: A/B se lo determina en la Tabla 3.4.

El código fuente (clases implementadas) que se utilizó para realizar los cálculos de las diferentes métricas seleccionadas se presentan a continuación:

DISEÑO

En el documento se especifica que la función Administrar Archivo posee seis operaciones: nuevo, abrir, cerrar, guardar, guardar como, evaluar calidad.

Funciones implementadas en el código fuente

```
public void newFile(final int x,final int y)
public String openFile()
public int exitFile(String codeSourceB, String codeSourceE, String pathCloseF, String nameCloseF)
public boolean saveFile(String nameSaveF, String pathSaveF, String codeSourceES)
public boolean saveAsFile(String pathSaveAsF, String codeSourceSave)
public String evaluateQuality(String path)
```

CODIGO FUENTE

Existen clases tanto para la capa de presentación y la de negocio pero no para la capa de datos porque no es un sistema que almene información.

Clases implementadas

```
public class IEvac  //Interface
public class ManagerFile //Negocio
```

RELACIÓN

Se refiere al cumplimiento del diseño y del código fuente con respecto al cumplimiento de la funcionalidad

	DISEÑO	CODIGO FUENTE	RELACION	TOTAL
B	6	2	2	10
A	6	2	2	10

A	B	X
10	10	1,00

CARACTERISTICA: FIABILIDAD

Subcaracterística: tolerancia a fallas

Métrica: anulación de operaciones incorrectas

Siendo A: funciones implementadas, B: funciones especificadas en los requisitos y X: A/B se lo determina en la Tabla 3.4.

El código fuente (funciones implementadas) que se utilizó para realizar el cálculo de las diferentes métricas seleccionadas, se presenta a continuación:

NUEVO

```
public void newFile(final int x,final int y)
```

ABRIR

```
public String openFile()
//Si no existe el archivo .C que se desea abrir
if (extFile.equals(".C")||extFile.equals(".c")) {
if (!selected.exists()) {
                JOptionPane.showMessageDialog(frameEvac,
                    "File " +nameCurrentFile+" does not exist",
                    "Information",
                    JOptionPane.ERROR_MESSAGE); }
else {
//Si desea abrir un archivo que no sea  .C
            JOptionPane.showMessageDialog(frameEvac,
                "Only allows to open files written in C ",
```

```
                "Information",
                JOptionPane.ERROR_MESSAGE);
        }
```

CERRAR

```
public int exitFile(String codeSourceB, String codeSourceE, String pathCloseF, String nameCloseF)
```

GUARDAR

```
public boolean saveFile(String nameSaveF, String pathSaveF, String codeSourceES)
try {…
    }
catch (IOException e) {
    JOptionPane.showMessageDialog(frameEvac,"It did not save the changes of the file "+nameSaveF,
            "Error",JOptionPane.ERROR_MESSAGE);}
```

GUARDAR COMO

```
public boolean saveAsFile(String pathSaveAsF, String codeSourceSave)
try {…
    }
catch (SecurityException e) {
        JOptionPane.showMessageDialog(frameEvac,"Error of access to the directory "+currentLocation,
            "Error",JOptionPane.ERROR_MESSAGE);}
```

EVALUAR CALIDAD

Parser

```
public int evaluateQualityCC(String path, GraphStructure graphStructure)
this.codErrParser= parser.evaluateQualityCC(path,graphStructure);
```

Registra Flujo

```
public int markPath(GraphStructure graphStructure,ArrayList listNode, int node) //No Cumple//
managerGraph.markPath(graphStructure,listNodes,1);
```

Registra Diagrama

```
public int[] fillMatrix(GraphStructure graphStructure,int[][] matrix, int node, int x, int y2, int m, boolean
incremental, int space) //No Cumple//
managerGraph.fillMatrix(graphStructure,matrix,0,0,0,lenthMatrix*3,true,5);
```

	NUEVO	ABRIR	CERRAR	GUARDAR	GUARDAR COMO	EVALUAR CALIDAD	TOTAL
B	1	1	1	1	1	3	8
A	1	1	1	1	1	1	6

A	B	X
6	8	0,75

CARACTERISTICA: USABILIDAD

Subcaracterística: capacidad para ser entendido

Métrica: funciones evidentes

Siendo A: funciones implementadas, B: funciones especificadas en los requisitos y X: A/B se lo determina en la Tabla 3.4.

