Universidad de las Ciencias Informáticas
Facultad 3

Título: Desarrollo de una herramienta para la evaluación de la mantenibilidad mediante métricas en el Laboratorio de Pruebas del CEIGE.

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

Autores:
Martha Rocío Fonseca Vega
Fernando Izaguirre Delgado

Tutoras:
Ing. Giselle Almeida González
Ing. Daileny Caridad Arias Pupo

Cotutor:
Lic. Miguel Dongil y Sánchez

Ciudad de la Habana
Año del 54 Aniversario del Triunfo de la Revolución

"En el momento en que asocias un número con una idea,
aprendes algo nuevo.
Entonces, sabes algo más de lo que sabías antes".

Paul F. Lazarsfeld

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Martha Rocío Fonseca Vega

_____

Firma del Autor

Fernando Izaguirre Delgado

_____

Firma del Autor

Ing. Giselle Almeida González

_____

Firma del Tutor

Ing. Daileny Caridad Arias Pupo

_____

Firma del Tutor

DATOS DE CONTACTOS

Síntesis de los tutores y el cotutor:

- Ing. Giselle Almeida González

Graduada de Ingeniera en Ciencias Informáticas en la UCI en el 2008. A partir de ese momento se incorpora como Analista en la línea de Costos y Procesos en el proyecto ERP Cuba en el Centro para la Informatización de Gestión de Entidades. Actualmente se desempeña como Especialista de Calidad en el grupo de Gestión de Calidad del mismo centro. En su trabajo de diploma identificó, modeló y realizó una propuesta de mejoras de los procesos de negocio que se desarrollaban en el laboratorio de Calisoft. Ha sido tribunal de tesis y ha impartido clases de Contabilidad y Finanzas, Ingeniería de Software I e Ingeniería de Software II.

Categoría docente: Instructor.

Jefa del grupo de Verificación y Validación del centro CEIGE.

Correo electrónico: galmeida@uci.cu

- Ing. Daileny Caridad Arias Pupo

Graduada de Ingeniera en Ciencias Informáticas en el año 2011. RGA de la Universidad de Ciencias Informáticas (UCI), Jefa del proyecto Despliegue de Cedrux en Calisoft con 4 años de experiencia en el desarrollo de software.

Correo electrónico: dcarias@uci.cu

- Lic. Miguel Dongil y Sánchez

Licenciado en Historia por la Universidad de Oviedo (España). Año 2010. Premio Fin de Carrera. Posteriormente cursó y obtuvo los títulos de Maestro en Métodos y Técnicas de Investigación Artística, Histórica y Geográfica y de Maestro en Formación del Profesorado de Formación Secundaria Obligatoria, Bachillerato y Formación Profesional. Actualmente es Investigador del Departamento de Historia de la citada Universidad, donde desarrolla su doctorado. Paralelamente es Miembro de la Comisión de Gobierno y de la Comisión de Docencia del Consejo de Departamento de Historia.

Correo electrónico: dongilmiguel@uniovi.es

AGRADECIMIENTOS

Primeramente agradecerle a nuestro Comandante en Jefe Fidel Castro Ruz por haber tenido la hermosa idea de crear la Universidad de las Ciencias Informáticas, que es una excelente casa de estudios.

A nuestros padres, por estar presentes dándonos todo su apoyo y sembrando en nosotros la semilla de los principios y valores.

A nuestras tutoras las Ingenieras Giselle Almeida González y Daileny Caridad Arias Pupo y al cotutor el Licenciado Miguel Dongil y Sánchez, por sus consejos, orientación, ayuda y paciencia.

A nuestros amigos de la universidad por permitirnos conocerlos y ser parte de su vida, especialmente a Yissell Soto Hidalgo, Reinier Silverio Figueroa y Janier Treto Portal por ayudarnos y estar con nosotros a lo largo de la carrera.

A todos nuestros familiares que cada día están presentes y nos dan su ayuda incondicional.

DEDICATORIA

Martha Rocío:

A mis padres por hacer de mí quien soy hoy y con sus consejos guiarme por los mejores caminos...

Fernando:

A mis padres y abuelas por estar conmigo en todo momento y ser parte imprescindible de mi vida…

RESUMEN

Uno de los problemas más importantes que se afronta en la esfera de la informática es la calidad del software, siendo motivo de preocupación para especialistas, ingenieros, investigadores y comercializadores, desde hace varias décadas, pues pese a que ningún sistema de software es perfecto, todo proyecto tiene como objetivo lograr un producto con la mejor calidad posible.

Un atributo de calidad primordial es la mantenibilidad de software, ya que esta influye directamente en el coste de los productos y en el tiempo de implementación. Actualmente en el Centro de Informatización de la Gestión de Entidades (CEIGE) perteneciente a la Universidad de Ciencias Informáticas (UCI), no existe un sistema automatizado capaz de medir dicho atributo de la calidad.

La presente investigación se basa en el diseño e implementación de una herramienta para la evaluación de la mantenibilidad mediante métricas externas en el Laboratorio de Pruebas del CEIGE. Con ella se pretende determinar el grado de mantenibilidad de los productos elaborados en el centro en la etapa de pruebas, mitigando posibles riesgos que atenten contra la sostenibilidad y coste de estos que pudieran afectar su la calidad.

Palabras Claves

Calidad, métrica externa, mantenibilidad de software.

Índice

Índice de Tablas.

Índice de Ilustraciones.

Introducción

La calidad del software es uno de los retos más importantes, al que se le dedican muchos esfuerzos en la actualidad. Aunque el software casi nunca es perfecto, todo proyecto tiene como objetivo producirlo con la mejor calidad posible, de modo que cumpla, y supere las expectativas de los usuarios.

Cuba no está exenta de lo anteriormente mencionado, por lo que dedica parte de sus esfuerzos humanos y tecnológicos a mejorar y fortalecer en todos sus centros de desarrollo de software la creación de productos con mayor calidad.

La Universidad de las Ciencias Informáticas (UCI), haciendo uso del gran potencial profesional con el que cuenta, ha creado varios centros para el desarrollo de software con el propósito de insertar en el mercado nacional e internacional productos de gran calidad, fomentando el prestigio en este ámbito del país. Dentro de estos se encuentra el Centro de Informatización de la Gestión de Entidades (CEIGE) y como parte de la estructura del mismo se halla la Subdirección de Producción conformada por un grupo de Arquitectura y Calidad. Dicho grupo tiene entre sus objetivos la validación y verificación de los productos informáticos que se desarrollan en el centro. Dentro de este se halla el Laboratorio de Pruebas Internas que tiene entre sus principales metas llevar a cabo el proceso de pruebas de liberación a los productos del CEIGE que soliciten el servicio.

El proceso de pruebas tiene como objetivo evaluar características de calidad, tales como: la fiabilidad, eficiencia, usabilidad, mantenibilidad, portabilidad y funcionalidad, para ello se aplican los diferentes tipos de prueba que verifican su cumplimiento, por ejemplo, las pruebas de usabilidad, funcionalidad, recuperación, tolerancia a fallas, carga y estrés.

Actualmente no existe una herramienta en el Laboratorio de Pruebas del centro para evaluar el cumplimiento de uno de estos atributos, siendo este la mantenibilidad del software.

Múltiples estudios señalan a la mantenibilidad como la parte más costosa del ciclo de vida del software y estadísticamente está comprobado que el coste de mantenimiento de un sistema a lo largo de toda su vida útil supone más del doble que los costes de su desarrollo (22), por lo que de no evaluarse este

atributo de calidad en los productos desarrollados en el CEIGE podrían generarse riesgos que atenten contra la sostenibilidad y coste de estos, viéndose así afectada la calidad e integridad de los mismos. Considerando lo analizado anteriormente, se define el siguiente **problema a resolver:** La insuficiencia en el proceso de medición de los componentes que se prueban en el Laboratorio de Pruebas del CEIGE, no permite estimar el grado de mantenibilidad de los productos del centro.

A partir del problema planteado se define como **objeto de estudio** a la mantenibilidad en el desarrollo del software.

Para darle solución al problema a resolver planteado se traza el siguiente **objetivo general:** Implementar métricas para la evaluación de los componentes que se prueban en el Laboratorio de Pruebas Internas del CEIGE, posibilitará estimar el grado de mantenibilidad de los productos del centro.

Para dar cumplimiento a lo anteriormente planteado se desglosan los siguientes **objetivos específicos:**

- Establecer el marco teórico-conceptual para fundamentar la investigación.
- Diseñar la solución de software para facilitar la implementación de las métricas que permiten evaluar la mantenibilidad de los componentes en el Laboratorio de Pruebas del CEIGE.
- Implementar el diseño de la solución de software para favorecer la determinación del grado de mantenibilidad de los componentes en el Laboratorio de Pruebas del CEIGE.
- Validar la solución de software propuesta mediante la aplicación de pruebas de Caja Blanca y Caja Negra.

El **campo de acción** de la presente investigación comprende la evaluación de la mantenibilidad del software, por tanto se concreta la siguiente **idea a defender**: Desarrollar una herramienta informática para la evaluación de la mantenibilidad de software mediante el uso de métricas externas, asegurando que cumpla con los requerimientos de calidad exigidos por Calisoft-UCI, posibilitará estimar el grado de mantenibilidad de los productos del CEIGE una vez puesta en explotación.

Como **tareas de la investigación** se definen las siguientes:

- Descripción de los conceptos de métricas de mantenibilidad, paradigmas de la Ingeniería de Software que han tenido entre sus principales objetivos la mantenibilidad.
- Caracterización del estado actual que presenta la mantenibilidad en los productos de software del CEIGE.
- Caracterización de las métricas y herramientas existentes para evaluar las sub-características de la mantenibilidad: analizabilidad, cambiabilidad y estabilidad.
- Caracterización de las métricas y herramientas existentes para evaluar las sub-características de la mantenibilidad: facilidad de pruebas y conformidad con la mantenibilidad.
- Selección de las métricas a implementar en la solución de software.
- Definición de las tecnologías a utilizar en la desarrollo de la solución de software propuesta.
- Análisis y diseño de la solución de software para la evaluación de la mantenibilidad de los componentes en el Laboratorio de Pruebas del CEIGE.
- Implementación de la solución de software para la evaluación de la mantenibilidad de los componentes en el Laboratorio de Pruebas del CEIGE.
- Validación de la solución propuesta mediante la aplicación de pruebas de Caja Negra.
- Validación de la solución propuesta mediante la aplicación de pruebas de Caja Blanca.

Para el desarrollo de este trabajo se combinan diferentes métodos y técnicas, como se muestra a continuación:

A nivel teórico:

Métodos de análisis-síntesis e inducción-deducción: para el estudio de las concepciones y conceptos empleados en el campo de la mantenibilidad de software, permitiendo la extracción de los elementos más importantes que se relacionan en este campo.

Análisis histórico-lógico: para conocer, con mayor profundidad, los antecedentes y las tendencias actuales referidas a la mantenibilidad del software; además de conceptos, términos y vocabularios propios del campo.

A nivel empírico:

Revisión de documentos: para el estudio y análisis, partiendo de documentación científica, de los conceptos relacionados con la mantenibilidad como atributo de la calidad del software.

El siguiente trabajo de diploma está definido estructuralmente en tres capítulos, descritos a continuación:

CAPÍTULO 1 Fundamentación teórica: en este capítulo se describe el estado del arte de los sistemas existentes para medir la mantenibilidad del software, se realiza el análisis de los conceptos fundamentales relacionados con el tema en cuestión, así como las técnicas, tendencias y tecnologías. Además se definen y adaptan métricas externas que posibiliten el cálculo de la mantenibilidad y se fundamenta la utilización de los lenguajes, herramientas y el modelo de desarrollo del software empleado.

CAPÍTULO 2 Análisis y diseño del sistema*:* en este capítulo se hace un análisis y descripción general de la propuesta de desarrollo del sistema, así como la realización del levantamiento de los requisitos funcionales y no funcionales, entrada de la implementación del software. También se tienen en cuenta la definición del modelo de dominio y el modelo de clases, al igual que los métodos a desarrollar más significativos. Se lleva a cabo una explicación de la organización del sistema expuesta en el diagrama de clase para el correcto funcionamiento de este.

CAPÍTULO 3 Implementación, validación y prueba: en este capítulo se describe la construcción de la solución, explicando los aspectos principales de la implementación. También se realiza una descripción de las clases y funcionalidades del sistema para evaluar la mantenibilidad, así como una muestra de los resultados obtenidos en la validación del sistema, además se hace el diseño y ejecución de pruebas sobre dicha aplicación en busca de errores relacionados con su funcionalidad.

Capítulo 1: Fundamentación teórica

1.1 Introducción

En este capítulo se abordarán temas y conceptos relacionados con la mantenibilidad del software. Se definirá su importancia, así como la relación existente entre la mantenibilidad y el proceso de desarrollo del software.

Se pretenden mencionar los más recientes sistemas que permiten medir la mantenibilidad del software en la Universidad de Ciencias Informáticas (UCI) como en el mundo, mostrando sus ventajas y desventajas. Siendo el objetivo principal de este capítulo la realización de un estudio sobre el estado del arte, el modelo de desarrollo, las tecnologías, herramientas, tendencias y lenguajes que serán empleados para el desarrollo de una aplicación informática que permita evaluar la mantenibilidad del software, todo ello, con el propósito de ser utilizado como soporte teórico de la investigación.

1.2 Definición de Mantenibilidad

La IEEE 1990 define mantenibilidad como: “La facilidad con la que un sistema o componente de software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno”. (1)

En la ISO 9126-1 del año 1998 se define a la mantenibilidad como la capacidad de un producto de software para ser modificado. Las modificaciones pueden incluir correcciones, mejoras o adaptación del software a cambios en el entorno, en los requerimientos o en las especificaciones funcionales. Está indicada por los siguientes sub-atributos: facilidad de análisis (analizabilidad), facilidad de cambio (cambiabilidad), estabilidad, conformidad y facilidad de prueba. (2)

Otras bibliografías consultadas establecen a la mantenibilidad como la capacidad de un programa de ordenador para ser retenido en su forma original, y se restaure a esta en caso de un fallo. (3) Además, se define que la mantenibilidad es la facilidad con la que se pueden realizar cambios a un sistema de software. Estos cambios pueden ser necesarios para la corrección de fallos, la adaptación del sistema para satisfacer nuevos requisitos o funciones, la eliminación de funcionalidades existentes, así como la corrección de errores o deficiencias en busca del perfeccionamiento o adaptación del software, para reducir aún más los costes de mantenimiento. (4)

Teniendo en cuenta las definiciones y criterios anteriormente expuestos, los autores de esta investigación precisan que la mantenibilidad es la capacidad de un producto de software para ser modificado. Estas modificaciones pueden incluir correcciones, mejoras o adaptación del software a cambios en el entorno, en los requerimientos o en las especificaciones funcionales. Se subdivide en cinco sub-características:

Analizabilidad: capacidad del producto de software de diagnosticar sus deficiencias o causas de fallos, o de identificar las partes que deben ser modificadas.

