

Universidad de las Ciencias Informáticas
“Facultad 3”



Título: Plugin para la evaluación de la diagnosticabilidad en los subsistemas de CedruX.

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

Autor: Greter Encinosa Hernández

Tutor(es): Ing. Dinia Zayas Romero

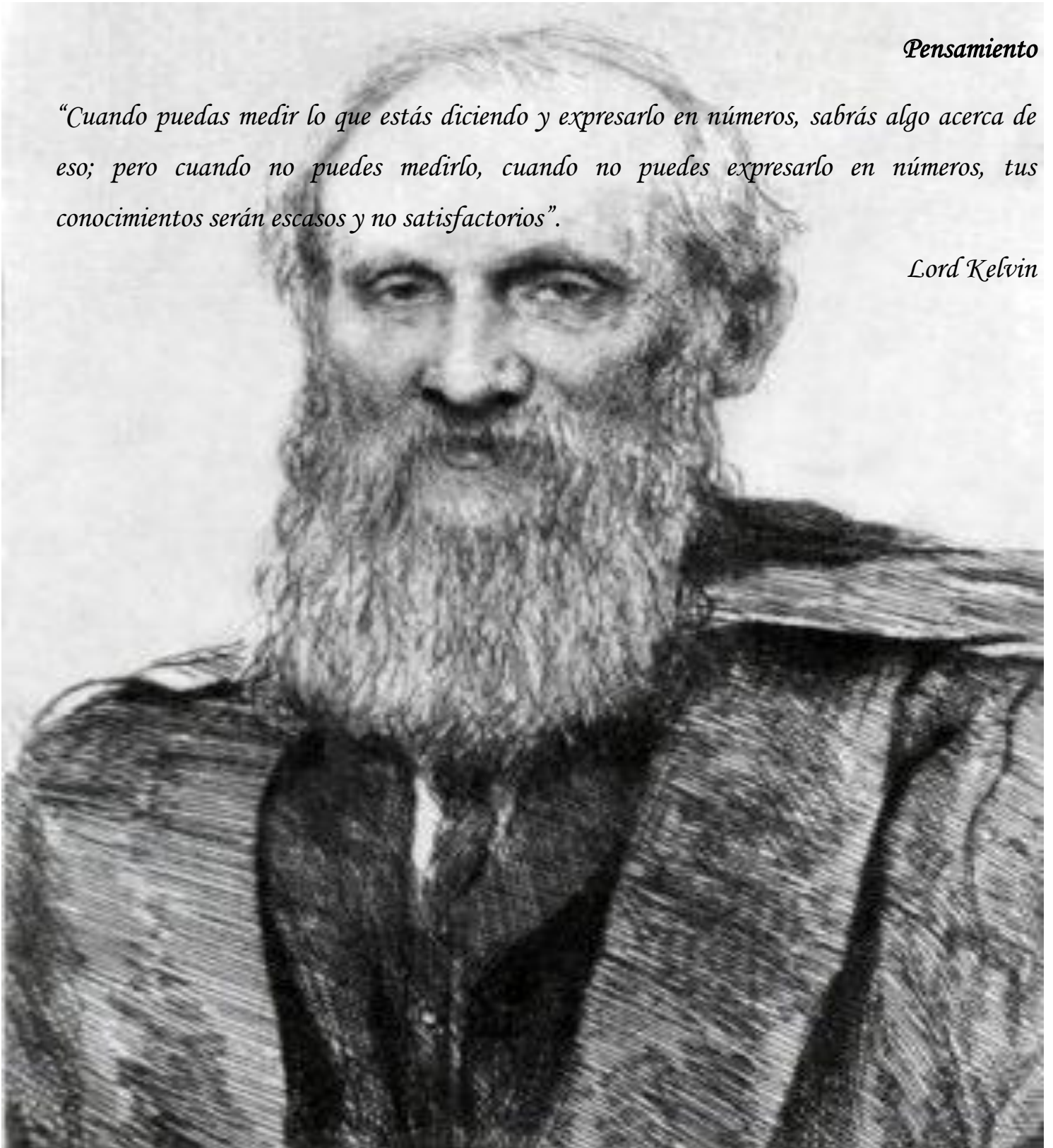
Ing. Leandro Pérez Borroto Vivero

La Habana. Junio, 2013

Pensamiento

“Cuando puedas medir lo que estás diciendo y expresarlo en números, sabrás algo acerca de eso; pero cuando no puedes medirlo, cuando no puedes expresarlo en números, tus conocimientos serán escasos y no satisfactorios”.

Lord Kelvin



DECLARACIÓN DE AUTORÍA

DECLARACIÓN DE AUTORÍA

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

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

Greter Encinosa Hernández

Autor

Ing. Leandro Pérez Borroto

Tutor

Ing. Dinia Zayas Romero

Tutor

Tutora:

Nombre y apellidos Ing. Dinia Zayas Romero

- ✓ Correo electrónico: dzayas@uci.cu.
- ✓ Situación laboral: Profesor.
- ✓ Institución: Universidad de las Ciencias Informáticas (UCI).
- ✓ Dirección: Carretera a San Antonio de los Baños, Km 2 1/2, Reparto Torrens, Boyeros.
- ✓ Rol: Analista principal subsistema Planificación empresarial y Presupuestaria Jefe de proyecto, CedruX v1.0.

Tutor:

Nombre y apellidos: Ing. Leandro Pérez Borroto Vivero

- ✓ Correo electrónico: lvivero@uci.cu.
- ✓ Situación laboral: Profesor.
- ✓ Institución: Universidad de las Ciencias Informáticas (UCI).
- ✓ Dirección: Carretera a San Antonio de los Baños, Km 2 1/2, Reparto Torrens, Boyeros.
- ✓ Rol:

DEDICATORIA

Dedicatoria

A mi abuelita Elsa por cuidarme siempre y a mi abuelo Juan que me hubiese encantado que estuviera aquí.