El código fuente (clases implementadas) que se utilizó para realizar los cálculos de las diferentes métricas seleccionadas se presentan a continuación:

```
public void newFile(final int x,final int y)
public String openFile()
public int exitFile(String codeSourceB, String codeSourceE, String pathCloseF, String nameCloseF)
public boolean saveFile(String nameSaveF, String pathSaveF, String codeSourceES)
public boolean saveAsFile(String pathSaveAsF, String codeSourceSave)
public String evaluateQuality(String path)
```

	NUEVO	ABRIR	CERRAR	GUARDAR	GUARDAR COMO	EVALUAR CALIDAD	TOTAL
B	1	1	1	1	1	1	6
A	1	1	1	1	1	1	6

A	B	X
6	6	1,00

Subcaracterística: operabilidad

Métrica: claridad del mensaje

Siendo A: funciones implementadas, B: funciones especificadas en los requisitos y X: A/B se lo determina en la Tabla 3.4.

El código fuente (clases implementadas) que se utilizó para realizar los cálculos de las diferentes métricas seleccionadas se presentan a continuación:

```
public String openFile()
//No existe el archivo
JOptionPane.showMessageDialog(frameEvac,
```

```
                    "File " +nameCurrentFile+" does not exist",
                    "Information",
                    JOptionPane.ERROR_MESSAGE);
//No se encuentra el archivo
JOptionPane.showMessageDialog(frameEvac,"File not found: " +
ex.getMessage(),"Error",JOptionPane.ERROR_MESSAGE);
//Solo se permite abrir archivos escritos en C ANSI
JOptionPane.showMessageDialog(frameEvac,
                "Only allows to open files written in C ",
                "Information",
                JOptionPane.ERROR_MESSAGE);

public int exitFile(String codeSourceB, String codeSourceE, String pathCloseF, String nameCloseF)
//Desea almacenar los cambios
JOptionPane.showConfirmDialog(frameEvac,
                "Do you want to save the changes of "+ nameCloseF,
                "EVAC",
                JOptionPane.YES_NO_CANCEL_OPTION,
                JOptionPane.QUESTION_MESSAGE);

public boolean saveFile(String nameSaveF, String pathSaveF, String codeSourceES)
//No se puedo almacenar los cambios en el archivo
JOptionPane.showMessageDialog(frameEvac,"It did not save the changes of the file "+nameSaveF,
                "Error",JOptionPane.ERROR_MESSAGE);

public boolean saveAsFile(String pathSaveAsF, String codeSourceSave)
//Reemplazar por el archivo existente
JOptionPane.showConfirmDialog(frameEvac,"Do you want to replace the file exists "+ nameF,
                    "EVAC", JOptionPane.YES_NO_OPTION,
                    JOptionPane.WARNING_MESSAGE);
//Error al acceder al directorio
JOptionPane.showMessageDialog(frameEvac,"Error of access to the directory "+currentLocation,
                "Error",JOptionPane.ERROR_MESSAGE);
```

A	B	X
7	7	1,00

Subcaracterística: operabilidad

Métrica: recuperabilidad del error operacional

Siendo A: excepciones implementadas, B: excepciones especificadas en los requisitos y X: A/B se lo determina en la Tabla 3.4.

El código fuente (clases implementadas) que se utilizó para realizar los cálculos de las diferentes métricas seleccionadas se presentan a continuación:

```
public String openFile()
{ .....   catch(Exception ex)  ……..}
public boolean saveFile(String nameSaveF, String pathSaveF, String codeSourceES)
{ .....   catch(Exception ex)  ……..}
public boolean saveAsFile(String pathSaveAsF, String codeSourceSave)
{ .....   catch(SecurityException e)  ……..}
```

A	B	X
3	3	1,00

CARACTERISTICA: PORTABILIDAD

Subcaracterística: Adaptabilidad

Métrica: Adaptabilidad De La Estructura De Datos

Siendo A: estructura de datos que es operable luego de la adaptación, B: estructuras de datos especificadas y X: A/B se lo determina en la Tabla 3.4.