Cambiabilidad: capacidad del producto de software de permitir implementar una modificación especificada previamente. La implementación incluye los cambios en el diseño, el código y la documentación. Si el software es modificado por el usuario final, entonces, la cambiabilidad puede afectar a la operabilidad.

Estabilidad: capacidad del producto de software de minimizar los efectos inesperados de las modificaciones.

Facilidad de prueba: capacidad del producto de software de permitir evaluar las partes modificadas.

Conformidad: capacidad del producto de software de satisfacer los estándares o convenciones relativas con la mantenibilidad.

1.2.1 Importancia de la Mantenibilidad

El esfuerzo en el mantenimiento del software podría disminuir de producirse un software nuevo de mejor calidad. Igualmente, también se podrán reducir los costes futuros, si el mantenimiento se realiza utilizando técnicas que mejoren alguna de sus características de calidad.

La mantenibilidad es el atributo de calidad del software que de forma más directa influye en los costes y necesidades del mantenimiento:

A mayor mantenibilidad (facilidad de mantenimiento), menores costes de mantenimiento, y viceversa.

Esta, establecida como objetivo en las fases iniciales del ciclo de vida del software reduce las posteriores necesidades del mantenimiento.

La mantenibilidad centrada como objetivo, durante la fase de mantenimiento reduciría los efectos laterales y otros inconvenientes ocultos.

Este atributo de calidad permite conocer la situación en la que se encuentra el producto de software, lo que facilita cómo gestionar y realizar cada petición de mantenimiento y, en general, planificar mejor el proceso de mantenimiento. (5)

1.2.2 Aspectos que influyen en la Mantenibilidad

Existen unos pocos factores que afectan directamente a la mantenibilidad, de forma que si alguno de ellos no se satisface adecuadamente, esta se resiente. Los tres más significativos son:

Proceso de desarrollo: la mantenibilidad debe formar parte integral del proceso de desarrollo del software. Las técnicas utilizadas deben ser lo menos intrusivas posible con el software existente. Los problemas que surgen en muchas organizaciones de mantenimiento son de doble naturaleza: mejorar la mantenibilidad y convencer a los responsables de que la mayor ganancia se obtendrá únicamente cuando la mantenibilidad esté incorporada intrínsecamente en los productos de software.

Documentación: En múltiples ocasiones, ni la documentación ni las especificaciones de diseño están disponibles, y por tanto, los costes se incrementan debido al tiempo requerido para que un mantenedor entienda el diseño del software antes de poder ponerse a modificarlo. Las decisiones sobre la documentación que debe desarrollarse son muy importantes cuando la responsabilidad del mantenimiento de un sistema se va a transferir a una organización nueva.

Comprensión de programas: La causa básica de la mayor parte de los altos costes es la presencia de obstáculos a la comprensión humana de los programas y sistemas existentes. Estos obstáculos surgen de tres fuentes principales:

La información disponible es incomprensible, incorrecta o insuficiente.

La complejidad del software, de la naturaleza de la aplicación o de ambos.

La confusión, mala interpretación u olvidos sobre el programa o sistema.

La utilización de algunas prácticas de desarrollo (técnicas de programación estructurada, paquetes software estándares y generadores de listados) suponen la reducción del esfuerzo de mantenimiento,

mientras que otras prácticas (uso de generadores de código) implican un aumento de la necesidad de mantenimiento**.**

Los factores concretos que influyen en la mantenibilidad son los siguientes:

- Falta de cuidado en las fases de diseño, codificación o prueba.
- Pobre configuración del producto de software.
- Adecuada calificación del equipo de desarrolladores del software.
- Estructura del software fácil de comprender.
- Facilidad de uso del sistema.
- Empleo de lenguajes de programación y sistemas operativos estandarizados.
- Estructura estandarizada de la documentación.
- Documentación disponible de los casos de prueba.
- Incorporación en el sistema de facilidades de depuración.
- Disponibilidad del equipo (computador y periféricos) adecuado para realizar el mantenimiento.
- Disponibilidad de la persona o grupo que desarrolló originalmente el software.
- Planificación del mantenimiento.(5)

1.2.3 Propiedades de la Mantenibilidad

La mantenibilidad se puede considerar como la combinación de dos propiedades diferentes: reparabilidad y flexibilidad.

Reparabilidad:

Un sistema informático es reparable si permite la corrección de sus defectos con una cantidad de trabajo limitada y razonable. Un componente de software no se deteriora con el paso del tiempo, por lo que la reparabilidad de este se ve afectada únicamente por la cantidad y tamaño de los componentes o piezas:

– Un producto de software que consiste en módulos bien diseñados es más fácil de analizar y reparar que uno monolítico, pero el incremento del número de módulos no implica un producto más reparable, ya que también aumenta la complejidad de las interconexiones entre módulos.

– Se debe buscar un punto de equilibrio con la estructura de módulos más adecuada para garantizar la reparabilidad facilitando la localización y eliminación de los errores en unos pocos módulos.

Flexibilidad:

Un sistema de software es flexible (evolucionable) si permite cambios para que se satisfagan nuevos requerimientos.

– El software es de naturaleza muy maleable ya que, debido a su carácter inmaterial, resulta mucho más fácil cambiar o incrementar sus funciones que en productos de naturaleza física (hardware).

– La flexibilidad disminuye con cada nueva versión de un producto informático. Cada versión complica la estructura del software y, por tanto, las futuras modificaciones serán más difíciles.

– La flexibilidad es una característica tanto del sistema informático como de los procesos relacionados. En términos de estos últimos, los procesos deben poderse acomodar a nuevas técnicas de gestión y organización o a cambios en la forma de entender la ingeniería (5).

1.3 Métricas

La definición más común de métrica del software es la siguiente: “Una métrica de software es un atributo del entorno de desarrollo del software, derivada de la medida de los atributos de ciertos componentes del software”. Un atributo es una cualidad, una propiedad o una característica de un objeto. En el entorno de desarrollo del software, el tamaño, el coste y el esfuerzo son algunos de los atributos del proyecto de software.

La IEEE 610.12 “Standard Glosar of Software Engering Terms” define como métrica: “Una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado”. (1)

Desde otro punto de vista bien distinto, Fenton aporta la siguiente definición (6): “Una métrica de software es una correspondencia entre uno o más atributos del entorno de desarrollo del software, y cualquier otro atributo”.

Con lo anteriormente expuesto se puede decir que una métrica es el número que se asocia a una idea, o sea, una métrica es la variable o medida de ciertos aspectos cuantitativos de un sistema. Por ejemplo, para un sistema de software, dichos aspectos cuantitativos serían el alcance, el tamaño, el coste y los riesgos, por solo mencionar algunos.

Muchos investigadores han intentado desarrollar una sola métrica que proporcione una medida completa de la complejidad del software. Aunque se han propuesto docenas de métricas o medidas, cada una de estas tiene un punto de vista diferente. Pese a que existe la necesidad de medir y controlar la complejidad del software, es difícil de obtener un solo valor de estas métricas de calidad.

Aunque todos estos obstáculos son motivo de preocupación, no son causa de rechazo hacia las métricas. Es por eso que la medición es esencial si se desea, realmente, alcanzar la calidad en el software.

Existen distintos tipos de métricas para poder evaluar, mejorar y clasificar el software final, que serán manejadas dependiendo del entorno de desarrollo del sistema al cual pretendan orientarse y dirigidas específicamente a características o atributos de la calidad.

En orden de determinar mejor los costes de los productos desarrollados en el CEIGE y evitar riesgos que atenten contra la sostenibilidad, calidad e integridad de los mismos, este trabajo está encaminado al análisis y diseño de una herramienta basada en métricas para evaluar la mantenibilidad, atributo de gran importancia en la calidad del software, teniendo en cuenta que las medias de la mantenibilidad una vez finalizado un sistema informático pueden servir de guía durante el proceso de mantenimiento, para evaluar el impacto de un cambio o para realizar un análisis comparativo entre varias propuestas.

1.3.1 Métricas de la Mantenibilidad

Existe una estrecha relación entre los conceptos de calidad del software y de mantenibilidad. Por esta razón, los estándares ISO e IEEE proponen métricas de calidad para medir la mantenibilidad del software.

Las métricas de mantenibilidad no pueden medir el coste de realizar un cambio particular al sistema, sino que miden aspectos de la complejidad y la calidad de los programas ya que existe una alta correlación entre la complejidad y la mantenibilidad (a mayor complejidad menor mantenibilidad) y entre la calidad y la mantenibilidad (a mayor calidad mayor mantenibilidad).

Existen maneras de medir la mantenibilidad para todos los elementos del software que están o estarán sometidos al mantenimiento: código, documentos de usuario, documentos de análisis o diseño.

El estándar ISO 9126, por el cual se rige el país, incluye un análisis de las métricas de calidad (entre ellas, las de mantenibilidad) y las interrelaciones entre unas y otras.

La ISO 9126 está dividida en 4 partes, las métricas específicamente ajustadas a la mantenibilidad se definen en la ISO 9126-2, donde se muestran métricas externas aplicadas al producto en ejecución, y en la ISO 9126-3 que expone las métricas internas para medir los atributos de las características de calidad definidas en la norma 9126-1. El estándar ISO 9126 también plantea el problema de la elección de métricas y los criterios de medida, así como los requerimientos que deben cumplir las mediciones utilizadas para comparar.

Existen dos aproximaciones diferentes para medir la mantenibilidad, según se tengan en cuenta los aspectos externos o internos de los atributos considerados.

– La mantenibilidad es claramente un atributo de producto externo porque no depende únicamente del producto, sino también de la persona que realiza el mantenimiento, del soporte documental, de las herramientas disponibles, y de la utilización real del software. La aproximación externa más directa para medir la mantenibilidad consiste en medir el proceso de mantenimiento; si el proceso es efectivo, entonces se asume que el producto es mantenible.

– La aproximación alternativa se utiliza para identificar atributos internos del producto y determinar cuáles de ellos son predictivos de las medidas del proceso.

Aunque esta aproximación es más práctica puesto que las medidas pueden realizarse fácilmente, la mantenibilidad nunca se puede definir sólo en términos de medidas internas.

Por dichas razones la presente investigación tomando como base el estándar utilizado en Cuba ISO/IEC 9126, pretende llevar a cabo la elaboración de una herramienta cuyo objetivo principal es el de medir la mantenibilidad mediante métricas de carácter externas, que no son más que aquellas aplicables al software en ejecución, relacionadas con la conducta del mantenedor o el usuario, cuando se hace alguna modificación sobre una aplicación informática en la realización de pruebas o el mantenimiento luego de terminado un sistema en el Laboratorio de Pruebas del CEIGE.

Las métricas externas utilizadas según el estándar ISO 9126, posibilitarán medir sub-atributos de la mantenibilidad como:

Analizabilidad: miden atributos relacionados con el esfuerzo del mantenedor, el usuario o los recursos gastados para diagnosticar deficiencias o causas de fallos, o para identificar las partes que deben ser modificadas.

Cambiabilidad: miden atributos relacionados con el esfuerzo del mantenedor o el usuario para medir la conducta del mantenedor, el usuario o el sistema de software cuando se intenta llevar a cabo una modificación determinada.

Estabilidad: miden atributos relacionados con la conducta inesperada del sistema cuando dicho software es probado u operado después de una modificación.

Facilidad de prueba: miden atributos relacionados con el esfuerzo del mantenedor o el usuario para medir la conducta del mantenedor, el usuario o el sistema cuando se intenta probar el software.

Conformidad: miden atributos relacionados con el número de casos u ocurrencias en que el producto no cumple las normas, convenciones o regulaciones requeridas relacionadas con la mantenibilidad.

El sub-atributo analizabilidad de la mantenibilidad, según la ISO 9126 posee dos métricas externas para su evaluación cuyos nombres son: “Grado de implementación de las funciones de diagnóstico” y “Capacidad de análisis de fallos”, seleccionándose esta última para el desarrollo de la herramienta, por ser capaz de medir de forma sencilla los recursos o el esfuerzo del usuario cuando se requiere diagnosticar las causas de un fallo del software, o identificar las partes que deben ser modificadas.

Nombre de la métrica	La métrica se propone medir	Método de aplicación	Medición (fórmula)	Interpretación del valor obtenido	Escala	Tipo de medida	Fuentes	Ref. a la ISO/IEC 12207	Destinatario
Capacidad de análisis de fallos	¿Puede el usuario identificar una operación que causa un fallo? (y evitarla, recurriendo a una operación alternativa) ¿Puede el serviciador fácilmente encontrar la causa del fallo?	Observe el comportamiento del operador que trata de resolver los fallos utilizando funciones de diagnóstico.	X = 1- A / B A - número de fallos cuyas causas aún no han sido encontradas. B - número total de fallos registrados (Recuperación y tolerancia a fallos)	0 <= X <= 1 Mientras más cercano al 1, mejor	Absoluta	X = Contable/contable A = Contable B = Contable	Informe de evaluación Reporte de solución de problemas Informe de implantación	5.3 Pruebas de calificación 5.4 Implantación 5.5 Mantenimiento	Desarrollador Evaluador Serviciador

Ilustración 1 Tabla de la métrica "Capacidad de análisis de fallos".