A mi mami Odalys por ser mi ángel guardián y alumbrar mi camino en la oscuridad.

A mi hermano Alejandro, para que vea en mí un espejo y no cometa los mismos errores. Yo siempre voy a ser tu apoyo y tu guía. Todo mi esfuerzo va dedicado a ti.

AGRADECIMIENTOS

Agradecimientos

AGRADECIMIENTOS GENERALES

A Fidel y la Revolución por crear la universidad y darme la oportunidad de estudiar en ella.

A mis tutores por ayudarme en todo lo que estaba en sus manos.

A todos los que de una forma u otra me alentaron a seguir por sobre todas las dificultades.

AGRADECIMIENTOS PERSONALES

A mi mamá por confiar en mí, por ser mi aliento, mi amiga, mi confidente y darme las fuerzas para seguir.

A mi hermano por quererme tanto y enseñarme a querer.

A mi papá y mi padrastro por ser los hombres que me han guiado en la vida, gracias por confiar en mí y brindarme todo su apoyo.

A mis abuelos Elsa, Nimia, Juan y Albertico por quererme tanto. Los adoro.

A toda la familia en general por apoyarme siempre.

A mis amigas Irina y Liarisbet por cuidar de mí y demostrarme que puedo ser una mejor persona.

A Janier y mis demás compañeros por compartir sus conocimientos sin protestar y por todos los momentos maravillosos que pasamos juntos durante la carrera.

A mi novio por estar a mi lado, por brindarme su apoyo en todo momento, y por hacerme feliz cada día.

A Luis Angel por brindarme sus consejos y alentarme a seguir adelante cuando flaqueo.

A mis amigos por estar siempre ahí y demostrarme cuán fuerte soy.

RESUMEN

La mantenibilidad es un atributo de calidad relevante para todos los Sistemas de Software dentro de estos los Sistemas de Planeación de Recursos Empresariales y uno de sus subatributos es la facilidad de prueba, el cual comprueba la capacidad del producto del software de ser objeto de un diagnóstico para detectar deficiencias o causas de los fallos totales en el software, o para identificar las partes que van a ser modificadas. Por tal motivo, el presente trabajo tiene como objetivo desarrollar un plugin que permita evaluar cuantitativamente la diagnosticabilidad de los subsistemas de Cedrux a partir del modelado de sus arquitecturas, para contribuir a la toma de decisiones con respecto a la arquitectura. Para la realización del mismo es de suma importancia la selección de una métrica adecuada para la evaluación de la arquitectura de los subsistemas, dada la cantidad de elementos a tener en cuenta así como las relaciones entre ellos y la cantidad de componentes de Cedrux. Las métricas de calidad en uso miden si un producto resuelve las necesidades de usuarios específicos para alcanzar metas específicas con eficacia, productividad, seguridad y satisfacción en un contexto dado de uso.

PALABRAS CLAVE:

Arquitectura, diagnosticabilidad, evaluación, mantenibilidad, métricas

ÍNDICE

Tabla de Contenidos

ÍNDICE DE TABLAS	3
ÍNDICE DE FIGURAS	3
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA	5
1.1. Introducción	5
1.2. Arquitectura de software	5
1.2.1. Importancia de evaluar la Arquitectura de Software	6
1.3. Calidad del software	6
1.3.1. Atributos de Calidad	7
1.3.2. Modelos de Calidad	8
1.3.3. Relación entre Arquitectura de Software y Atributos de Calidad	10
1.4. Mantenibilidad	10
1.4.1. Diagnosticabilidad	11
1.5. Evaluación de la arquitectura de software	12
1.5.1. Técnicas de Evaluación de la arquitectura de software	13
1.5.1.1. Las métricas.....	14
1.6. Métricas de diagnosticabilidad	15
Consideraciones generales de las métricas de diagnosticabilidad descritas.....	16
1.7. Tecnologías y herramientas de desarrollo	16
1.7.1. Lenguajes	16
1.7.2. Herramientas.....	20
1.8. Metodología de desarrollo de software	23
1.8.1. Extreme Programming.....	23
1.9. Conclusiones del capítulo	24
CAPÍTULO 2: PROPUESTA DE SOLUCIÓN	26
2.1. Introducción	26
2.2. Propuesta de solución	26
2.3. Modelo Conceptual	26

ÍNDICE

2.3.1. Representación del modelo de dominio	27
2.4. Desarrollo del modelado de la arquitectura en la herramienta AcmeStudio	27
2.4.1. Modelado de la Arquitectura	30
2.4.2. Desarrollo de la métrica	30
2.5. Requisitos del software	33
2.5.1. Requisitos Funcionales	33
2.5.2. Requisitos no Funcionales.....	34
2.6. Modelado del sistema	35
2.6.1. Lista de reserva del producto	35
2.6.2. Descripción de las historias de usuarios	36
2.7. Modelo de Diseño.....	38
2.7.1. Patrones de asignación de responsabilidades utilizados en el diseño	40
2.8. Conclusiones del capítulo	41
CAPÍTULO 3: VALIDACIÓN Y PRUEBAS.....	43
3.1. Introducción.....	43
3.2. Importancia del desarrollo de la propuesta de solución	43
3.3. Proceso de validación de la métrica	43
3.4. Elección de expertos	44
3.5. Elaboración y lanzamiento del cuestionario.....	46
3.6. Desarrollo práctico y análisis de los resultados	48
3.7. Pruebas de Software	51
3.7.1. Pruebas de Caja Blanca	52
3.7.2. Pruebas de Aceptación	55
3.8. Casos de pruebas	57
3.9. Conclusiones del capítulo	59
CONCLUSIONES	60
RECOMENDACIONES	61
BIBLIOGRAFÍA	62