El código fuente (clases implementadas) que se utilizó para realizar los cálculos de las diferentes métricas seleccionadas se presentan a continuación:

Librería de Clase IEvac (PRESENTACIÓN)

```
public class IEvac
```

Librería de Clase ManagerFile (NEGOCIO)

```
public class ManagerFile
```

A	B	X
1	1	1,00

FUNCION: ADMINISTRAR REPORTE

CARACTERÍSTICA: FUNCIONALIDAD

Subcaracterística: cumplimiento de la funcionalidad
Métrica: cumplimiento de la funcionalidad

Siendo A: funciones implementadas, B: funciones especificadas en los requrimientos y X: A/B se lo determina en la Tabla 3.4.

El código fuente (clases implementadas) que se utilizó para realizar los cálculos de las diferentes métricas seleccionadas se presentan a continuación:

DISEÑO

En el documento se especifica que la función Administrar Reporte posee dos operaciones: guardar, guardar como.

Funciones implementadas en el código fuente

```
public boolean saveReport(String nameReport,String pathReport, String informationR, BufferedImage bufferedImage)
public boolean saveAsReport(String pathReport, String informationR,   BufferedImage bufferedImage)
```

CODIGO FUENTE

Existen clases tanto para la capa de presentación y la de negocio pero no para la capa de datos porque no es un sistema que almene información.

Clases implementadas

```
public class IEvac
public class ManagerReport
```

RELACIÓN

Se refiere al cumplimiento del diseño y del código fuente con respecto al cumplimiento de la funcionalidad

	DISEÑO	CODIGO FUENTE	RELACION	TOTAL
B	2	2	2	6
A	2	2	2	6

A	B	X
6	6	1,00

CARACTERISTICA: FIABILIDAD

Subcaracterística: tolerancia a fallas

Métrica: anulación de operaciones incorrectas

Siendo A: funciones implementadas, B: funciones especificadas en los requisitos y X: A/B se lo determina en la Tabla 3.4.

El código fuente (clases implementadas) que se utilizó para realizar los cálculos de las diferentes métricas seleccionadas se presentan a continuación:

GUARDAR

```
public boolean saveReport(String nameReport,String pathReport, String informationR, BufferedImage
bufferedImage)
if (pathReport==null){
        return saveAsReport(nameReport,informationR,bufferedImage);
    } else{}
```

GUARDAR COMO

```
public boolean saveAsReport(String pathReport, String informationR,   BufferedImage bufferedImage)
if (fileS.exists()){…}
else{…}
```

	GUARDAR	GUARDAR COMO	TOTAL
B	1	1	2
A	1	1	2

A	B	X
2	2	1,00

CARACTERISTICA: USABILIDAD

Subcaracterística: capacidad para ser entendido

Métrica: funciones evidentes

Siendo A: funciones implementadas, B: funciones especificadas en los requisitos y X: A/B se lo determina en la Tabla 3.4.

El código fuente (clases implementadas) que se utilizó para realizar los cálculos de las diferentes métricas seleccionadas se presentan a continuación:

```
public boolean saveReport(String nameReport,String pathReport, String informationR, BufferedImage
bufferedImage)
public boolean saveAsReport(String pathReport, String informationR,   BufferedImage bufferedImage)
```

	GUARDAR	GUARDAR COMO	TOTAL
B	1	1	2
A	1	1	2

A	B	X
2	2	1,00

Subcaracterística: operabilidad

Métrica: claridad del mensaje

Siendo A: funciones implementadas, B: funciones especificadas en los requisitos y X: A/B se lo determina en la Tabla 3.4.