Otro sub-atributo a evaluar de la mantenibilidad es la cambiabilidad, encargada de medir los atributos, como el esfuerzo del personal de mantenimiento o del usuario así como de su comportamiento, incluyendo el software, cuando se trata de implementar una modificación especificada. Se escogió entre 5 métricas posibles la que lleva por nombre: "Tiempo empleado en implementar un cambio por el personal de mantenimiento", utilizada en la medición de la mantenibilidad por el Centro Nacional de Calidad de Software (CALISOF) (7) y que responde a la pregunta: ¿Puede el personal de mantenimiento realizar fácilmente cambios al software para resolver los problemas del fracaso? Se calcula:

$$Tp = (Tout - Tin)/N \quad [1]$$

Donde:

Tp: es el tiempo promedio.

Tout: es el tiempo promedio en que se resuelven los fallos.

Tin: es el tiempo promedio en el que se encuentran los fallos.

N: es el número de fallos registrados y eliminados.

Para la utilización de esta métrica debe observarse el comportamiento del usuario y el personal de mantenimiento al tratar de modificar el software. En caso contrario, es necesario analizar el informe de

resolución de problemas o el informe de mantenimiento. Se debe acotar que el trabajo en el área se realiza por turnos, mañana y tarde, cada uno se conforma por 4 horas, haciendo un total de 8 horas laborales.

En la evaluación del sub-atributo Facilidad de Prueba se seleccionó una métrica capaz de controlar el esfuerzo del personal de mantenimiento o del usuario, midiendo el comportamiento de estos y del software, que puede haber sido modificado o no al ser probado. La métrica escogida se nombra: "Localidad de la modificación" siendo la misma empleada en CALISOF (7), y responde a las preguntas: ¿Puede el usuario operar el sistema de software sin fallos después de su mantenimiento? y ¿El mantenedor puede fácilmente reducir las averías causadas por los efectos secundarios de mantenimiento? Se calcula:

X= Sum (T)/N [2]

Donde:

T: es el tiempo empleado en probar con el fin de asegurar si el informe de fallo ha sido o no resuelto.

N: es el número de fallos resueltos.

Para evaluar el sub-atributo de Conformidad de la Mantenibilidad la ISO 9126 propone una métrica capaz de controlar atributos tales como: el número de funciones, ocurrencias de problemas de conformidad, que ocasionan que el software no logre cumplir los estándares, convenciones o regulaciones relativas a la mantenibilidad establecidos, esta se nombra: "Cobertura de satisfacción de elementos de conformidad de la mantenibilidad" y responde a la pregunta: ¿Cómo es la compatibilidad con el mantenimiento del producto para aplicar reglamentos, normas y convenciones? Se calcula:

X=1-A/B [3]

Donde:

A: número de elementos de conformidad que fallaron durante las pruebas.

B: número de elementos de conformidad totales.

Como última métrica para evaluar la mantenibilidad es necesaria una capaz de medir el sub-atributo estabilidad. La ISO 9126 propone 2 métricas de carácter externas, de las cuales fue seleccionada: “Impacto de la localización de la modificación”, que posibilita estimar de forma más sencilla cuán estable se encuentra un sistema luego de ocurrido fallas en él al efectuarse alguna modificación durante el mantenimiento.

La métrica cuenta las ocurrencias de fallos después de los cambios cuando son detectados y modificados conjuntamente y responde a las siguientes interrogantes: ¿Encuentra el usuario fallas después de los cambios de mantenimiento? ¿Puede el mantenedor fácilmente mitigar los fallos causados por los efectos del mantenimiento?

Se calcula de la siguiente manera:

X=A/N [4]

Donde:

A: número de fallos surgidos después de modificaciones realizadas en el mantenimiento.

N: número de fallas resueltas.

El resultado puede interpretarse tomando en cuenta como respuesta más óptima la más cercana a 0.

En resumen para la confección de una aplicación que permita evaluar los sub-atributos de la mantenibilidad mencionados, se han seleccionado cinco métricas externas, una por cada sub-atributo. Se ha tenido en cuenta en la selección el grado de idoneidad de dichas métricas para su adaptación a los procesos del CEIGE y que posibiliten la medición de la mantenibilidad durante la realización de pruebas internas a un software luego de su finalización. El sistema permitirá mediante la evaluación de las métricas externas conocer si el comportamiento de un producto es mantenible en la etapa de pruebas.

1.3.2 Adaptación de las métricas aplicadas en la etapa de pruebas de los productos del CEIGE

Como se ha indicado anteriormente, la Norma ISO/IEC 9126 se ha creado en base a diferentes modelos de calidad ya existentes, se ha perfeccionado a lo largo de los años y ha sido adaptada para su utilización en herramientas informáticas que posibiliten la evaluación de las métricas que propone.

En el presente trabajo para hacer uso de las métricas descritas se realizó un análisis de cada una de estas, aterrizando sus variables a datos obtenidos durante el desarrollo del proceso de pruebas en el laboratorio de calidad del centro, de forma que se ajustaran correctamente en su aplicación y que la obtención de datos fuera satisfactoria, posibilitando evaluar adecuadamente la mantenibilidad del software.

Cabe mencionar que las adaptaciones introducidas son leves cambios que se justifican en el curso del análisis, con ellas no se pretende mejorar o modificar el modelo existente, sino ajustarlo al proceso de pruebas que se desarrolla en el laboratorio de calidad del CEIGE, para la confección de la herramienta que facilite el cálculo de la mantenibilidad.

Se determinó tratar los fallos del software o de documentación como No Conformidades (NC) que presentan los mismos, aplicando el término en las métricas, ya que una NC, según la norma ISO 9000:2005 (3.1.2), es un incumplimiento, una ausencia o fallo en implantar y mantener uno o más requisitos del sistema, así como errores en su documentación.

Las NC tratadas en el laboratorio de calidad del CEIGE de proceder presentan 3 fases, la primera cuando son encontrada en las pruebas exploratorias, se declara como abierta, si esta no es resuelta en una prueba de regresión o iteración se mantiene en estado pendiente y por último es declarada como resuelta.

A continuación se muestran las adaptaciones realizadas al aplicar las métricas seleccionadas en el epígrafe anterior:

Métrica	Atributo de la Mantenibilidad	Variable	Descripción de la variable	Descripción de la adaptación
“Capacidad de Fallos”	Analizabilidad	A	Número de fallos cuyas causas no han sido encontradas.	Cantidad de NC pendientes.
		B	Número total de fallos registrados.	Cantidad total de NC registradas.

"Tiempo empleado en implementar un cambio por el personal de mantenimiento"	Cambiabilidad	Tout	Es el tiempo promedio en que se resuelven los fallos.	Cantidad de sesiones de trabajo en las que se resuelven las NC por los responsables del proyecto.
		Tin	Es el tiempo promedio en el que se encuentran los fallos.	Cantidad de sesiones de trabajo en las que se encuentran las NC.
		N	Número de fallos registrados y eliminados.	Cantidad de NC resueltas.
"Localidad de la modificación"	Facilidad de prueba	T	Es el tiempo empleado en probar con el fin de asegurar si el informe de fallo ha sido o no resuelto.	Cantidad de sesiones de trabajo en las que se revisaron la resolución o no de las NC.
		N	Es el número de fallos resueltos.	Cantidad de NC resueltas.

"Cobertura de satisfacción de elementos de conformidad de la mantenibilidad"	Conformidad	A	Número de elementos de conformidad que fallaron durante las pruebas.	Cantidad de NC pendientes.
		B	Número de elementos de conformidad totales.	Cantidad total de NC registradas.
"Impacto de la localización de la modificación"	Estabilidad	A	Número de fallos surgidos después de modificaciones realizadas en el mantenimiento.	Cantidad de NC surgidas en las pruebas de regresión.
		N	Número de fallas resueltas.	Cantidad de NC resueltas.

Tabla 1 Adaptación de métricas seleccionadas.

Los datos o variables necesarias para el cálculo de las métricas podrán ser capturados de una hoja de cálculo donde se registran las NC encontradas sobre un producto durante las diferentes pruebas que se le realicen.

Es importante aclarar que se entiende como mantenimiento en la etapa de pruebas, al proceso de resolución de no conformidades por parte del proyecto cliente, tomando en cuenta las variables descritas en la tabla anterior como elementos primordiales en este proceso.

Las adaptaciones realizadas fueron consultadas con especialistas del laboratorio de calidad del CEIGE y del Centro Nacional de Calidad de Software (CALISOFT) como: las especialistas de calidad

del software de la dirección de CALISOFT Daniuska Fresneda Cruzata y Lisandra Díaz Figueredo, además de ser sugeridas y aceptadas por los clientes.

1.4 Tendencias estudiadas

Tendencias internacionales:

De acuerdo con la lectura de la norma ISO 9126 se han concluido una serie de palabras claves utilizadas para realizar la búsqueda de herramientas informáticas que sean capaces de medir la mantenibilidad en Internet particularmente en la web Sourceforge[1], sitio que alberga actualmente la mayor cantidad de proyectos de código abierto sobre otros como: Codeplex, Google Code o Kenai. En particular, apoya a más de 260.000 proyectos de software con una comunidad de más de 2.7 millones de usuarios registrados.

En las búsquedas se han tenido en cuenta los lenguajes de programación más populares, identificándose herramientas asociadas con los lenguajes .NET (tanto en VB.NET y C #), Java y C/C++, ya que estos son los marcos de desarrollo más populares de acuerdo con el índice de la Comunidad de Programación TIOBE[2] (indicador de la popularidad de los lenguajes de programación). Durante la investigación se han considerado las herramientas de análisis estático de código de software libre y se incluyen herramientas que miden el acoplamiento y la complejidad ciclomática, ya que han sido ampliamente reconocidos como los indicadores más valiosos para evaluar la capacidad de mantenimiento de sistemas orientados a objetos.

Entre las herramientas encontradas resaltan CodeProAnalytix, SourceMonitor y JDepend como las más representativas de las métricas de mantenibilidad, aunque hayan podido ser incluidas otras similares. Estas herramientas tienen en cuenta distintos sub-atributos de la mantenibilidad.

Específicamente JDepend es una herramienta que recorre el directorio de clases Java y genera métricas de calidad de diseño para cada uno de los paquetes. JDepend permite medir automáticamente la calidad del diseño en términos de extensibilidad, reusabilidad y mantenibilidad

[1] Web de proyectos de código abierto Sourceforge: www.sourceforge.net

[2] Comunidad de Programación TIOBE: www.tiobe.com/index.php/content/paperinfo/tpci/index.html

para gestionar efectivamente las dependencias entre paquetes. Puede utilizarse para medir sub-atributos como la analizabilidad, cambiabilidad y estabilidad de la mantenibilidad. Esta puede encontrarse además como un plugin del IDE Eclipse, pero solo se restringe al diseño, además de utilizar métricas de carácter interno, que no controlan las acciones relacionadas con el esfuerzo del mantenedor o el usuario cuando se intenta probar el software (26).

CodeProAnalytix es un plugin que evalúa el código fuente escrito en lenguaje Java, es uno de los más completos en cuanto a la cantidad de métricas básicas que obtiene. Es la principal herramienta de prueba de software Java para desarrolladores de Eclipse que quieren ser participantes activos en la mejora de la calidad y la seguridad del código que producen (26).

CodeProAnalytix se integra perfectamente en el entorno de Eclipse, usando el análisis automatizado de código fuente para detectar problemas de calidad y las vulnerabilidades de seguridad antes de que el código alcance la fase de control de calidad. Nos encontramos ante una herramienta de pago. La herramienta no admite la opción de comparar dos versiones de un mismo código ni cargándolas en Eclipse ni a través de un repositorio de versiones.

SourceMonitor es una herramienta para el cálculo de las métricas recopilándolas solo a través de archivos de código fuente escrito en C + +, C, C #, VB.NET, Java, Delphi, Visual Basic (VB6) o HTML. Ofrece la opción de modificación de complejidad métrica. Es gratis y fácil de usar. Exporta las métricas a XML o CSV (valores separados por comas) para su procesamiento posterior con otras herramientas. Esta herramienta gestiona el cálculo de métricas pero no se centra específicamente en la evaluación de la mantenibilidad de los sistemas informáticos (26).

Las herramientas mencionadas contienen varias funcionalidades específicas para procesos que se desarrollan en diferentes organizaciones o bien constituyen plugins integrados a IDEs de programación, generalmente son muy costosas y no miden específicamente métricas externas de la mantenibilidad, por esta razón en la presente investigación se desarrolla una herramienta que se adapte al proceso.

Tendencias nacionales:

En la UCI se encuentra ubicado el Centro Nacional de Calidad de Software (CALISOFT), organización enfocada a contribuir al desarrollo de la Industria Cubana de Software (InCuSoft). En CALISOFT se

encuentra el Laboratorio Industrial de Pruebas de Software (LIPS), fuente de muchas investigaciones sobre la calidad de software. Una de ellas es la “Confección de un catálogo automatizado de métricas de calidad para evaluar los productos en las pruebas” (8), esta propone y desarrolla una herramienta capaz de gestionar y evaluar métricas guiadas según el modelo de calidad ISO 9126-1:2005. Tiene en cuenta muchas características de la calidad que permiten obtener resultados del comportamiento de sistemas, sin embargo específicamente una de ellas, tal como la mantenibilidad, no es tenida en cuenta debido a que no se han definido tipos de pruebas a realizar durante la liberación de un producto en CALISOF, ya que se necesita para ello un estudio a fondo de la forma en que se puede comprobar si las aplicaciones informáticas cumplen con los elementos esenciales de ser mantenibles en el tiempo.

Otra investigación realizada en el mismo centro se basa en una “Propuesta de métricas en el laboratorio de pruebas para evaluar la mantenibilidad de un software.” (7), pero no muestra como resultado una herramienta encargada de la evaluación de las métricas que propone, lo que refuerza el motivo de la presente investigación.

1.5 Modelo de desarrollo de software.

Para el desarrollo de cualquier producto de software se realizan una serie de tareas entre la idea inicial y el producto final. Este proceso de desarrollo se hace muy difícil de controlar sin un modelo que establezca el orden en el que se realizarán las actividades planificadas para la elaboración del proyecto, por lo cual se hace imprescindible apoyarse en uno de ellos para llevar a cabo la construcción de un producto.