ÍNDICE

ANEXOS	64
GLOSARIO DE TERMINOS	71
ÍNDICE DE TABLAS	
Tabla 1. Clasificación de los componentes	28
Tabla 2. Actores del sistema	35
Tabla 3. Lista de reserva del producto	36
Tabla 4. HU1: Calcular la diagnosticabilidad	36
Tabla 5. HU2.Exportar los resultados obtenidos	37
Tabla 6. Descripción de las Clases del Sistema.....	39
Tabla 7. Coeficiente de expertos.....	45
Tabla 8. Datos obtenidos de los expertos	47
Tabla 9. Frecuencia absoluta de las preguntas de las encuesta	48
Tabla 10. Frecuencia absoluta acumulada.....	49
Tabla 11. Frecuencia relativa acumulada.....	49
Tabla 12. Punto de cortes	50
Tabla 13. Casos de pruebas de Caja Blanca por caminos	54
Tabla 14. Evaluación de los componentes	56
Tabla 15. Caso de prueba de aceptación: HU1	58
Tabla 16. Caso de prueba de aceptación HU2.....	58
ÍNDICE DE FIGURAS	
Ilustración 1. Clasificación de las técnicas de evaluación [12].....	14
Ilustración 2.Modelo de Dominio	27
Ilustración 3. Parte de la Matriz de Integración de subsistema Inventario[22]	28
Ilustración 4. Arquitectura de Software del subsistema Inventario.....	30
Ilustración 5. Diagrama de Clases del Sistema	39
Ilustración 6. Análisis de las frecuencia absoluta de las preguntas de la encuestas.....	49
Ilustración 7. Puntos de cortes	51
Ilustración 8. Código fuente del método entradasPorComponentes de la clase AcmeConverter.....	53
Ilustración 9. Grafo de flujo asociado al método entradasPorComponentes.....	54

INTRODUCCIÓN

Introducción

En la medida que los sistemas de software crecen en complejidad, bien sea por número de requerimientos o por el impacto de los mismos, se hace necesario establecer medios para el manejo de esta realidad. En general, la técnica es descomponer el sistema en piezas que agrupan aspectos específicos del mismo, producto de un proceso de abstracción y que al organizarse de cierta manera constituyen la base de la solución de un problema en particular.

La arquitectura representa un papel fundamental en la aplicación de esta técnica como pieza del sistema, definida como el conjunto de decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y sus interfaces por las que se compone el sistema, junto con su comportamiento tal como se especifica en las colaboraciones entre esos elementos.[1]

La Arquitectura de los Sistemas de Software al ser construidos se convierte en un factor primordial, para lograr que este tenga un alto nivel de calidad. Por lo que se considera necesario brindarle un lugar importante a la definición y evaluación de la Arquitectura de Software, debido a que este elemento permite a un nivel de abstracción significativo analizar la estructura y composición del sistema. Evaluar una arquitectura es realizar una valoración del comportamiento de los atributos de calidad requeridos desde el diseño arquitectónico, e identificar cuáles son los riesgos que representa la selección de una arquitectura determinada. Los atributos de calidad como mantenibilidad, estabilidad, facilidad de análisis, facilidad de cambio y facilidad de pruebas constituyen elementos de alta relevancia a ser considerados en la arquitectura.

Dentro de la Calidad de Software como característica fundamental a tener en cuenta se encuentra la mantenibilidad según el modelo de calidad ISO/IEC 9126 se define 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.[2]

Para que un sistema tenga la calidad requerida, confiabilidad, seguridad de la información y una buena repercusión en el mercado es de suma importancia aplicar las propiedades de mantenibilidad entre ellas la diagnosticabilidad, para realizar evaluaciones a la arquitectura del mismo. Con ella se comprueba la capacidad del producto de ser objeto de un diagnóstico para detectar deficiencias o causas de los fallos totales en el software, o para identificar las partes que van a ser modificadas.

Cuba crea la Universidad de las Ciencias Informáticas (UCI) como parte de las tareas para la informatización de los servicios como la educación. Entre los objetivos para lo que fue fundada se encuentra el desarrollo de software, el cual desempeña un papel importante en la vida económica del

INTRODUCCIÓN

país. A pesar de que en la UCI se han creado varios sistemas de gestión como solución a diferentes problemáticas que han surgido dentro de los distintos centros de desarrollo de software que se encuentran en las facultades que la integran, en estos momentos la Facultad 3 cuenta con dos centros de desarrollo de software, uno de estos es el Centro de Informatización de la Gestión de Entidades (CEIGE), en el cual uno de los productos que se desarrollan es el sistema integral de gestión de recursos empresariales, Cedrux.

Este proyecto contiene diferentes subsistemas como Capital Humano, Facturación, Contabilidad, Costos y Procesos, Seguridad, Estructura y Composición e Inventario. El subsistema que se utilizará como muestra para realizar la evaluación y aplicarle la técnica seleccionada es el que se encarga de la gestión de control de Inventario el cual es un proceso que sirve para guiar la gestión empresarial hacia los objetivos de la organización y un instrumento para evaluarla. El sistema de control de inventario posibilita el registro de los productos, sus existencias, la gestión control de lotes, así como los movimientos de entrada y salida de los productos en el almacén.

La forma actual de evaluar la arquitectura en Cedrux, es mediante la revisión de diferentes artefactos de forma manual, lo que conlleva a que la toma de decisiones con respecto al rediseño de la arquitectura y los posibles cambios que puedan ocurrir en ella se pueden ver atrasados durante el proceso de desarrollo, situación esta que deriva a un incumplimiento con el tiempo pactado con el cliente y el tiempo de desarrollo de la arquitectura.