El código fuente (clases implementadas) que se utilizó para realizar los cálculos de las diferentes métricas seleccionadas se presentan a continuación:

```
public boolean saveReport(String nameReport,String pathReport, String informationR, BufferedImage
bufferedImage)
//Error al tartar de generar el archivo .PDF en el directorio
JOptionPane.showMessageDialog(null,"Error to generate pdf in the directory "+pathReport+".-
"+e.getMessage(),
```

```
                "Error",JOptionPane.ERROR_MESSAGE);
//Error al acceder al directorio
JOptionPane.showMessageDialog(null,"Error of access to the directory "+currentLocation,
                "Error",JOptionPane.ERROR_MESSAGE);
//Desea reemplazar el archivo existente
JOptionPane.showConfirmDialog(null,"Do you want to replace the file exists "+nameR,
                  "EVAC", JOptionPane.YES_NO_OPTION,
                  JOptionPane.WARNING_MESSAGE);
```

A	B	X
3	3	1,00

Subcaracterística: operabilidad

Métrica: recuperabilidad del error operacional

Siendo A: excepciones implementadas, B: excepciones especificadas en los requisitos y X: A/B se lo determina en la Tabla 3.4.

El código fuente (clases implementadas) que se utilizó para realizar los cálculos de las diferentes métricas seleccionadas se presentan a continuación:

```
public boolean saveReport(String nameReport,String pathReport, String informationR, BufferedImage
bufferedImage)
{ …….   catch(Exception ex)  ……..}
public boolean saveAsReport(String pathReport, String informationR,   BufferedImage bufferedImage)
{ ……. catch (SecurityException e) ……..}
```

A	B	X
2	2	1,00

CARACTERISTICA: PORTABILIDAD

Subcaracterística: Adaptabilidad

Métrica: Adaptabilidad De La Estructura De Datos

Siendo A: estructura de datos que es operable luego de la adaptación, B: estructuras de datos especificadas y X: A/B se lo determina en la Tabla 3.3.

El código fuente (clases implementadas) que se utilizó para realizar los cálculos de las diferentes métricas seleccionadas se presentan a continuación:

Librería de Clase IEvac (PRESENTACIÓN)

```
public class IEvac
```

Librería de Clase ManagerReport (NEGOCIO)

```
public class ManagerReport
```

A	B	X
1	1	1,00

ANEXO F. REGISTRO DE EVALUACIÓN (TOMA DE MEDIDAS)

Nombre del Producto: EVAC
Módulo del Sistema a Evaluar: Administrar Archivo, Administrar Reporte

FUNCION: ADMINISTRAR ARCHIVO

CARACTERISTICA: FUNCIONALIDAD

SUBCARACTERISTICA: Cumplimiento de la Funcionalidad

METRICA: Cumplimiento de la Funcionalidad

	DISEÑO	CODIGO FUENTE	RELACION	TOTAL
B	6	2	2	10
A	6	2	2	10

A	B	X	CRITERIO
10	10	1,00	ACEPTABLE

CARACTERISTICA: FIABILIDAD

SUBCARACTERISTICA: Tolerancia a fallas

METRICA: Anulación de operación incorrecta

	NUEVO	ABRIR	CERRAR	GUARDAR	GUARDAR COMO	EVALUAR CALIDAD	TOTAL
B	1	1	1	1	1	3	8
A	1	1	1	1	1	1	6

A	B	X	CRITERIO
6	8	0,75	ACEPTABLE

CARACTERISTICA: USABILIDAD

SUBCARACTERISTICA: Capacidad para ser entendido

METRICA: Funciones Evidentes

	NUEVO	ABRIR	CERRAR	GUARDAR	GUARDAR COMO	EVALUAR CALIDAD	TOTAL
B	1	1	1	1	1	1	6
A	1	1	1	1	1	1	6

A	B	X	CRITERIO
6	6	1,00	ACEPTABLE

SUBCARACTERISTICA: Operabilidad

METRICA: Claridad del mensaje

A	B	X	CRITERIO
7	7	1,00	ACEPTABLE

METRICA: Recuperabilidad de error operacional (Excepciones)