Para la realización de este trabajo se ha seleccionado el modelo propuesto por la Subdirección de Producción del CEIGE, denominado Modelo de Desarrollo del CEIGE, el cual describe una secuencia de actividades para desarrollar soluciones de software mostrando los artefactos correspondientes a medida que se vaya construyendo el sistema.

Lenguajes y herramientas para el modelado

Lenguaje unificado de Modelado (UML)

El Lenguaje de Modelado (UML) permite la especificación, visualización, construcción y documentación de los artefactos de un proceso de sistema intensivo, siendo utilizado para la modelación de sistemas, con características orientado a objetos. Facilita a los integrantes de un

equipo multidisciplinario participar e intercomunicarse fácilmente, promueve la reutilización e incrementa la capacidad de lo que se puede hacer con otros métodos de análisis y diseño orientados a objetos. (9)

Para el desarrollo de la presente investigación se decidió utilizar UML como lenguaje de modelado por las comodidades y facilidades de comunicación que brinda a los integrantes de un equipo en la confección de la documentación necesaria para el desarrollo del software, facilitando la creación de los artefactos generados por el proyecto.

CASE

Existen varias herramientas para el modelado UML, entre ellas las de Ingeniería de Software Asistida por Ordenador (Computer Aided Software Engineering, CASE por sus siglas en inglés), sistemas que intentan proporcionar ayuda automatizada a las actividades del proceso de software. Estas herramientas tienen vital importancia en múltiples aspectos del ciclo de vida del desarrollo del software, como son: la mejora de la productividad en el mantenimiento y desarrollo del software, la mejora del tiempo, el costo de desarrollo y el mantenimiento de los sistemas informáticos, la implementación del código con el diseño dado, el aumento de la calidad del software y documentación o detección de errores. (10)

Visual Paradigm 8.0 for UML

Entre las herramientas CASE para el modelado encontramos al Visual Paradigm, cuyo objetivo es el diseño UML, creada para ayudar al desarrollo de software. Es compatible con los equipos de desarrollo en la captura de requisitos, en los software de planificación (el análisis de casos), el código de la ingeniería, modelado de clases y modelado de datos. Ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas. Esta herramienta está enfocada al negocio que genera un software de mayor calidad, es fácil de instalar y actualizar, presenta la posibilidad de generar código para el lenguaje Java y utiliza un lenguaje estándar común para todo el equipo de desarrollo que facilita la comunicación. (10)

1.6 Patrones de Diseño

En la actualidad es muy común el uso de patrones en el desarrollo del software, los mismos proveen a los equipos de desarrollo de un mecanismo que les permitirá diseñar aplicaciones de alta calidad en el

menor tiempo posible, debido a que constituyen soluciones a problemas específicos del diseño, promueven la reutilización del código y la agilización del proceso de desarrollo de software.

Un patrón de diseño es una solución a un problema de diseño. Se consideran como procedimientos para solucionar diversos problemas del mismo tipo y son la base para la búsqueda de soluciones a dificultades comunes en el desarrollo de software. A continuación se especifican algunos patrones de diseño:

Los patrones de diseño se dividen en dos grupos principales, los patrones GRASP (patrones para la asignación de responsabilidades) y los patrones GOF, estos últimos se clasifican de la siguiente forma (11):

Patrones Creacionales: inicialización y configuración de objetos.

Patrones Estructurales: separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan para formar estructuras más grandes.

Patrones de Comportamiento: describen la comunicación entre objetos o clases.

Para garantizar un diseño claro y simple que cumpla con los requerimientos de la herramienta informática que posibilitará el cálculo de la mantenibilidad en el laboratorio de calidad del CEIGE, se emplearon los siguientes patrones:

Patrones GRASP empleados:

Controlador

El patrón Controlador aumenta el potencial de reutilización y asegura que la lógica de la aplicación no se maneje en la capa de presentación. Un controlador es un objeto que no pertenece a la interfaz de usuario y es responsable de recibir o manejar un evento del sistema.

El desempeño de este patrón consiste en asignar la responsabilidad de controlar el flujo de eventos del sistema a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad, entre otras). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión.

Experto

Consiste en asignar una responsabilidad al experto en información, o sea, a la clase que tiene la información necesaria para realizar la responsabilidad. Un modelo de diseño podría definir miles de clases y una aplicación podría requerir que se realicen gran cantidad de responsabilidades. Si se aplican de la forma correcta, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, existiendo más oportunidades para reutilizar componentes en futuras aplicaciones.

Con el uso de este patrón se mantiene el encapsulamiento de la información, puesto que los objetos utilizan su propia información para llevar a cabo las tareas.

Creador

El patrón creador nos ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. Una de las consecuencias de usar este patrón es la visibilidad entre la clase creada y la clase creador. Una ventaja es el bajo acoplamiento, lo cual supone facilidad de mantenimiento y reutilización. La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien, el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización.

Patrón GOF empleado:

Estrategia

El patrón estrategia es un patrón de comportamiento que trata sobre algoritmos y la asignación de responsabilidades entre objetos y no solo describe la comunicación entre ellos. Este permite que el diseñador se concentre sólo en cómo interconectar objetos.

El patrón estrategia tiene como propósito definir una familia de algoritmos, encapsula cada uno de ellos, y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que los usan (12).

1.7 Ambiente de desarrollo

1.7.1 Lenguajes de Programación

Java

Es un lenguaje de programación muy utilizado en cuanto al desarrollo de herramientas y aplicaciones de negocio. Es rápido, seguro, fiable, su distribución es gratuita, brinda soporte a las técnicas de desarrollo de la Programación Orientado a Objetos (POO) y en resumen a la reutilización de

componentes de software. Otro de los factores más importantes de JAVA es que no se quiebra fácilmente ante errores de programación, no es posible escribir en áreas arbitrarias de memoria ni realizar operaciones que corrompan el código. Puede aplicarse a la realización de aplicaciones en las que ocurra más de una cosa a la vez lo que se denomina multihilos. (13)

JAVA es un lenguaje multiplataforma, ya que funciona en cualquier sistema operativo que tenga instalada la máquina virtual para este lenguaje. Además brinda mucha seguridad, pues su máquina virtual al ejecutar el código, realiza comprobaciones de seguridad y carece de características poco seguras como los punteros. Por estas razones se decide utilizar JAVA como lenguaje de programación.

SQL

Structured Query Language es un lenguaje estándar de comunicación con bases de datos. Por tanto es un lenguaje normalizado que permite trabajar con cualquier otro tipo de lenguaje (ASP o PHP) en combinación con cualquier tipo de base de datos (MS Access, SQL Server, MySQL...). Además de esta universalidad, el SQL posee otras dos características muy apreciadas. Por una parte, presenta una potencia y versatilidad notables que contrasta, por otra, con su accesibilidad de aprendizaje. El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos. (14)

1.7.2 Herramientas de desarrollo

IDE

Un entorno de desarrollo integrado, llamado también IDE (siglas en inglés *Integrated Development de Environment*) es un entorno de programación que ha sido empaquetado como un programa de aplicación, o sea, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solos o pueden ser parte de aplicaciones existentes, estos ofrecen un marco de trabajo amigable para la mayoría de los lenguajes de programación, también son multiplataformas, depuradores y reconocedores de sintaxis. (15)

NetBeans 7.1

Es una herramienta que permite rápida y fácilmente desarrollar aplicaciones en el lenguaje Java para escritorio, móviles y o la web, es gratuito y de código abierto. Tiene una gran comunidad de usuarios y desarrolladores en todo el mundo. Proporciona soporte y complementos para las últimas tecnologías

Java, así como mejoras con anticipación a otros IDE. Presenta una buena organización del código y facilita diferentes vistas de los datos. Además *NetBeans* brinda herramientas de análisis estático, y contiene un depurador que permite colocar puntos de interrupción en el código fuente, añadir relojes de campo, muestra cada paso a través del código dirigido a los métodos, permite tomar instantáneas de la ejecución. Puede ser instalado con comodidad en cualquier sistema operativo y es capaz de trabajar sin presentar problemas con aplicaciones que realicen varias tareas a la misma vez. (16)

PostgreSQL 9.1

PostgreSQL es un sistema gestor de base de datos objeto-relacional, bajo licencia BSD. La licencia BSD al contrario que la GPL permite el uso del código fuente en software no libre. Es el sistema de gestión de bases de datos de código abierto más avanzado del mundo y en sus últimas versiones posee muchas características que solo se podían ver en productos comerciales de alto calibre.

PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando.

Se ejecuta en casi todos los principales sistemas operativos: *Linux*, *Unix*, *BSDs*, *Mac OS*, *Beos*, y *Windows*. Posee una documentación muy bien organizada, pública y libre, con comentarios de los propios usuarios. Brinda soporte nativo para los lenguajes más populares del medio: PHP, C, C++, Perl, Python, JAVA, entre otros. Además tolera todas las características de una base de datos profesional (triggers, store procedures, funciones, secuencias, relaciones, reglas, tipos de datos definidos por usuarios, vistas y vistas materializadas). Es por tanto altamente adaptable a las necesidades del cliente. (17)

1.8 Conclusiones

En este capítulo, luego de definirse algunos conceptos necesarios para el entendimiento de la investigación científica en cuestión, la realización de un estudio de sistemas informáticos que tuviesen en cuenta la evaluación de la mantenibilidad del software, la selección de las tecnologías, los lenguajes y las herramientas a utilizar en la implementación; así como el análisis y adaptación de las métricas externas escogidas para la confección de un sistema capaz de medir la mantenibilidad, se arriba a las siguientes conclusiones:

- Los sistemas expuestos no cumplen con todo los parámetros para evaluar de una forma más íntegra la mantenibilidad del software.
- Los sistemas están desarrollados bajo condiciones diferentes, o como plugins integrados a un IDE de programación y no como una herramienta especializada.
- La evaluación de la mantenibilidad actualmente en la UCI se realiza de forma manual.
- Las aplicaciones estudiadas están implementadas bajo tecnologías o herramientas propietarias y solamente evalúan métricas internas de la mantenibilidad.
- Las métricas externas seleccionadas fueron sometidas a adaptaciones que permitieran la obtención de resultados aterrizados al centro.

Por tanto se determinó la necesidad de crear una solución propia que responda a las características particulares del CEIGE y que posibilite la evaluación de la mantenibilidad de los productos en la etapa de pruebas.

Capítulo 2: Análisis y diseño del sistema

2.1 Introducción

En este capítulo se presentan los primeros flujos de trabajo propuestos por el Modelo de Desarrollo de Software del CEIGE para la confección de la solución propuesta: Requisitos, Análisis y Diseño. En la etapa de Requisitos se identifican y describen los requisitos funcionales y no funcionales. En el Análisis y el Diseño, se definen todos los artefactos propuestos por el modelo para cada caso. Igualmente, se especifican los patrones de diseño utilizados en la concepción de la solución.

2.2 Propuesta del sistema

Se propone la confección de una herramienta que posibilite medir el comportamiento de la mantenibilidad utilizando métricas de carácter externas que faciliten determinar el estado del software en relación a este atributo de calidad durante la realización de pruebas o el mantenimiento de un sistema luego de su terminación. En el análisis de dichas métricas se utiliza una escala cuantitativa para determinar e interpretar el estado de una aplicación informática según una escala del 0 al 1 como: no mantenible (0 - 0.3), poco mantenible (0.4 - 0.6) y mantenible (0.7 - 1).

Modelo de Dominio

El modelo de dominio explica las nociones más significativas del problema. Muestra: conceptos, asociaciones entre estos y sus atributos. Una cualidad esencial que este debe ofrecer es la de representar claramente elementos del mundo real. En proyectos pequeños el análisis llega a resumirse en el modelo de dominio al no tenerse casos de uso y/o procesos, por tanto se ajusta y utiliza en la realización de este trabajo.

El modelo de dominio mostrado en la figura 2 explica que la mantenibilidad está conformada por cinco sub-atributos, los que en conjunto permiten evaluar el grado de mantenibilidad de un producto.

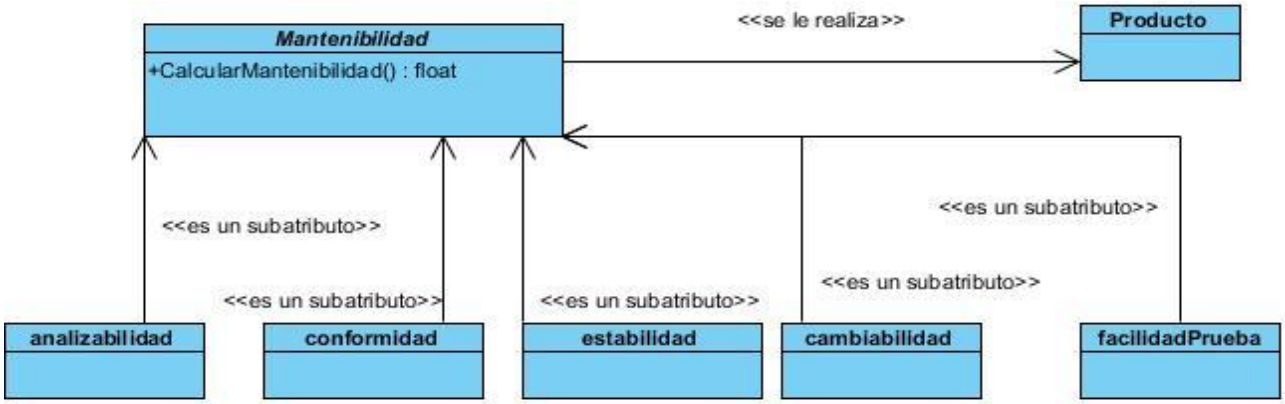

Ilustración 2 Modelo de dominio.

2.3 Especificación de los requisitos de software

En este flujo de trabajo se analizan los aspectos correspondientes a las conciliaciones del cliente y los directivos del proyecto con relación a las funcionalidades que el sistema debe permitir y las que debe cumplir. El propósito fundamental de este flujo de trabajo es guiar el desarrollo hacia un sistema más completo. Esto se consigue mediante una descripción de los requisitos del sistema lo suficientemente acertada como para que pueda llegarse a un entendimiento mutuo entre el cliente y los desarrolladores sobre qué debe y qué no debe hacer el sistema.