De acuerdo con la problemática planteada se define como **problema a resolver**: ¿Cómo evaluar cuantitativamente la diagnosticabilidad de la arquitectura en los subsistemas de Cedrux, para contribuir a la toma de decisiones?

El mencionado problema a resolver determina como **objeto de estudio**: Evaluación de la arquitectura de software.

Campo de acción: Evaluación cuantitativa de la diagnosticabilidad en la arquitectura de software.

Por lo tanto, para el desarrollo de la investigación se precisó como **objetivo general**: Desarrollar un plugin que permita evaluar cuantitativamente la diagnosticabilidad de los subsistemas de Cedrux a partir del modelado de sus arquitecturas, para contribuir a la toma de decisiones.

Para darle cumplimiento al objetivo general se desglosan los siguientes **objetivos específicos**:

INTRODUCCIÓN

1. Realizar un estudio de las tendencias en la evaluación de las arquitecturas de software para seleccionar una técnica que permita evaluar la diagnosticabilidad de la arquitectura de los subsistemas de Cedrux.
2. Diseñar un plugin para Eclipse que permita aplicar la técnica seleccionada a la arquitectura de los subsistemas de Cedrux.
3. Implementar el diseño del plugin para obtener el grado de diagnosticabilidad de los subsistemas de Cedrux.
4. Validar la solución propuesta a través de pruebas y criterio de expertos.

Para dar cumplimiento a los objetivos específicos de la investigación se han propuesto las siguientes **tareas de la investigación:**

- Estudio de las tendencias para la evaluación de las arquitecturas de software.
- Selección de la métrica que se utilizará para la evaluación.
- Modelado de la arquitectura del subsistema Inventario utilizando el lenguaje de Descripción de la Arquitectura Acme.
- Diseño de un plugin para Eclipse que permita aplicar la métrica seleccionada para la evaluación de las arquitecturas de los subsistemas de Cedrux a partir del modelo de su arquitectura.
- Implementación del diseño realizado.
- Validación de la solución propuesta.

Con el propósito de desarrollar las tareas planteadas para el desarrollo de la investigación se utilizaron los **métodos de investigación** siguientes:

✓ **Métodos teóricos:**

Analítico-Sintético: Para estudiar diferentes fuentes bibliográficas relacionados con las distintas técnicas de la evaluación de la arquitectura de software. Se realizaron además resúmenes y valoraciones de conceptos relevantes relacionados con arquitectura, calidad de software y evaluación de la arquitectura. Además, para analizar a través de una profunda búsqueda las tecnologías y herramientas a utilizar en el desarrollo de la herramienta.

Inductivo-Deductivo: Este método se empleó en la descripción e implementación de las funcionalidades del sistema.

Modelado: Se utilizó para la conformación del modelado de la arquitectura en la herramienta AcmeStudio.

INTRODUCCIÓN

✓ **Métodos empíricos:**

Entrevista: Se realiza este método con el objetivo de obtener información sobre la gestión del subsistema Inventario del Sistema Integral de Gestión de Recursos Empresariales Cedrux, intercambiando directamente con las personas involucradas en el mismo. Además, permite reunir y determinar la información necesaria para la recogida de requisitos del sistema.

Capítulo 1: Fundamentación Teórica

En este capítulo se realiza un análisis de los conceptos y elementos de gran utilidad para la evaluación de la diagnosticabilidad en los subsistemas de Cedrux, desarrollados bajo el Lenguaje de Descripción de la Arquitectura (ADL por sus siglas) y el lenguaje de programación Java. Además, se hace un estudio de las técnicas que existen actualmente y la importancia que tiene realizar un análisis del software utilizando las técnicas de evaluación. Además, se exponen las herramientas, los lenguajes y la metodología que se emplearán en el desarrollo del plugin.

Capítulo 2: Propuesta de Solución

En este capítulo se explica la métrica seleccionada para la evaluación de la arquitectura. Se ofrece la propuesta de solución para evaluar la diagnosticabilidad de las arquitecturas de los subsistemas de Cedrux, desarrollados bajo el lenguaje de Descripción de la Arquitectura y el lenguaje de programación Java. Se describen las principales características del sistema a desarrollar. Así como la especificación de los requisitos funcionales que son las funcionalidades que brindará el plugin y los requisitos no funcionales.

Capítulo 3: Validación y Pruebas

En este capítulo se valida la propuesta de solución a partir de la aplicación de la evaluación de la diagnosticabilidad del subsistema Inventario a partir del modelado de su arquitectura, desarrollado en el Sistema Integral de Gestión de Recursos Empresariales (ERP por sus siglas), Cedrux, bajo el Lenguaje de Descripción de la Arquitectura y el lenguaje de programación Java, evaluando la diagnosticabilidad de los mismos mediante métricas. Además, se desarrolla un modelo de prueba para comprobar la efectividad del sistema.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

Durante el desarrollo del capítulo se abordarán las diferentes definiciones que resultarán necesarias a la hora de desarrollar el plugin para la evaluación de la diagnosticabilidad de los subsistemas de CedruX, a través de su arquitectura. Se hará una descripción de las técnicas de evaluación de la arquitectura de software enfocado en las cuantitativas para determinar qué técnica es la que más se ajusta para la realización del plugin y de esta técnica se estudiarán las métricas de diagnosticabilidad. También se explicarán algunos conceptos fundamentales como arquitectura, evaluación de software, calidad de software entre otras. Además, se expondrán las tecnologías y herramientas propuestas para su desarrollo.