A	B	X	CRITERIO
3	3	1,00	ACEPTABLE

CARACTERISTICA: PORTABILIDAD

SUBCARACTERISTICA: Adaptabilidad

METRICA: Adaptabilidad de la estructura de datos

A	B	X	CRITERIO
2	2	1.00	ACEPTABLE

FUNCION: ADMINISTRAR REPORTE

CARACTERISTICA: FUNCIONALIDAD

SUBCARACTERISTICA: Cumplimiento de la Funcionalidad

METRICA: Cumplimiento de la Funcionalidad

	DISEÑO	CODIGO FUENTE	RELACION	TOTAL
B	2	2	2	6
A	2	2	2	6

A	B	X	CRITERIO
6	6	1,00	ACEPTABLE

CARACTERISTICA: FIABILIDAD

SUBCARACTERISTICA: Tolerancia a fallas

METRICA: Anulación de operación incorrecta

	GUARDAR	GUARDAR COMO	TOTAL
B	1	1	2
A	1	1	2

A	B	X	CRITERIO
2	2	1,00	ACEPTABLE

CARACTERISTICA: USABILIDAD

SUBCARACTERISTICA: Capacidad para ser entendido

METRICA: Funciones Evidentes

	GUARDAR	GUARDAR COMO	TOTAL
B	1	1	2
A	1	1	2

A	B	X	CRITERIO
2	2	1,00	ACEPTABLE

SUBCARACTERISTICA: Operabilidad

METRICA: Claridad del mensaje

A	B	X	CRITERIO
3	3	1,00	ACEPTABLE

METRICA: Recuperabilidad de error operacional (Excepciones)

A	B	X	CRITERIO
2	2	1,00	ACEPTABLE

CARACTERISTICA: PORTABILIDAD

SUBCARACTERISTICA: Adaptabilidad

METRICA: Adaptabilidad de la estructura de datos

A	B	X	CRITERIO
2	2	1.00	ACEPTABLE

ANEXO G. MÓDULO DE EVALUACIÓN

a) Prólogo e Introducción

Prólogo

La aplicación a ser evaluada es EVAC, la cual permite evaluar la calidad del código fuente generado en C ANSI en base a la utilización de la métrica de complejidad ciclomática.

Introducción

El propósito de realizar el módulo de evaluación es el de determinar cada una de las actividades a ejecutarse en la evaluación del caso de estudio (EVAC).

b) Alcance

Características

El modelo de calidad que se utilizará para realizar la evaluación es el Modelo de Calidad preescrito en la norma ISO/IEC 9126-1. Las características que componen el módulo de evaluación son: Funcionalidad, Fiabilidad, Usabilidad, Eficiencia, Portabilidad las cuales se especifican en la siguiente tabla:

CARACTERISTICA	SUBCARACTERISTICA	METRICA
FUNCIONALIDAD	Cumplimiento de la Funcionalidad	Cumplimiento de la Funcionalidad
FIABILIDAD	Tolerancia a Fallas	Anulación de Operación Incorrecta
USABILIDAD	Capacidad para ser entendido	Funciones Evidentes
	Operabilidad	Claridad del mensaje

CARACTERISTICA	SUBCARACTERISTICA	METRICA
		Recuperabilidad de error operacional
PORTABILIDAD	Adaptabilidad	Adaptabilidad de la estructura de datos

Nivel de evaluación

Según el Anexo F se determina que la evaluación es de nivel D en todos los aspectos (salvamento, económico, seguridad y medioambiental), ya que este no representa significativos.

Técnicas

Para realizar las mediciones de las métricas establecidas en el Anexo D se utilizará la técnica de Inspección y como procedimiento la revisión de código. Las herramientas utilizadas para realizar las medidas son propias de NetBeans 5.5 y son las siguientes:

- Examinador de Objetos
- Vista de Clases
- Ir a referencia
- Esquematización (Colapsar rutina o clase)
- Comando Buscar

Aplicabilidad

El propósito de la evaluación es determinar la calidad del código fuente de la aplicación EVAC desarrollado en lenguaje JAVA, usando métricas de las normas internacionales ISO/IEC 9126 e ISO/IEC 14598

c) Referencias

La aplicación a evaluar NO depende de resultados anteriores ya que es la primera versión.

d) Términos y Definiciones

Métrica: es una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado [IEEE "Standard Glosary of Software Engering Terms].

Componente: Los componentes del producto de Software son: Especificación de los requisitos de Software, Código fuente del producto, Código Ejecutable, Manual técnico, Manual de usuario, Manual de instalación, Documentación del Desarrollo del producto de Software. [ISO/IEC 14598-5].

Niveles de Puntuación: La escala de los valores a ser medidos, en el cual se determinan los valores aceptables de calidad.