La captura de requisitos tiene gran importancia en el proceso de desarrollo del software, ya que a través de estos se identifica lo que se desea y de esta forma se obtiene un producto con calidad. El levantamiento de los requisitos puede realizarse teniendo en cuenta dos conceptos: por escenario y/o por casos de uso.

Para llevar a cabo el sistema que motiva la presente investigación, se decidió realizar la captura de requisitos mediante el concepto de escenario, ya que este describe un ejemplo del uso de la aplicación en términos de una serie de interacciones entre el usuario y la aplicación, además de ser una descripción concreta enfocada a una característica del software. Su actividad fundamental es la identificación de escenarios que describen narrativamente cómo las personas podrían interactuar con el sistema. (18)

Los requisitos se pueden clasificar en:

- ✓ Funcionales.
- ✓ No Funcionales.

Requisitos funcionales:

Los requisitos funcionales no son más que capacidades o condiciones que el sistema debe cumplir, estos especifican comportamientos particulares de un sistema. En la presente investigación se identificaron varios requisitos que darán solución a la problemática planteada, estos son:

- **RF1- Buscar producto**

El sistema facilitará hacer una búsqueda al introducir las iniciales de un producto para realizarle el cálculo de mantenibilidad.

- **RF2- Listar productos registrados**

El sistema con esta funcionalidad posibilita obtener un listado de todos los productos que se encuentran almacenados en la base de datos, a los cuales se le ha realizado el cálculo de la mantenibilidad.

- **RF3- Adicionar producto**

De no encontrarse un producto en la base de datos del sistema, este permitirá automáticamente adicionarlo para realizar posteriormente el cálculo de la mantenibilidad. Se introducen datos como: el nombre del producto, sus iniciales y la fecha del cálculo de la mantenibilidad.

- **RF4- Cargar datos de un producto**

El sistema permitirá cargar los datos de un producto para cada sub-atributo de la mantenibilidad desde un Excel y posteriormente ser evaluados.

- **RF5- Calcular analizabilidad**

Luego de adicionado o encontrado un producto en la base de datos, el sistema posibilitará habilitar esta funcionalidad, para efectuar el cálculo del sub-atributo analizabilidad de la mantenibilidad de un producto introduciendo los valores: cantidad de no conformidades (NC) pendientes y cantidad total de NC registradas.

- **RF6- Calcular cambiabilidad**

Luego de adicionado o encontrado un producto en la base de datos, el sistema posibilitará habilitar esta funcionalidad, para efectuar el cálculo del sub-atributo cambiabilidad de la mantenibilidad de un producto introduciendo los valores: cantidad de sesiones de trabajo en la que son resueltas las NC por

los responsables del producto, cantidad de sesiones de trabajo en las que se detectan las NC y la cantidad total de NC resueltas.

- **RF7- Calcular conformidad**

Luego de adicionado o encontrado un producto en la base de datos, el sistema posibilitará habilitar esta funcionalidad, para efectuar el cálculo del sub-atributo conformidad de la mantenibilidad de un producto introduciendo los valores: cantidad de NC pendientes y cantidad total de NC registradas.

- **RF8- Calcular facilidad de prueba**

Luego de adicionado o encontrado un producto en la base de datos, el sistema posibilitará habilitar esta funcionalidad, para efectuar el cálculo del sub-atributo facilidad de prueba de la mantenibilidad de un producto introduciendo los valores: cantidad de sesiones de trabajo en las se revisaron la resolución o no de las NC y cantidad de NC resueltas.

- **RF9- Calcular estabilidad**

Luego de adicionado o encontrado un producto en la base de datos, el sistema posibilitará habilitar esta funcionalidad, para efectuar el cálculo del sub-atributo estabilidad de la mantenibilidad de un producto introduciendo los valores: cantidad de NC encontradas en las pruebas de regresión y cantidad total de NC registradas.

- **RF10- Mostrar resultado final**

Luego de realizar el cálculo de todos los sub-atributos de la mantenibilidad, esta funcionalidad posibilitará conocer el resultado final de la mantenibilidad de un producto.

- **RF11- Generar reporte**

Luego de haber realizado el cálculo de la mantenibilidad del sistema se habilitará esta funcionalidad, que permitirá generar un reporte en formato pdf con los resultados de los cálculos efectuados para cada uno de los sub-atributos de la mantenibilidad, su interpretación, así como la gráfica correspondiente que los represente.

- **RF12- Graficar resultados**

Luego de haber realizado el cálculo de la mantenibilidad del sistema se habilitará esta funcionalidad que permitirá mostrar una gráfica representativa del resultado de los cálculos efectuados, lo que posibilitará tener una idea general del estado de la mantenibilidad del software evaluado.

Requisitos no funcionales:

Los requisitos no funcionales son aquellos que no incluyen una relación directa con el comportamiento funcional del sistema, sino con las propiedades que este debe tener. Describen aspectos del software que generalmente pueden ser visibles por el usuario (19) (20). Para la confección de la aplicación se identificaron los siguientes requisitos no funcionales:

- **RNF1- Usabilidad**

El sistema debe ser fácil de utilizar para los usuarios que tengan niveles básicos de computación. No se utilizarán textos extensos para las etiquetas de la interfaz de usuario.

- **RNF2- Portabilidad**

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

- **RNF3- Hardware**

Los requerimientos mínimos necesarios para el correcto funcionamiento de la aplicación son los siguientes:

- Memoria RAM de 1GB en adelante.

- Disco duro de 60GB o superior.

- **RNF4- Software**

El lenguaje de programación a utilizar será Java, como herramienta de desarrollo se utilizará el NetBeans en su versión 7.1 y para garantizar la gestión de los datos el PostgreSQL 9.1.

- **RNF5- Funcionalidad**

El sistema mostrará los errores en forma de mensajes. Todos los mensajes de error del sistema deberán incluir una descripción textual del mismo.

2.4 Validación de requisitos

Esta actividad se realizó con el objetivo de garantizar que los requisitos fueran correctos y cumplieran con las necesidades del cliente. Se desarrolló mediante las siguientes técnicas:

- ✓ Validación de requisitos mediante prototipos y escenarios.

Se presentaron los prototipos elaborados durante la especificación a especialistas funcionales, para corroborar su correspondencia con las necesidades y aspiraciones del cliente. Para ello, se desarrollaron varios escenarios posibles con el auxilio de juegos de datos, de forma tal que se visualizaran las diferentes funcionalidades que tendría el sistema. Las no conformidades fueron documentadas y corregidas.

- ✓ Revisión técnica de artefactos.

Se realizaron revisiones de los artefactos por parte del equipo de calidad del proyecto y fueron corregidas las no conformidades identificadas para la liberación de la documentación.

2.5 Análisis y Diseño

Para que el diseño, la implementación, la gestión de la información y el conocimiento de un proyecto sea adecuado y eficaz, el análisis es clave, pues permite estudiar, interpretar y comprender con mayor facilidad y exactitud los requisitos descritos con anterioridad en su captura.

La mayor diferencia existente entre la captura de requisitos y el análisis está dada en que el primero se realiza utilizando el lenguaje del cliente, mientras que el segundo se basa fundamentalmente en el lenguaje de los desarrolladores. (21)

Por su parte en el diseño se determina la arquitectura y comportamiento general del sistema, adaptando las especificaciones realizada en la etapa anterior. En esta fase se establece cómo el software debe reaccionar ante diferentes acontecimientos. El resultado obtenido de la etapa de diseño facilita enormemente la implementación posterior del sistema, proporcionando su estructura básica, así como los diferentes componentes que actúan y se relacionan entre sí. (22)

2.5.1 Diagrama de clases del Diseño

Un diagrama de clases es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema y los componentes que se encargarán del funcionamiento y la relación entre uno y otro. El diagrama de clases de diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. En la figura 3 se muestra la estructura y relaciones entre clases que conforman la solución propuesta.

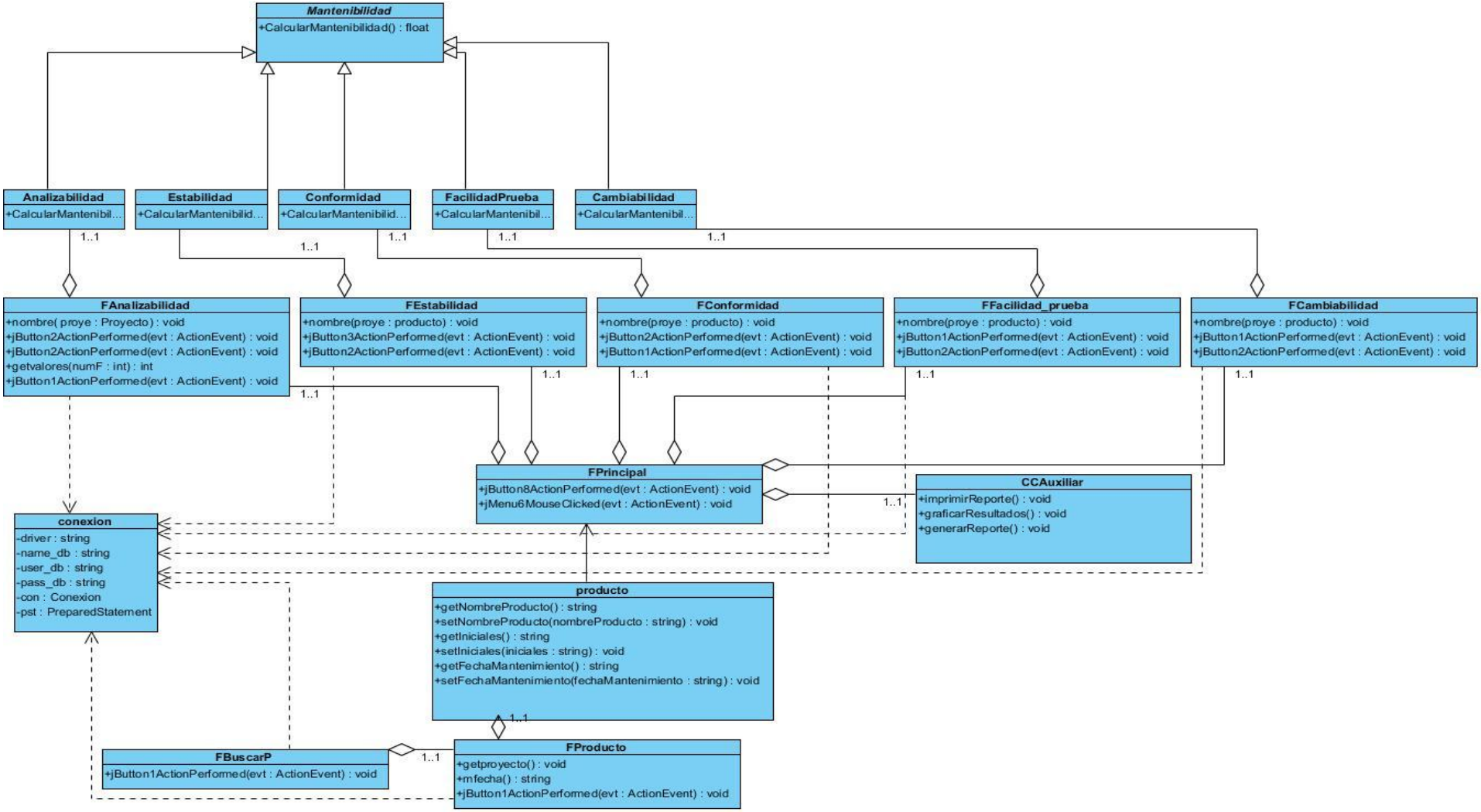

Ilustración 3 Diagrama de clases del Diseño.

2.5.2 Diagrama de secuencia

Para representar gráficamente cómo las clases y subsistemas interactúan para llevar a cabo las funcionalidades del sistema, se confeccionan los diagramas de secuencia.

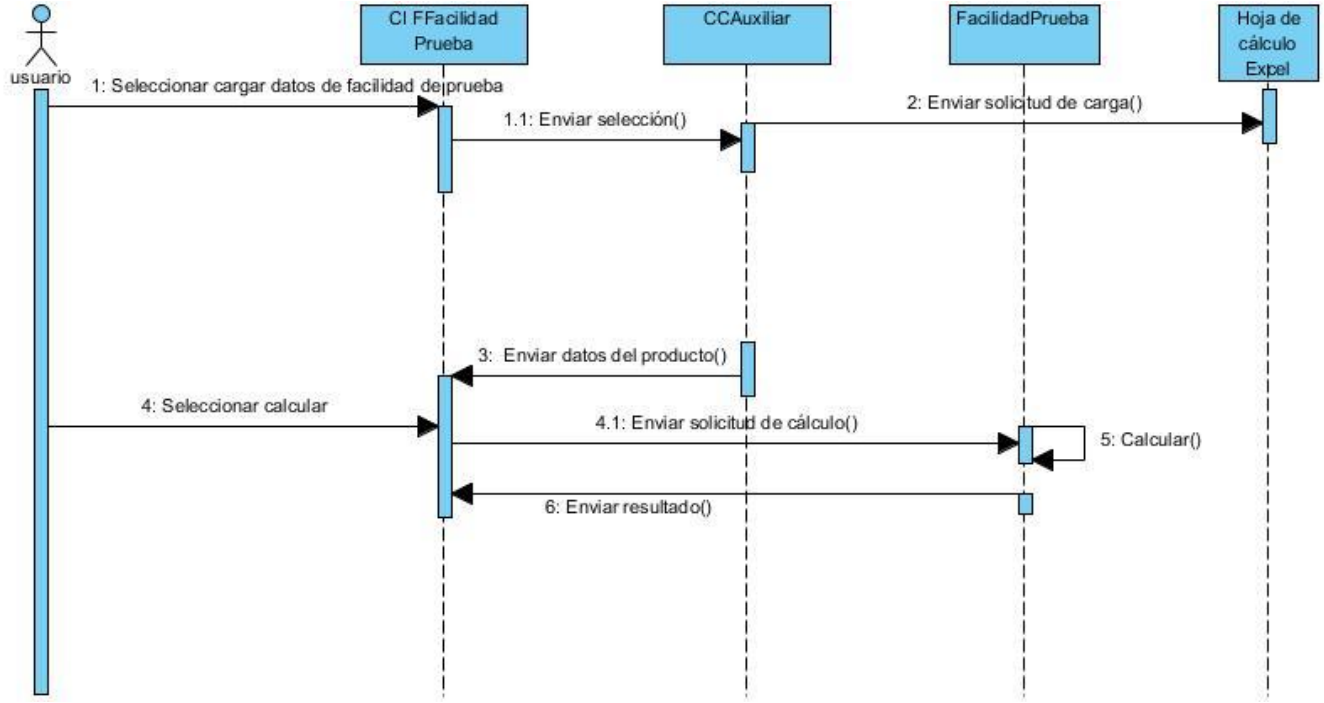

Ilustración 4 Diagrama de secuencia de facilidad de prueba.