1.2. Arquitectura de software

Actualmente es posible encontrar numerosas definiciones del término Arquitectura de Software, cada una con planteamientos diversos. Se hace evidente que su conceptualización sigue todavía en discusión, ya que no es posible referirse a un diccionario en busca de un significado y tampoco existe un estándar que pueda ser tomado como marco de referencia. Sin embargo, al hacer un análisis detallado de cada uno de los conceptos disponibles, resulta interesante la existencia de ideas comunes entre los mismos, sin observarse planteamientos contradictorios, sino complementarios.

La arquitectura de software puede ser vista como la estructura del sistema en función de la definición de los componentes y sus interacciones. La arquitectura de software puede considerarse como el “puente” entre los requerimientos del sistema y la implementación.[1]

Según el estándar IEEE (Instituto de Ingenieros Eléctricos y Electrónicos): “La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución”.[3]

De acuerdo con las definiciones antes mencionadas se puede decir entonces que la Arquitectura de Software no es más que tener una vista global y abstracta del sistema que se desea desarrollar, que permita tomar las decisiones adecuadas, definir los parámetros y delimitación de dicho sistema tomando en cuenta los requerimientos funcionales y no funcionales. Esta es considerada como la estructura a grandes rasgos del sistema, consistente en elementos y la relación entre ellos. Al describir la Arquitectura de Software es importante identificar cuáles son los elementos de interés y seleccionar una manera apropiada para describirlos.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.2.1. Importancia de evaluar la Arquitectura de Software

Una arquitectura de software define la estructura del sistema. Esta estructura se constituye de componentes, módulos o piezas de código que nacen de la noción de abstracción, cumpliendo funciones específicas, e interactuando entre sí con un comportamiento definido. Los componentes se organizan de acuerdo con ciertos criterios, que representan decisiones de diseño. En este sentido, hay autores que plantean que la arquitectura de software incluye justificaciones referentes a la organización y el tipo de componentes, garantizando que la configuración resultante satisface los requerimientos del sistema.

La necesidad de evaluar la arquitectura de software nace del impacto que esta posee sobre la calidad del sistema. Si se tiene en cuenta que del buen diseño arquitectónico depende el cumplimiento de los atributos de calidad y el éxito en sí del producto; entonces es preciso que se tomen las medidas necesarias para garantizar que este diseño, haya sido desarrollado de la manera correcta. La principal tarea que se debe llevar a cabo para asegurar que la arquitectura propuesta cumple con las necesidades del sistema, es evaluar dicha arquitectura.

El propósito de realizar evaluaciones a la arquitectura, es para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante, verificar que los requerimientos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad.[4]

1.3. Calidad del software

En la actualidad el término calidad ha evolucionado de forma ascendente logrando notables avances en el mejoramiento de la posición de la organización, aumentado sus niveles de eficiencia y por ende sus beneficios. Por lo cual han surgido diferentes criterios acerca de la calidad de software y en correspondencia a esto las empresas se empeñan en mejorar cada vez más la calidad de sus productos.

Se entiende en la investigación por Calidad de Software “la concordancia con los requerimientos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo documentados y con las características implícitas que se esperan de todo software desarrollado profesionalmente”. [5]

De acuerdo con la terminología de la IEEE “la calidad de un sistema, componente o proceso de desarrollo de software se obtiene en función del cumplimiento de los requerimientos especificados por el cliente o usuario final”. Es una compleja mezcla de factores que variarán a través de diferentes aplicaciones y según los clientes que las pidan. Dichos factores se pueden dividir en 2 grupos:[6]

- Factores que se pueden medir directamente.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

- Factores que se pueden medir solo indirectamente.

La Arquitectura de los Sistemas de Software al ser construidos, se convierte en un factor de importancia para lograr que este tenga un alto nivel de calidad. El poseer una buena Arquitectura es de suma importancia, ya que esta es el corazón de todo sistema y determina cuáles serán los niveles de calidad asociados al sistema.

De lo anterior se puede deducir que los objetivos de evaluar una arquitectura son saber si puede habilitar los requerimientos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los stakeholders y en qué grado lo hace. Además de analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante.

1.3.1. Atributos de Calidad

La calidad de software se define como el grado en el cual el software posee una combinación deseada de atributos. Tales atributos son requerimientos adicionales del sistema, que hacen referencia a características que este debe satisfacer, diferentes a los requerimientos funcionales. Estas características o atributos se conocen con el nombre de atributos de calidad, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios.

A grandes rasgos, Bass establece una clasificación de los atributos de calidad en dos categorías:[7]

- Observables vía ejecución: Aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución.
- No observable vía ejecución: Aquellos atributos que se establecen durante el desarrollo del sistema.

1.3.1.1. Atributos de calidad observables vía ejecución

- Disponibilidad: Es la medida de disponibilidad del sistema para el uso.
- Confidencialidad: Es la ausencia de acceso no autorizado a la información.
- Funcionalidad: Habilidad del sistema para realizar el trabajo para el cual fue concebido.
- Desempeño: Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria.
- Confiabilidad: Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

- Seguridad externa: Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información.
- Seguridad interna: Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.[7]

1.3.1.2. Atributos de calidad no observable vía ejecución

- Configurabilidad: Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
- Integralidad: Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
- Integridad: Es la ausencia de alteraciones inapropiadas de la información.
- Interoperabilidad: Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema.
- Modificabilidad: Es la habilidad de realizar cambios futuros al sistema.
- Mantenibilidad: Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.
- Portabilidad: Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.
- Reusabilidad: Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.
- Escalabilidad: Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
- Capacidad de Prueba: Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba.[7]

1.3.2. Modelos de Calidad

Habiendo visto con anterioridad la importancia adquirida por los atributos de calidad en la producción mundial de software, corresponde presentar algunos de los modelos de calidad, surgidos con la idea de brindar una organizada, detallada y estructurada descomposición de la calidad global en características y subcaracterísticas de calidad. Los factores que afectan a la calidad del software no cambian, por lo que

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

resulta de gran utilidad el estudio de los modelos de calidad propuestos. Existen diferentes modelos de calidad como son el Modelo de McCall, el Modelo de Dromey, el Modelo FURPS y el Modelo ISO/IEC 9126. Este último modelo es el que se utiliza en el centro ya que brinda unas series de características muy importantes para lo que se quiere desarrollar.