Modelo MOSCA: (Modelo Sistémico para estimar la Calidad de los Sistemas de Software) desarrollado en la Universidad Simón Bolívar de Caracas Venezuela

Error: Problema humano que se traduce en un Software incorrecto (por ejemplo: entender mal lo que quiere el usuario a nivel de los requisitos) [5]

[5] CARRILLO, BRITO. Evaluación De La Calidad Del Código Fuente En Base A Las Normas Iso/Iec 9126, Iso/Iec 14598. 2005.

e) Entradas y Métricas

Entradas para la evaluación

Componentes del Producto: Especificación de los requisitos de Software, Código fuente del producto, Código Ejecutable, Manual técnico, Manual de usuario, Manual de instalación, Documentación del Desarrollo del producto de Software

Elementos de los datos

- Número de detalles que se han reunido y que requieren cumplimiento de Funcionalidad.
- Número de funciones implementadas que evitan críticas y serias fallas causadas por operaciones incorrectas.
- Número de funciones que son evidentes al usuario.
- Número de mensajes implementados con explicaciones claras.
- Número de funciones implementadas que toleran errores de usuarios.
- Número de estructuras de datos que son operables y no tienen ninguna limitación.

Métricas y medidas

$$Vsc = \frac{\sum m}{n}$$; donde: Vsc=Valor de subcaracterística, m= Valor de Métrica y n= número de métricas.

$$Vc = \frac{\sum Vsc}{nsc}$$; donde: : Vc= Valor de característica, Vsc=Valor de subcaracterística, nsc= número de subcaraterísticas.

f) Interpretación de Resultados

Mapeo de medidas

Los niveles de puntuación se fundamentan en el Modelo MOSCA, en el cual se determinan que los valores aceptables deben ser mayores o iguales al 0.75 (75%).

Informes

Fase de Evaluación	Informe
Establecer requisitos de la evaluación	Requisitos de la evaluación: describen los objetivos de la evaluación, en particular, describe requisitos de calidad para el producto
Especificación de la evaluación	La especificación de la evaluación define todo el análisis y medidas a realizar en el producto y en sus componentes
Diseño de la evaluación	El plan de la evaluación describe procedimientos operacionales necesarios para llevar a cabo la especificación de la evaluación; en particular se describen todos los métodos y herramientas a usarse en la evaluación
Ejecución de la evaluación	Los registros de la evaluación se fundamentan del plan de evaluación, llevando una cuenta del detalle de acciones realizadas por el evaluador mientras ejecuta el plan de la evaluación; estos archivos son guardados o almacenados por el evaluador. El borrador del informe de la evaluación es un documento producido por la síntesis de los resultados de la evaluación.
Conclusión de la evaluación	El informe de la evaluación contiene requisitos de la evaluación, la especificación de la evaluación, los resultados de las medidas y análisis realizados y cualquier otra información necesaria para poder repetir o reproducir la evaluación

SUMARIZACIÓN DE SUBCARACTERÍSTICAS

FUNCIONALIDAD

SUBCARACTERISTICA	METRICA	∑m /n	Vsc
Cumplimiento de la Funcionalidad	Cumplimiento de la Funcionalidad	1	1

FIABILIDAD

SUBCARACTERISTICA	METRICA	∑m /n	Vsc
Tolerancia a Fallas	Prevención del Mal Uso de Datos	0.87	0.87

USABILIDAD

SUBCARACTERISTICA	METRICA	∑m /n	Vsc
Capacidad para ser Entendido	Funciones Evidentes	1.00	1.00
Operabilidad	Claridad del mensaje	1.00	1.00
	Recuperabilidad de error operacional	1.00	

PORTABILIDAD

SUBCARACTERISTICA	METRICA	∑m /n	Vsc
Adaptabilidad	Adaptabilidad de la Estructura de Datos	1.00	1.00

SUMARIZACIÓN DE CARACTERÍSTICAS

CARACTERISTICA	∑ Vsc	nsc	Vc
Funcionalidad	1	1	1.00
Fiabilidad	0.87	1	0.87
Usabilidad	2.00	2	1.00
Portabilidad	1.00	1	1.00

ANEXO H. NIVELES DE EVALUACIÓN

Se refiere a los tipos de niveles existentes para los diferentes aspectos cuando se realiza evaluación.