2.5.3 Patrones de Diseño

El desarrollo de un sistema conlleva, en la mayoría de las ocasiones, a darle solución a problemas muy complejos que ya alguien ha resuelto con anterioridad. Por esta razón uno de los pasos a tener en cuenta cuando se decide desarrollar un proyecto de software es identificar qué patrones pueden ser utilizados. Entiéndase por patrón como una solución estándar para un problema común de programación. A continuación se muestran los patrones aplicados en la confección de la herramienta para evaluar la mantenibilidad en el laboratorio de calidad del CEIGE:

Durante el diseño del componente se emplearon patrones GRASP, específicamente:

✓ **Controlador:** la clase del visual Principal, constituye un ejemplo de la aplicación de este patrón, debido a que en esta se coordina cada una de las responsabilidades de los objetos.

✓ **Experto:** se evidencia en la definición de las clases de acuerdo con las funcionalidades que deben realizar a partir de la información que manejan. Específicamente las clases: Mantenibilidad, Analizabilidad, Cambiabilidad, Conformidad, Facilidad Prueba y Estabilidad, serán las responsables de efectuar las operaciones correspondientes al cálculo de los sub-atributos de la mantenibilidad.

✓ **Creador:** el patrón creador nos ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases, este patrón se pone de manifiesto en la clase del visual FProducto, ya que esta realiza una instancia de la clase Producto.

Además en el diseño fue tomado en cuenta el siguiente patrón GOF:

✓ **Estrategia:** el patrón de comportamiento estrategia se basa en la asignación de responsabilidades entre objetos, lo que se pone de manifiesto en la herencia entre la clase Mantenibilidad y sus hijos.

2.5.4 Prototipo de interfaz

Un prototipo es una visión preliminar del sistema futuro, es un modelo operable, fácilmente ampliable y modificable, que tiene todas las características que hasta el momento debe tener el sistema.

Las ventajas principales de los prototipos son:

- ✓ Posibilidad de cambiar el modelo.
- ✓ Oportunidad para suspender el desarrollo del modelo si no es funcional.
- ✓ Posibilidad de crear un nuevo modelo que se ajuste mejor a las necesidades y expectativas del usuario.

Para el desarrollo de la aplicación se ha decidido crear una interfaz de usuario amigable y sencilla que permita al usuario interactuar con el sistema sin necesitar un entrenamiento profundo.

A continuación se muestran los prototipos de dos interfaces del software, la interfaz principal y la correspondiente al cálculo de la analizabilidad.

Ilustración 5 Prototipo de la Interfaz Principal del Sistema.

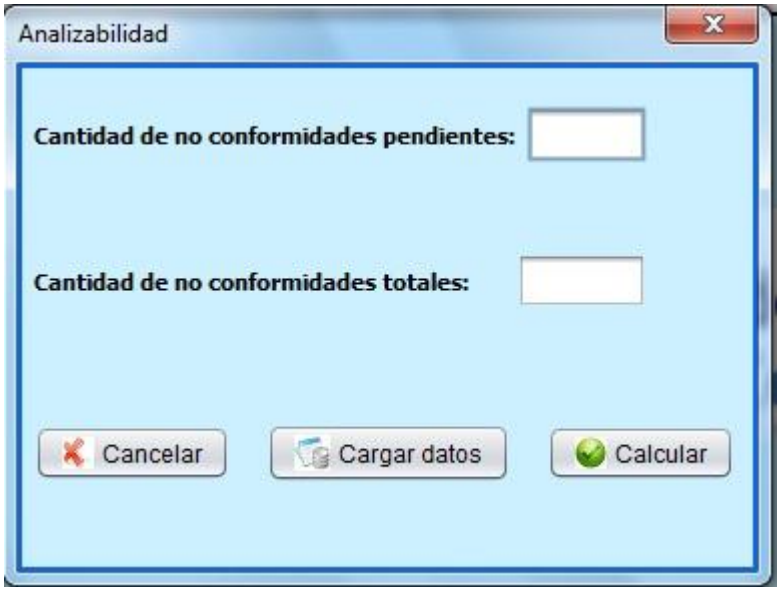

Ilustración 6 Prototipo de la Interfaz para el Cálculo de la Mantenibilidad.

2.5.5 Modelo de datos

Los modelos de datos aportan la base conceptual para diseñar aplicaciones que hacen un uso intensivo de datos, así como la base formal para las herramientas y técnicas empleadas en el

desarrollo y uso de sistemas de información, permiten además, describir los elementos de la realidad que intervienen en un problema dado y la forma en que se relacionan esos elementos entre sí.

Un modelo de datos es por tanto una colección de conceptos bien definidos matemáticamente que ayudan a expresar las propiedades estáticas y dinámicas de una aplicación con un uso de datos intensivo. A continuación se muestra el modelo de datos del sistema propuesto.

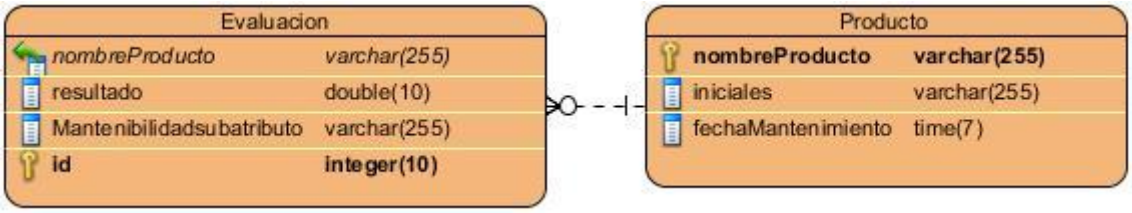

Ilustración 7 Modelo de datos.

2.5.6 Diagrama de clases persistentes

Las clases persistentes son aquellas clases que tienen un mayor tiempo de vida, más allá del tiempo de ejecución de la aplicación. Generalmente las clases persistentes coinciden con los conceptos de información que se usan en el sistema. También se les denomina, en ocasiones, clases de dominio. En la imagen 8 se observa el diagrama de clases persistentes de la solución propuesta.

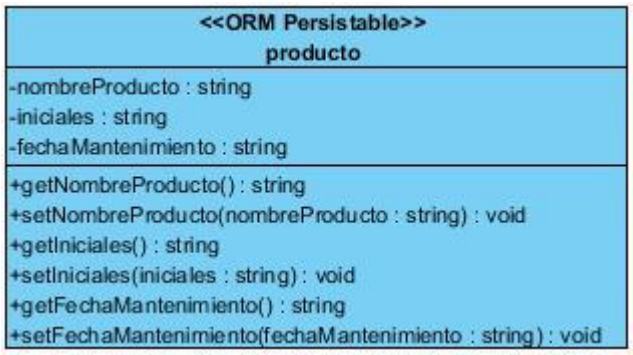

Ilustración 8 Diagrama de clases persistentes.

2.5.7 Métricas de validación del diseño

Las métricas del software permiten medir de forma cuantitativa la calidad de los atributos de un producto, posibilitando al ingeniero evaluar la calidad durante el desarrollo del sistema. Las métricas, específicamente las de diseño a nivel de componentes, se concentran en las características internas

de estos, dando como resultado medidas que pueden ayudar al desarrollador a juzgar la calidad de un diseño a nivel de componentes (22).

Un aspecto importante a tener en cuenta en la evaluación del diseño, es la creación de métricas básicas inspiradas en el estudio de la calidad del diseño orientado a objetos. Los atributos de calidad que se tienen en cuenta son (22):

Responsabilidad: es la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.

Complejidad de implementación: es la complejidad de implementación que posee una estructura de diseño de clases.

Reutilización: es el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

Acoplamiento: es el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.

Complejidad del mantenimiento: es el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.

Cantidad de pruebas: es el número o el grado de esfuerzo para realizar las pruebas de calidad (unidad) del producto (componente, módulo, clase y conjunto de clases) diseñado.

Las métricas seleccionadas como instrumento para medir la calidad del diseño de la herramienta que posibilitará la evaluación de la mantenibilidad en el laboratorio de calidad del CEIGE son las siguientes:

Tamaño Operacional de Clase (TOC): está dado por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad (23):

Atributo de calidad	Modo en que lo afecta
Responsabilidad	Aumento del TOC provoca aumento de la responsabilidad asignada a la clase.

Complejidad de implementación	Aumento del TOC provoca aumento de la complejidad de implementación de la clase.
Reutilización	Aumento del TOC provoca disminución del grado de reutilización de la clase.

Tabla 2 Atributos de calidad evaluados por la métrica TOC (23).

Los criterios y categorías definidos para la evaluación de los atributos de calidad anteriores se presentan en la siguiente tabla:

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

Tabla 3 Criterios de evaluación para la métrica TOC (23).

Relaciones entre Clases (RC): está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad (23):

Atributo de calidad	Modo en que lo afecta
Acoplamiento	Aumento del RC provoca aumento del Acoplamiento de la clase.
Complejidad de mantenimiento	Aumento del RC provoca aumento de la complejidad del mantenimiento de la clase.
Reutilización	Aumento del RC provoca disminución en el grado de reutilización de la clase.

Cantidad de pruebas	Aumento del RC provoca aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 4 Atributos de calidad evaluados por la métrica RC (23).

Los criterios y categorías definidos para la evaluación de los atributos de calidad anteriores se presentan en la siguiente tabla:

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

Tabla 5 Criterios de evaluación para la métrica RC (23).

Resultados obtenidos en la aplicación de la métrica TOC

Luego de aplicarse la métrica de diseño TOC se obtuvieron resultados que permiten evaluar el diseño propuesto de calidad aceptable teniendo en cuenta que el 70% de las clases empleadas en el sistema poseen 6 operaciones o menos, lo que conlleva a evaluaciones positivas de los atributos de calidad involucrados (responsabilidad, complejidad de implementación y reutilización).

Responsabilidad:

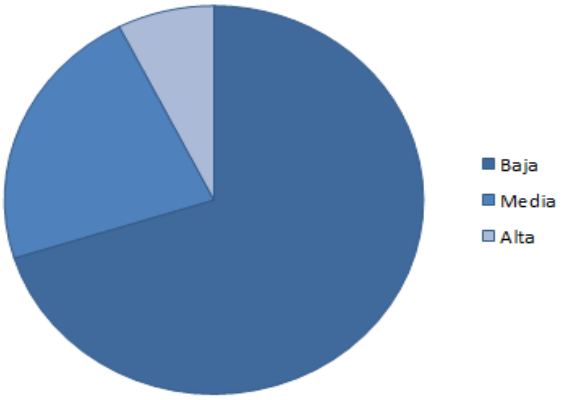

Ilustración 9 Resultados de la evaluación de la métrica TOC para el atributo Responsabilidad.

Complejidad de implementación:

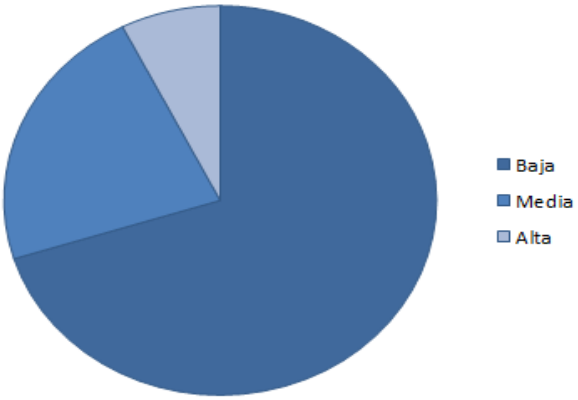

Ilustración 10 Resultados de la evaluación de la métrica TOC para el atributo Complejidad.

Reutilización:

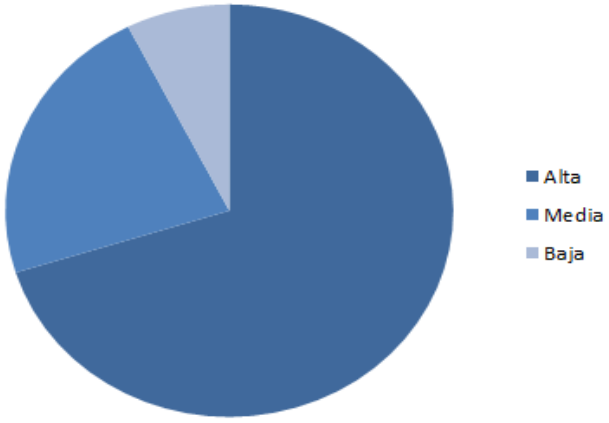

Ilustración 11 Resultados de la evaluación de la métrica TOC para el atributo Reutilización.

Resultados obtenidos de la aplicación de la métrica RC

Luego de aplicarse la métrica de diseño RC se obtuvieron resultados que permiten evaluar el diseño propuesto de calidad aceptable teniendo en cuenta que más del 80% de las clases empleadas en el sistema poseen 1 o menos dependencias con otras clases lo que conlleva a evaluaciones positivas de los atributos de calidad involucrados (acoplamiento, complejidad de mantenimiento, cantidad de pruebas y reutilización).

Acoplamiento:

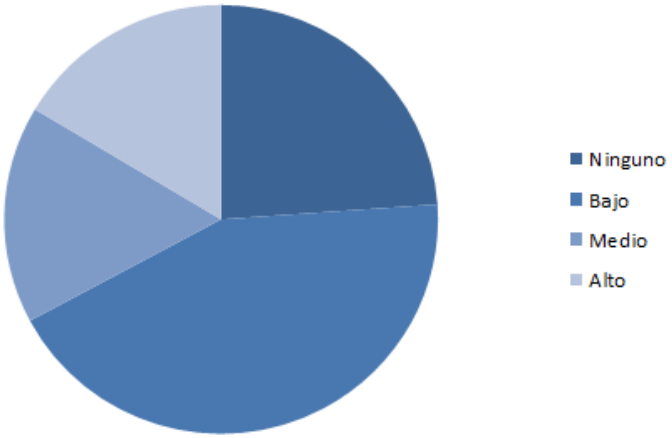

Ilustración 12 Resultados de la evaluación de la métrica RC para el atributo Acoplamiento.