➤ **ISO/IEC 9126**

Estándar internacional para la evaluación del software. Es supervisado por el proyecto SquaRE (Ingeniería de Requisitos de Calidad de Seguridad) y la ISO 25000:2005, que siguen los mismos conceptos generales. Este surge debido a la necesidad de un modelo único para expresar la calidad de un software. El estándar está dividido en cuatro partes las cuales dirigen, respectivamente, lo siguiente: Modelo de calidad, métricas externas, métricas internas y calidad en las métricas de uso y expendido. El modelo de calidad establecido en la primera parte del estándar, ISO 9126-1, clasifica la calidad del software en un conjunto estructurado de características y subcaracterísticas de la siguiente manera:

- Funcionalidad
- Fiabilidad
- Usabilidad
- Eficiencia
- Mantenibilidad
- Portabilidad
- Estabilidad
- Facilidad de análisis
- Facilidad de cambio
- Facilidad de pruebas

Cada subcaracterísticas está dividida en atributos. Un atributo es una entidad la cual puede ser verificada o medida en el producto software. Los atributos no están definidos en el estándar, ya que varían entre diferentes productos software. Un producto software está definido en un sentido amplio como: Los ejecutables, código fuente y descripciones de arquitectura. Como resultado, la noción de usuario se amplía tanto a operadores como a programadores, los cuales son usuarios de componentes como son bibliotecas software.

El estándar provee un entorno para que las organizaciones definan un modelo de calidad para el producto software. Haciendo esto así, sin embargo, se lleva a cada organización la tarea de especificar

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

precisamente su propio modelo. Esto podría ser hecho, por ejemplo, especificando los objetivos para las métricas de calidad las cuales evalúan el grado de presencia de los atributos de calidad. La calidad en las métricas de uso está solo disponible cuando el producto final es usado en condiciones reales.

Métricas internas son aquellas que no dependen de la ejecución del software (medidas estáticas). Métricas externas son aquellas aplicables al software en ejecución.[8]

1.3.3. Relación entre Arquitectura de Software y Atributos de Calidad

Bass establece que la relación entre arquitectura de software y atributos de calidad tiene diversos beneficios, que justifican su documentación. Realiza en gran medida el proceso de análisis y diseño arquitectónico, puesto que el arquitecto puede reutilizar análisis existentes y determinar acuerdos explícitamente en lugar de hacerlo sobre la marcha. Los arquitectos experimentados hacen esto intuitivamente, basados en su experiencia en codificación. Por ejemplo, durante el análisis, un arquitecto puede reconocer el impacto de la codificación de una estructura en los atributos de calidad.

Una vez que el arquitecto entiende el impacto de los componentes arquitectónicos sobre uno o varios atributos de calidad, estaría en capacidad de reemplazar un conjunto de componentes por otro cuando lo considere necesario.

Una vez que se codifica la relación entre arquitectura y atributos de calidad, es posible construir un protocolo de pruebas que habilitará la certificación por parte de terceros. Por todo lo expuesto, se reconoce la importancia de la arquitectura de un sistema de software como la base de diseño de un sistema, así como también un artefacto que determina atributos de calidad.

A lo largo del proceso de diseño y desarrollo, los atributos de calidad desempeñan un papel importante, pues en función de estos se generan las decisiones de diseño y argumentos que los justifican. Dado que la arquitectura de software inhibe o facilita los atributos de calidad, resulta de particular interés analizar la influencia de ciertos elementos de diseño utilizados para la definición de la misma, determinando sus características.[7]

1.4. Mantenibilidad

Dentro de la Calidad de Software como característica fundamental a tener en cuenta en esta investigación se encuentra la mantenibilidad, la cual se define 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. Es el atributo de calidad del

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

software que más directamente influye en los costes y necesidades del mantenimiento, debido a que a mayor mantenibilidad (facilidad de mantenimiento), menores costes de mantenimiento y viceversa.[9]

El IEEE (Institute of Electrical and Electronics Engineers) define la mantenibilidad como: La facilidad con la que un sistema o componente software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno.[2]

Estas definiciones están directamente conectadas con la definición del IEEE para mantenimiento del software: Es el proceso de modificar un componente o sistema software después de su entrega para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarlo a cambios en el entorno.[2] En este sentido aunque las modificaciones internas no serán realizadas por el usuario del componente (desarrollador), sí necesitará probar el componente antes de incluirlo en su aplicación o cambiar alguno de los parámetros que se pueden particularizar. Por ello, las subcaracterísticas Confiabilidad y Facilidad de prueba son las que deben ser medidas para los componentes.