NIVEL	Aspectos de salvamento	Aspectos Económicos	Aspectos de seguridad	Aspectos medioambientales
A	Muchas personas muertas	Desastre financiero (Compañía no puede sobrevivir)	Protección de datos estratégicos y servicios.	Daño medioambiental irrecuperable.
B	Amenaza a las vidas humanas	Grandes pérdidas económicas (Compañía comprometida)	Protección de datos críticos y servicios.	Recuperable daño medioambiental.
C	Daño a la propiedad, pocas personas heridas	Pérdidas económicas significantes (Compañía afectada)	Protección contra riesgo del error.	Contaminación local.
D	Pequeños daños a la propiedad, sin riesgos a las personas.	Pérdidas económicas despreciables	Ningún riesgo específico identificado.	Ningún riesgo medioambiental

Fuente: ISO / IEC 14598 - 6

ANEXO I. GLOSARIO DE TÉRMINOS

EVAC: Herramienta para evaluar la calidad del código fuente generado en C ANSI.

RUP: Proceso Racional Unificado, es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

Parser: el analizador sintactico o parser de expresiones regulares de un lenguaje de programación

JavaCC: Java Compiler Compiler, es un generador de parser de código abierto (open source) para lenguaje de programación Java. JavaCC es similar a Yacc.

PDF: tipo de documento Adobe Acrobat Document.

UML: Unified Modeling Language, Lenguaje Unificado de Modelado, es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir, su objetivo es lograr que, además de describir con cierto grado de formalismo tales sistemas, puedan ser entendidos por los usuarios de aquello que se modela.

Característica: un conjunto de atributos de un ente ó específicamente, de un producto, por medio de los cuales se describe y evalúa su calidad. Una característica permite ser descripta y evaluada por la descomposición recursiva en subcaracterísticas y/o atributos. Entre las características que describen a la calidad desde cierto perfil de usuario, se pueden citar a la usabilidad, funcionalidad, confiabilidad, eficiencia, portabilidad, y mantenibilidad.

Componente: Los componentes del producto de Software son: Especificación de los requisitos de Software, Código fuente del producto, Código Ejecutable, Manual

técnico, Manual de usuario, Manual de instalación, Documentación del Desarrollo del producto de Software. [ISO/IEC 14598-5].

Defecto: Anomalía en un producto, o equivocación, (suele ser la consecuencia de un error: así un error en los requisitos puede originar un diseño defectuoso)

Error: Problema humano que se traduce en un Software incorrecto (por ejemplo: entender mal lo que quiere el usuario a nivel de los requisitos)

ISO/IEC: ISO (International Organization for Standardization) y la IEC (International Electrotechnical Commission) conforman el sistema especializado para la normalización o estandarización para la industria así como para la ciencia de la computación a nivel mundial.

Métrica: es una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado [IEEE "Standard Glosary of Software Engering Terms].

Modelo de Calidad: conjunto de características, subcaracterísticas y las relaciones entre ellas que proveen la base para especificar requisitos de calidad con el fin de realizar un proceso de evaluación.

Modelo MOSCA: Modelo Sistémico para estimar la Calidad de los Sistemas de Software, desarrollado en la Universidad Simón Bolívar de Caracas Venezuela.

Niveles de Puntuación: La escala de los valores a ser medidos, en el cual se determinan los valores aceptables de calidad.

Proceso de Evaluación: (Proceso) un conjunto parcialmente ordenado de subprocesos a los que se les asocia una colección de recursos, agentes, condiciones, y constructores de proceso, con el fin de producir los distribuibles conforme a las metas establecidas, en un proyecto de evaluación.

Proceso de Software. (Proceso) un conjunto parcialmente ordenado de subprocesos a los que se les asocia una colección de recursos, agentes, condiciones, y constructores de proceso, con el fin de producir los distribuibles conforme a las metas establecidas en un proyecto de software.

Subcaracterística: parte o componente de una característica de un ente (producto, proceso, o recurso), usualmente como resultado de aplicar un mecanismo de descomposición.