Complejidad de mantenimiento:

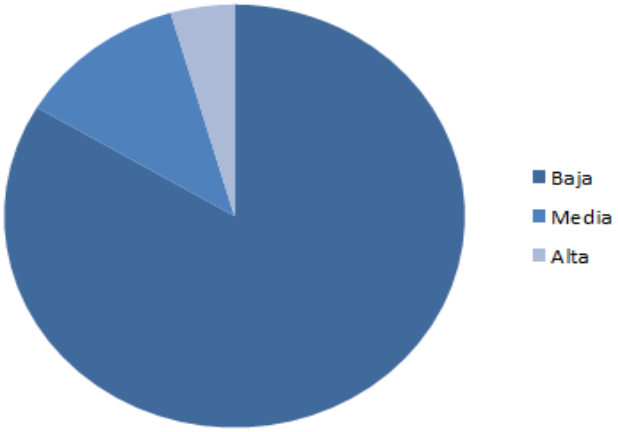

Ilustración 13 Resultados de la evaluación de la métrica RC para el atributo Complejidad de mantenimiento.

Reutilización:

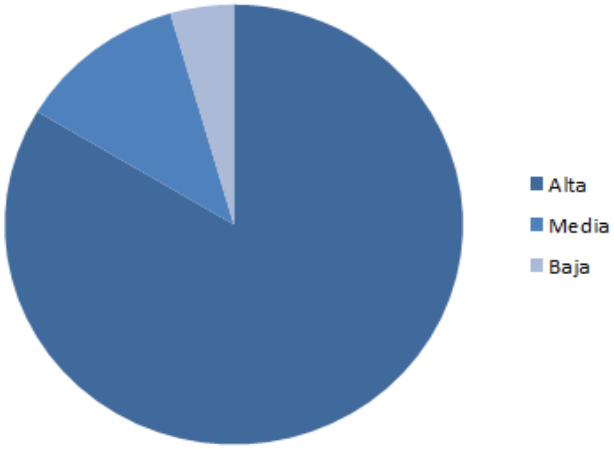

Ilustración 14 Resultados de la evaluación de la métrica RC para el atributo Reutilización.

Cantidad de pruebas:

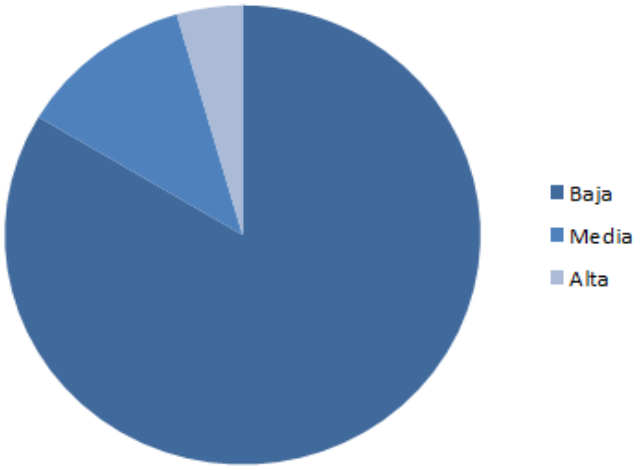

Ilustración 15 Resultados de la evaluación de la métrica RC para el atributo Cantidad de pruebas.

2.5.8 Estándares de Codificación

Convenciones de nombres

En la implementación del sistema los nombres se regirán por la convención del código de Java. Como principio todas las denominaciones serán en español exceptuando aquellas que correspondan al de una librería.

Clases

Los nombres de las clases deberán ser sustantivos, en el caso de ser compuestos tendrán la primera letra de cada palabra que lo forme en mayúscula. Estos deben ser simples y descriptivos. Además, se deben utilizar palabras completas y evitar acrónimos o abreviaturas innecesarias.

Métodos

La denominación de los métodos debe ser un infinitivo demostrando así la acción que ejecutan, cuando sean compuestos tendrán la primera letra en minúscula, y la inicial de las siguientes palabras en mayúscula.

Ejemplo: calcularMantenibilidad().

2.6 Conclusiones

Como resultado de este capítulo se generaron los artefactos del análisis y el diseño del sistema propuesto. El diseño se llevó a cabo mediante la correcta utilización de patrones, que posibilitó la organización del trabajo. Este fue validado a través de métricas que arrojaron resultados satisfactorios.

El trabajo realizado en el transcurso del capítulo sirve de base fundamental para la confección de la herramienta, pues se logra una mayor organización, uniformidad, proporciona rapidez y precisión en el momento de la implementación.

Capítulo 3: Implementación, validación y prueba

3.1 Introducción

En el presente capítulo se describen las principales funcionalidades implementadas además de la realización de un estudio de los diferentes métodos y tipos de pruebas a llevar a cabo, con el objetivo de garantizar la calidad del sistema y el total cumplimiento de los requisitos establecidos con el cliente.

3.2 Aspectos fundamentales de la implementación

En la actualidad el uso de librerías se ha generalizado. La mayoría de los sistemas operativos modernos proporcionan bibliotecas que implementan los servicios del sistema. Como tal, la mayor parte del código utilizado por las aplicaciones modernas se ofrece en estas bibliotecas. De esta forma se reutilizan códigos completamente funcionales, facilitando el trabajo y disminuyendo el tiempo de implementación.

En este epígrafe se explica de forma concisa los métodos o algoritmos más complejos implementados en la confección de la herramienta, que para su funcionamiento hacen inserción de librerías.

El sistema debe permitir realizar diferentes funcionalidades que solo son posibles de ejecutar utilizando librerías javas ya implementadas. Es el caso de la funcionalidad *Generar reporte*, la cual debe posibilitar la creación de un documento en formato pdf que recoja los resultados de la evaluación de la mantenibilidad de un producto, para hacer posible esto se utilizó la librería iText.jar.

La función *generarReporte(Producto ge)* responde al requisito mencionado. El método haciendo un llamado al algoritmo *colocarDestino()* facilita una ruta para guardar un documento que contenga los resultados del cálculo de la mantenibilidad del producto introducido. Esto es posible mediante consultas realizadas a la base de datos y la utilización de la librería iText.jar que permite la generación de un reporte en formato pdf. A continuación se muestra en la figura 16 fragmentos del código que admite la realización de dichas operaciones.

```
public void generarReporte( Producto  ge) {
    Colocar_Destino();        //abre ventana de dialogo "guardar el doc
    ResultSet rs = con.consulta("SELECT * FROM mantenibilidad.evaluacion WHERE nombreproyecto=" + "'" + ge.getNom

    if (this.rutaDestino != null) {
        try {
            Document mipdf = new Document();// se establece una instancia a un documento pdf
            PdfWriter.getInstance(mipdf, new FileOutputStream(this.rutaDestino + ".pdf"));//se utiliza la libreri

            mipdf.open();
            while (rs.next()) {

                mipdf.add(new Paragraph(rs.getString(2)));//adicionando parrafos y tablas
                mipdf.add(new Paragraph(rs.getString(3)));
                mipdf.add(new Paragraph(rs.getString(4)));
            }
            mipdf.close(); //se cierra el PDF
        } catch (SQLException ex) {
            Logger.getLogger(Auxiliar.class.getName()).log(Level.SEVERE, null, ex);
        } catch (FileNotFoundException ex) {
            Logger.getLogger(Auxiliar.class.getName()).log(Level.SEVERE, null, ex);
```

Ilustración 16 Fragmento de código de la función generarReporte(Producto ge).

Otra funcionalidad que necesitó para su implementación de una librería fue *Graficar resultado*, que utilizando jfreechart.jar posibilitó la obtención de un gráfico con los valores calculados en la aplicación de las métricas de la mantenibilidad a un producto.

La función *graficar(Producto nu)* responde a la funcionalidad mencionada. El algoritmo crea un nuevo gráfico en un objeto dataset de tipo DefaultCategoryDataset. Este tipo de dato es proporcionado por la librería jfreechart.jar, para la generación de gráficas. Los datos necesarios para representar la funcionalidad son obtenidos de la base de datos mediante consultas de selección y luego graficados, especificando el esquema que se desea. A continuación se muestra en la figura 17 fragmentos del código que posibilita hacer dichas operaciones.

```
public void Graficar (Producto nu)     {
    try {
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();//data.. es una librería paar ir creando e
        int registros = 0;
        List<Double> sub = new ArrayList<Double>();//(creo la lista)se gurdan los resultados obtenidos de la bas
        ResultSet rs = con.consulta("SELECT * FROM mantenibilidad.evaluacion WHERE nombreproyecto Like " + "'" +
        ResultSetMetaData rsm = rs.getMetaData();
        while (rs.next()) {
            registros++;
            sub.add((double) rs.getFloat("resultados"));//add a al lista
        }
        Iterator<Double> iter = sub.iterator();
        while (iter.hasNext()) {
            dataset.addValue(iter.next(), "Analizabilidad", "columna 1");//add valores para graficar
            dataset.addValue(iter.next(), "Estabilidad", "Columna 2");
            dataset.addValue(iter.next(), "Cambiabilidad", "Columna 3");
            dataset.addValue(iter.next(), "Conformidad", "Columna 4");
            dataset.addValue(iter.next(), "Facilidad de Prueba", "Columna 5");
        }
        JFreeChart chart = ChartFactory.createBarChart3D("Gráfico sobre el mantenimiento del " + nu.getNombrePr
                dataset, PlotOrientation.VERTICAL, true, true, true);
        chart.setBackgroundPaint(new Color(255, 225, 155));
        CategoryPlot plot = chart.getCategoryPlot();
        plot.setBackgroundPaint(Color.lightGray);
        plot.setDomainGridlinesVisible(false);
        plot.setRangeGridlinePaint(Color.BLACK);
        BarRenderer renderer = (BarRenderer) plot.getRenderer();
        GradientPaint gp0 = new GradientPaint(0.0f, 0.0f, new Color(0, 102, 153),
                0.0f, 0.0f, new Color(0, 0, 64));
        renderer.setSeriesPaint(0, gp0);
```

Ilustración 17 Fragmento de código de la función graficar(Producto nu).

Además se utilizaron las librerías Excel-jxl.jar para la captura de datos de un Excel y BD-postgresql-9.1-903.jdbc4.jar para la conexión con la base de datos. Estas se evidencian fundamentalmente en la función *leerArchivoExcel(String archivoDestino, String sub),* que posibilita a través de las librería iText y las mencionadas con anterioridad, capturar valores de los campos de una hoja de cálculo para el sub-atributo de la mantenibilidad introducido, teniendo en cuenta el destino del documento insertado como parámetro y posteriormente guardar los resultados en la base de datos. A continuación se muestra en la figura 18 fragmentos del código que permite hacer dichas operaciones.

```
public void leerArchivoExcel(String archivoDestino, String sub) {//
    try {

        WorkbookSettings conf = new WorkbookSettings();// libreria itex Configurando el
        conf.setEncoding("ISO-8859-1");
        conf.setLocale(new Locale("es", "ES"));
        conf.setExcelDisplayLanguage("ES");
        conf.setExcelRegionalSettings("ES");
        conf.setCharacterSet(CountryCode.SPAIN.getValue());
        Workbook archivoExcel = Workbook.getWorkbook(new File(archivoDestino), conf);//

        for (int nHojas = 0; nHojas < archivoExcel.getNumberOfSheets(); nHojas++) {//re
            Sheet hoja = archivoExcel.getSheet(nHojas);//creando columnas y filas del e
            int nColumnas = hoja.getColumns();
            int nFilas = hoja.getRows();
            String datos = null;
            int num1 = 0;
            int num2 = 0;

            for (int filas = 0; filas < nFilas; filas++) {
                for (int columna = 0; columna < nColumnas; columna++) {//recorriendo fi
                    if (sub.equals("Analizabilidad")) {
                        num1 = Integer.parseInt(hoja.getCell(12, 8).getContents());//ca
                        num2 = Integer.parseInt(hoja.getCell(10, 8).getContents());
                        datos = hoja.getCell(12, 8).getContents();
                    }
                }
```

Ilustración 18 Fragmento de código de la función leerArchivoExcel(String archivoDestino, String sub).

3.2.1 Descripción de clases

Para un mayor entendimiento de la implementación, a continuación se detallan las clases de mayor peso en el sistema. Se describe la clase controladora Auxiliar encargada de manejar el flujo del software en la tabla 6 y la clase controladora perteneciente al visual Principal en la tabla 7, responsable de la creación de los objetos pertinentes que asocian cada una de las interfaces y permiten su visualización.

Nombre: Auxiliar	
Tipo de clase: Controladora	
Atributos	**Tipo**
rutaDestino	String
Para cada responsabilidad	
Nombre:	**Descripción:**

generarReporte(Producto ge)	Permite la generación de un reporte en formato pdf con los resultados del cálculo de la mantenibilidad de un producto.
colocarDestino()	Posibilita una ruta para guardar un documento que contenga los resultados del cálculo de la mantenibilidad.
leerArchivoExcel(String archivoDestino, String sub)	Capturar valores de los campos de una hoja de cálculo para el sub-atributo de la mantenibilidad introducido, teniendo en cuenta el destino del documento insertado como parámetro.
graficar(Producto nu)	Crea un gráfico en un objeto dataset con los resultados del cálculo de la mantenibilidad de un producto.

Tabla 6 Clase Controladora Auxiliar.

Nombre: Principal	
Tipo de clase: Controladora	
Atributos	**Tipo**
conexión	Conexión
Para cada responsabilidad	
Nombre:	**Descripción:**
jMenuItem8ActionPerformed(java.awt.event.ActionEvent evt)	Esta acción posibilita visualizar el resultado del cálculo de la mantenibilidad.
jMenuItem10ActionPerformed(java.awt.event.ActionEvent evt)	Esta acción permite visualizar si un producto se encuentra en la base de datos.
jMenuItem15ActionPerformed(java.awt.event.ActionEvent evt)	Esta acción permite visualizar el listado de productos almacenados en la base de datos.