1.4.1. Diagnosticabilidad

El término de diagnosticabilidad se refiere a la capacidad del producto del software de ser objeto de un diagnóstico para detectar deficiencias o causas de los fallos totales en el software, o para identificar las partes que van a ser modificadas.[2]

Esta subcaracterística también se puede ver en otras bibliografías con otro nombre y se conoce la Facilidad de prueba como lo fácil que se puede probar un programa de computadora. Un software fácil de probar se refiere a la facilidad con la cual el software puede ser construido para posteriormente encontrar defectos o problemas directamente probando sus componentes. Debe existir una lista de comprobación que proporcione un conjunto de características que llevan a un software fácil de probar.[10]

Para que un sistema sea apropiadamente fácil de probar, este debe ser capaz de controlar cada estado interno de los componentes y las entradas para así observar las salidas. Frecuentemente esto se hace directamente usando un software específico para realizar pruebas. Algunas de las posibles medidas utilizadas en los escenarios de facilidad de pruebas son: El tiempo en realizar una prueba (tiempo promedio registrado en realizar una prueba a un componente de un tamaño específico) y el número de errores encontrados durante la ejecución de las pruebas (se realiza una estimación de la probabilidad de encontrar más fallas después de realizar las pruebas a un componente).[7]

La definición por la que se regirá la investigación será la planteada por la ISO/IEC 9126 donde define que un software fácil de probar se refiere a la facilidad con la cual el software puede ser construido para

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

posteriormente encontrar defectos o problemas directamente probando sus componentes. Como la prueba es tan profundamente difícil, merece la pena saber qué puede hacer para hacerlo más sencillo.

1.5. Evaluación de la arquitectura de software

La evaluación de una arquitectura de software no es una tarea sencilla y en muchos de los casos no se realiza; sin embargo, el no llevar a cabo su evaluación puede acarrear problemas que, en etapas posteriores del desarrollo, implicarían más costos, tiempo, recursos, etc. La evaluación es un estudio de factibilidad que pretende detectar posibles riesgos, así como buscar recomendaciones para contenerlos. La diferencia entre evaluar y verificar es que la evaluación se realiza antes de la implementación de la solución. La verificación es con el producto ya construido.[11]

Dado que la Arquitectura de Software puede ser vista como la estructura o estructuras del sistema que comprende componentes de software, propiedades externas de esos componentes y la interacción entre ellos, resulta interesante enfocarse en evaluarla para poder detectar problemas que si no se descubren podría ser peor descubrirlos tardíamente, obteniendo como resultado mejores arquitecturas. La manera de comprobar que se está desarrollando una arquitectura de calidad es a través de su evaluación.

El primer paso para la evaluación de una arquitectura es conocer qué es lo que se quiere evaluar. De esta forma, es posible establecer la base para la evaluación, puesto que la intención es saber qué se puede evaluar y qué no. Resulta interesante el estudio de la evaluación de una arquitectura, si las decisiones que se toman sobre la misma determinan los atributos de calidad del sistema, es entonces posible evaluar las decisiones de tipo arquitectónico con respecto al impacto sobre estos atributos.[1]

La arquitectura de software posee un gran impacto sobre la calidad de un sistema, por lo que es muy importante estar en capacidad de tomar decisiones acertadas sobre ella, en diversos tipos de situaciones, entre las cuales destacan:[7]

- Comparación de alternativas similares.
- Comparación de la arquitectura original y la modificada.
- Comparación de la arquitectura de software con respecto a los requerimientos del sistema.
- Comparación de una arquitectura de software con una propuesta teórica.
- Valoración de una arquitectura en base a escalas específicas.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.5.1. Técnicas de Evaluación de la arquitectura de software

Las técnicas de evaluación de la arquitectura, son instrumentos o herramientas que conjugadas con otros elementos, proveen a los involucrados en el proceso, de una vía para alcanzar sus objetivos. Estas permiten al mismo tiempo pronosticar el comportamiento de la arquitectura de software en su etapa de diseño.

Un gran grupo de estas técnicas están divididas en cualitativas y cuantitativas. Por lo regular, las técnicas de evaluación cualitativas son utilizadas cuando la arquitectura está en construcción. Se aplican para la comparación entre arquitecturas candidatas y tiene relación con la intención de saber la opción que se adapta mejor a cierto atributo de calidad. Este tipo de medición brinda respuestas afirmativas o negativas, sin mayor nivel de detalle.

Las técnicas de evaluación cuantitativa se usan cuando la arquitectura ya ha sido implantada. Estas buscan la obtención de valores que permitan tomar decisiones en cuanto a los atributos de calidad de una arquitectura de software. El esquema general, es la comparación con márgenes establecidos, como lo es el caso de los requisitos de desempeño, para establecer el grado de cumplimiento de una arquitectura candidata, o tomar decisiones sobre ella. Este enfoque permite establecer comparaciones, pero se ve limitado en tanto no se conozcan los valores teóricos máximos y mínimos de las mediciones con las que se realiza la comparación.[12]

En las técnicas cualitativas están comprendidas las técnicas de escenarios, cuestionarios y listas de verificación, mientras que las cuantitativas abarcan las técnicas basadas en métricas, simulaciones, prototipos, experimentos o modelos matemáticos [4]. La figura 1 muestra las diferentes técnicas de evaluación.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