Tabla 7 Clase Controladora Principal.

3.3 Pruebas de software

Las pruebas de software consisten en la verificación del comportamiento de un programa en un conjunto finito de casos de prueba diseñados utilizando técnicas definidas y debidamente

seleccionados de generalmente infinitas ejecuciones de dominio, contra la del comportamiento esperado. (24)

Las pruebas tienen gran importancia en el desarrollo de un sistema informático, ya que mediante éstas se pueden detectar y corregir posibles fallos de implementación, calidad o usabilidad tempranamente y son un elemento crítico para la garantía del correcto funcionamiento del software.

La práctica de pruebas en la construcción de software reduce el tiempo de desarrollo reduciendo la cantidad de tiempo necesario para integrar y estabilizar versiones. Mejora la productividad encontrando y arreglando errores rápidamente, además de incrementar la calidad global del software garantizando que todo el código nuevo ha sido probado, y a todo el código existente se le aplican pruebas de regresión antes de ser integrados a la base central de código.

3.4 Prueba de Caja Blanca

Las pruebas de caja blanca realizan un seguimiento del código fuente según se van ejecutando los casos de prueba, de manera que se determinan de manera concreta las instrucciones en las que existen errores. Requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas del código.

Objetivo

El objetivo de realizar este tipo de prueba al sistema es garantizar que se ejerciten al menos una vez todos los caminos independientes de cada módulo o método, todos los bucles en sus límites operacionales, las estructuras internas de datos y que se alcancen todas las decisiones lógicas en sus vertientes verdadera y falsas para asegurar su validez. (22)

Alcance

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

Descripción

La prueba de caja blanca es considerada como uno de los tipos de pruebas más importantes que se le aplican al software, logrando como resultado que disminuyan en gran por ciento el número de errores existentes en los sistemas, aumentando la calidad y confiabilidad de los mismos (22)

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

Este tipo de prueba es totalmente objetiva y por lo tanto puede realizarla un ordenador de forma repetitiva, no depende de la experiencia del programador ni de los gustos de la persona que hace la revisión de código.(22)

Las pruebas se realizaron a las clases que contienen funcionalidades que determinan el correcto funcionamiento del sistema entre ellas están:

Para explicar el proceso de realización de las pruebas de Caja Blanca con JUnit se toma como objeto de estudio el método *Cálculo*(), función abstracta que se encuentra en la clase Mantenibilidad y cada una de sus hijas como la clase Analizabilidad, la que se muestra a continuación.

```
public class Analizabilidad extends Mantenibilidad{

    public Analizabilidad() {
    }

    //Capacidad de análisis de fallos
    @Override
    public float Calculo(float Nfallos, float Ntotal) {
        float x=0;
        x= 1-Nfallos/Ntotal;
        return x;
    }
    /*Nfallos: Número de fallos cuyas causas aun no han sido encontradas.
     * Ntotal: Número total de fallos registrados.
     */

}
```

Ilustración 19 Método empleado en las pruebas de Caja Blanca con JUnit.

La realización de pruebas haciendo uso de JUnit permite comprobar si el resultado obtenido luego de la ejecución de la funcionalidad a probar es el esperado. Para la realización de pruebas al método previamente seleccionado se creó, primeramente, una clase la cual debe extender la clase TestCase brindada por el framework JUnit, en este caso dicha clase se nombra AnalizabilidadTest en la cual se realizan una serie de pasos los cuales se muestran y describen a continuación.

```
public class AnalizabilidadTest extends TestCase {

    public AnalizabilidadTest(String testName) {
        super(testName);
    }
```

Ilustración 20 Clase AnalizabilidadTest empleada para la realización de las pruebas.

Primeramente se crea un objeto de la clase en la cual se encuentra el método que se va a probar, luego se definen cada uno de los parámetros que recibe el método. Posteriormente en el método *setUP ()* se inicializan todos los atributos anteriormente creados.

```
public AnalizabilidadTest(String testName) {
    super(testName);
}

public static Test suite() {
    TestSuite suite = new TestSuite(AnalizabilidadTest.class);
    return suite;
}

@Override
protected void setUp() throws Exception {
    super.setUp();
}

@Override
protected void tearDown() throws Exception {
    super.tearDown();
}

/**
 * Test of Calculo method, of class Analizabilidad.
 */
public void testCalculo() {
    System.out.println("Calculo");
    float Nfallos = 1;
    float Ntotal = 2;
    Analizabilidad instance = new Analizabilidad();
    float expResult = 0.0F;
    float result = instance.Calculo(Nfallos, Ntotal);
    assertEquals(expResult, result, 0.5);
```

Ilustración 21 Método testCálculo() de la clase AnalizabilidadTest.

Los resultados de realización de las pruebas son almacenados dentro de una lista. Luego, haciendo uso del método *assertEquals (Object expected, Object actual)* el framework JUnit comprueba que cada

uno de los resultados obtenidos coincide con los resultados esperados y muestra en una ventana los resultados obtenidos, en caso de obtener resultados satisfactorios para todas las pruebas realizadas se observa una línea verde en la ventana, en caso contrario aparece una línea roja. En la figura 22 aparecen los resultados de las pruebas realizadas.

Ilustración 22 Resultados de las pruebas con JUnit.

3.5 Prueba de Caja Negra

Las pruebas de caja negra se llevan a cabo sobre la interfaz de usuario, se centran principalmente en los requisitos funcionales del software. Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Examinan aspectos del modelo, principalmente del sistema, sin tener en cuenta la estructura interna del software.

Objetivo

El objetivo de realizar pruebas de caja negra a un sistema es el de detectar fallos que impidan el correcto o completo funcionamiento de este, así como los errores de interfaces, rendimiento, de inicialización y terminación. (22)

Alcance

El proceso de pruebas de caja negra se va a centrar principalmente en los requisitos funcionales del software para verificar el comportamiento de la unidad observable externamente y su calidad, por lo que se llevarán a cabo las pruebas funcionales sobre las interfaces del sistema y la de aceptación, por parte del cliente.

Descripción

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

La prueba de caja negra no es una alternativa a las técnicas de prueba de la caja blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la caja blanca. (25)

Las pruebas se realizaron sobre todos los requisitos funcionales de la herramienta, haciendo uso de los casos de prueba correspondientes a cada uno de estos. A continuación se muestra un ejemplo:

1.1 Requisitos a probar

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1: Generar reporte.	Se exportan los resultados almacenados en la base de datos del cálculo de la mantenibilidad de un proyecto al formato pdf.	EP 1.1: Generar reporte introduciendo un nombre.	– Se selecciona la opción **Generar reporte**. – Se introduce el nombre del documento con formato pdf que se desea generar en el campo **Nombre de archivo**. – Se presiona el botón **Guardar**.
		EP 1.2 Generar reporte dejando el campo Nombre de archivo vacío.	– Se selecciona la opción **Generar reporte**. – Se presiona el botón **Guardar** sin haber introducido un nombre en el campo **Nombre de archivo**. – El sistema no deja crear el reporte si no es introducido un nombre.
		EP 1.3 Cancelar	– Se selecciona la opción **Generar reporte**. – Se cancela la opción, cerrándose la ventana al presionar el botón **Cancelar**.

Ilustración 23 Fragmento del Caso de Prueba Generar Reporte.

Las pruebas realizadas al sistema fueron satisfactorias desde el punto de vista interno y funcional, estas abarcaron requerimientos, funciones y la lógica interna del sistema. Se aplicaron los métodos de caja negra y caja blanca para validar tanto la interfaz como el correcto funcionamiento interno del software. Combinar ambos enfoques permitió lograr mayor fiabilidad en el proceso de pruebas al diseñar los casos de prueba.

El sistema luego de resueltas las 6 NC detectadas durante las pruebas realizadas fue liberado por el Grupo de Verificación y Validación del CEIGE. Además el cliente efectuó las pruebas de aceptación en el Laboratorio de Pruebas del CEIGE, donde fue finalmente aprobada la herramienta, encontrándose completamente disponible para su utilización.

3.6 Conclusiones

Con las pruebas realizadas y la validación de la solución finaliza el presente trabajo. La confección de este último capítulo posibilitó arribar a las conclusiones siguientes:

- El framework JUnit aumenta las posibilidades que brinda el lenguaje Java para las pruebas de caja blanca y de manera sencilla demuestra la calidad del código generado.
- Los resultados de las pruebas realizadas lograron su objetivo principal: demostrar la calidad de la solución propuesta a través de un código limpio altamente funcional y un funcionamiento general exitoso. Se constató la eficacia de la herramienta JUnit para pruebas de caja blanca.

Conclusiones

Una vez culminado el trabajo y como resultado del mismo se logró alcanzar el objetivo general propuesto, tras cumplir todas las tareas trazadas a inicio de la investigación:

- A partir del análisis de los diferentes conceptos de métricas estudiados referentes a la mantenibilidad se definió un criterio propio para el trabajo con este atributo de calidad.
- Mediante un análisis del estado que presentaba la mantenibilidad en los productos del CEIGE, se demostró la importancia de la presente investigación, al no ser evaluado este atributo de calidad actualmente.
- A partir del análisis de los sistemas informáticos existentes que evalúan las diferentes métricas de la mantenibilidad, se demostró la necesidad e importancia de desarrollar un software que realice el cálculo de métricas externas de la mantenibilidad, pues los sistemas estudiados solo tienen en cuenta las métricas internas y las respuestas que se obtienen de los mismos no están adaptadas al proceso de pruebas del CEIGE.
- Se realizó un estudio de las posibles métricas a implementar seleccionándose aquellas que pudiesen adaptarse a los procesos de pruebas que se realizan en el CEIGE, realizando un aporte en cuanto a la forma de evaluación de este atributo de la calidad.
- Se llevó a cabo el análisis y diseño de la herramienta para la evaluación de la mantenibilidad, aplicando patrones que sirvieron de base fundamental en la implementación del sistema proporcionando rapidez y precisión en el momento de la implementación.
- La implementación de la solución basada en tecnologías libres representa un gran aporte a la soberanía tecnológica por concepto de compra de licencias de software.
- La realización de pruebas permitió validar la implementación de las funcionalidades desarrolladas corroborando la calidad del sistema y demostrando que el mismo está listo para su uso.
- La investigación realizada constituye un aporte decisivo a la informatización de sistemas que permitan evaluar atributos de calidad tales como la mantenibilidad.

Recomendaciones

Tomando como base la investigación realizada y la experiencia acumulada durante el desarrollo del presente trabajo de diploma se recomienda:

- Continuar el estudio del tema con el objetivo de encontrar nuevas funcionalidades para futuras versiones de la aplicación.
- Extender el alcance del sistema de manera tal que este permita la evaluación de otros atributos de calidad.

Bibliografía

1*Institute of Electrical and Electronics Engineers, IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries* New York, NYStandard Glossary of Software Engineering Terminology. IEEE Computer SIEEE Std. 610.12

2*ISO/IEC 9126 -P1*

3WebFinance Inc*WebFinance Inc*http://www.webfinanceinc.com/

4*The Testing Standards Working Party* **BCS SIGiST (British Computer Society Specialist Interest Group in Software Testing)**1998

5*Mantenimiento del Software ,* **Ruiz, Francisco**Dep. de Informática, ESCUELA SUPERIOR DE INFORMÁTICA, Real Grupo Alarcos2001

6*Software Metrics: A Rigorous and Practical Approach* **Fenton N. E. & Pfleeger**International Thomsom Computer Press

7**Díaz Figueredo, Fresneda Cusata, & Batista González,***PROPUESTA DE MÉTRICAS A EMPLEAR EN EL LABORATORIO DE PRUEBAS PARA EVALUAR LA MANTENIBILIDAD DE UN SOFTWARE*La Habana2012

8**Rodríguez***Catálogo automatizado de métricas de calidad para evaluar los productos en las pruebas*La Habana2011

9**Cornejo, José Enrique González**¿Qué es UML?http://www.docirs.cl/uml.ht

10**Herramienta Case Visual Paradigm.** Herramienta Case Visual Paradigmhttp://dianbeel.blogspot.com/2012/06/segundo-trabajo-herramienta-case-visual.html.

11**Herrera, Cristhian.**Informática Profesional.http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=hibernateVSEJB3

12**Larman Cray, J. e. Fred L. Drake, Ed.**UML y Patroneshttp://desarrollo.net

13**JAVA** http://www.java.com/es/download/faq/whatis_java.xml

14**EcuRed**http://www.ecured.cu/index.php/SQL

15 **EcuRed. In: IDE de Desarrollo**EcuRedhttp://www.ecured.cu/index.php/IDE_de_Programaci%C3%B3n

16**¿Qué es NetBeans?** ¿Qué es NetBeans? http://ayuda-java.blogspot.com/2007/07/qu-es-netbeans.html.

17**PostgreSQL: PostgreSQL 9.1** PostgreSQL http://www.postgresql.org/about/press/presskit91/es/.

18**Quiroga, Juan Pablo***Requerimientos Funcionales y No Funcionales*Universidad de los Andes, Dpto de Ingeniería de Sistemas y Computación

19**Dutoi, BerndBrueggey AllenH**Object OrientedSoftware Engineering*Object OrientedSoftware Engineering*PrenticeHal2000

20**Karl, E.Wiegers**Software Requirements*Software Requirements*Microsoft Press1999

21**Kendall, Kendall &**Análisis y Diseño de SistemasPearson Educación.

22**Hill, Roger S Pressman & Mc Graw**Ingeniería del Software: Un enfoque práctico 4ª Edición2005

23**Lorenz, Mark y Kidd, Jeff**Object-Oriented Software Metrics*Object-Oriented Software Metrics*Prentice-Hal994

24**EcuRed**Pruebas de softwarehttp://www.ecured.cu/index.php/Pruebas_de_software

25**Grupo Alarcos**Grupo Alarcoshttp://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema09.pdf.

26*Análisis de métricas básicas y herramientas de código libre para medir la* **Irrazabal, Emanuel; Garzás, Javier;**3Madrid, EspañaAsociación de Técnicos de Informática56-65