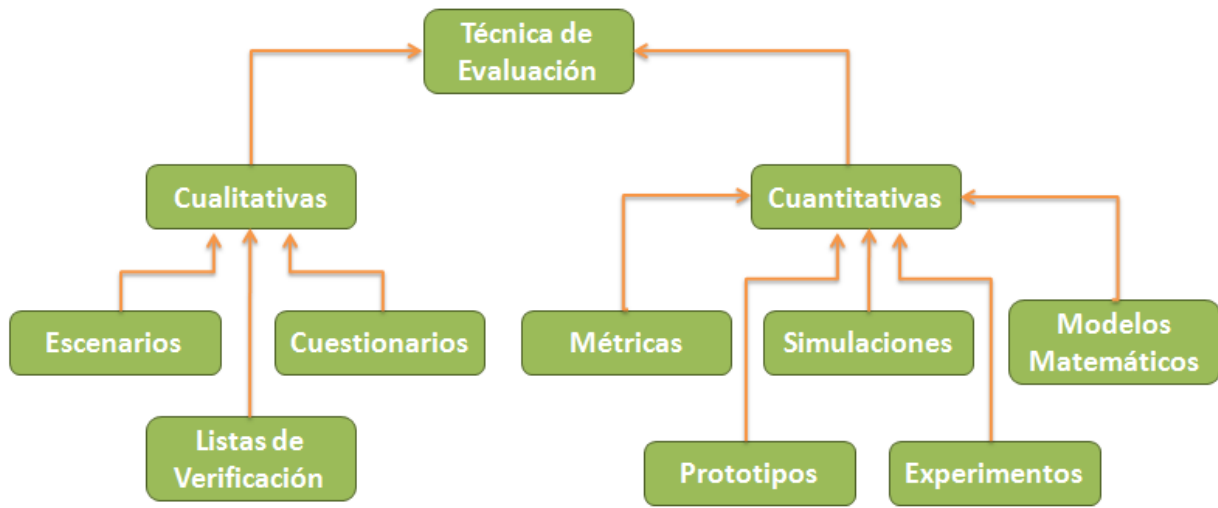


Ilustración 1. Clasificación de las técnicas de evaluación [12]

1.5.1.1. Las métricas

Las métricas pueden ser usadas para medir la calidad del producto de software a través de la medición del comportamiento del sistema del cual el software forma parte. Las métricas de calidad en uso miden si un producto resuelve las necesidades de usuarios específicos para alcanzar metas específicas con eficacia, productividad, seguridad y satisfacción en un contexto dado de uso. Esto solo puede lograrse en un entorno real del sistema. Esta norma permite especificar y evaluar la calidad del producto de software desde las perspectivas de aquellos asociados con la adquisición, regulación, desarrollo, uso, evaluación, soporte, mantenimiento, aseguramiento de la calidad y auditoría del software.[12]

La utilización de métricas para medir la calidad software no se tenía muy en cuenta hasta la aparición de la norma ISO/IEC 9126, la cual condujo al uso de estas por parte de las empresas desarrolladoras de software debido a que dicha norma definió métricas para medir cada una de las subcaracterísticas de los atributos de calidad contenidos en su clasificación. Las métricas se definen como: Interpretaciones cuantitativas sobre mediciones observables realizadas a la arquitectura, las cuales establecen una medida del estado en el que se encuentra el atributo de calidad en cuestión, dentro del sistema.[12]

La posibilidad que ofrecen estas de ser adaptadas al sistema y de emitir un resultado que ayude a la toma de decisiones, es precisamente la razón principal que las sitúa como la técnica cuantitativa elegida para efectuar la medición de la diagnosticabilidad de los subsistemas de Cedrux. Durante el desarrollo de la investigación se realizó un estudio de métricas, las cuales establecen indicadores y criterios para la evaluación de la diagnosticabilidad.[12]

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Consideraciones sobre las Técnicas de Evaluación

En este sentido, se decidió utilizar las técnicas cuantitativas de evaluación de la arquitectura, especialmente las métricas, puesto que establecen criterios de medida a través de los cuales se pueden realizar interpretaciones de la evaluación y por tanto contribuir a la toma de decisiones. Las métricas miden si un producto cumple con todos los requerimientos para complacer a los clientes del sistema. Es importante medir el proceso de ingeniería de software y el producto que se elabora porque es la forma más objetiva de comprender y mejorar el proceso de desarrollo. Si no se realizan mediciones, no hay forma de determinar si se está mejorando, las decisiones se basan solo en evaluaciones subjetivas, lo que puede llevar a malas estimaciones o interpretaciones erróneas del proceso. Para establecer objetivos de mejora es necesario conocer el estado actual de desarrollo del software. La evaluación de la arquitectura constituye un proceso relevante y de estricto cumplimiento, el nivel de control y de detalle que este puede introducir dará al traste innumerables aportes significativos en la calidad del producto.

1.6. Métricas de diagnosticabilidad

Las métricas externas de diagnosticabilidad deben ser capaces de medir un atributo como es el esfuerzo de mantenimiento o del usuario y el gasto de recursos cuando se intente diagnosticar defectos o causas de fallos o para la identificación de partes a modificar.[13]

Las métricas se proponen medir:

- ¿Cuán capaces son las funciones de diagnóstico implementadas para efectuar análisis de causas?
- ¿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?

El Método de aplicación: Observe el comportamiento del operador que trata de resolver los fallos utilizando funciones de diagnóstico. **Interpretación del valor obtenido:** $0 \leq X \leq 1$ mientras más cercano al 1, mejor. **Escala:** Absoluta, **Tipo de medida:** $X = \text{contable} / \text{contable}$, $A = \text{contable}$ y $B = \text{contable}$. **Fuentes:** Informe de evaluación y Reporte de solución de problemas. **Ref. a la ISO/IEC 12207:** Pruebas de calificación, Implantación y Mantenimiento.[13]

A) Nombre de la métrica: Grado de implementación de las funciones de diagnóstico.

Medición (fórmula): $X = A / B$

A - Número de fallos que pueden ser diagnosticadas (utilizando las funciones de diagnóstico) para

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

entender la correlación causa- efecto y B - número total de fallos registrados.

B) Nombre de la métrica: Capacidad de análisis de fallos.

Medición (fórmula): $X = 1 - A / B$

A - Número de fallos cuyas causas aún no han sido encontradas y B - número total de fallos registrados.

C) Nombre de la métrica: Localidad de la modificación.

Medición (fórmula): $X = \text{Sum}(T) / N$

Donde T – es el tiempo empleado en probar con el fin de asegurar si el informe de fallo ha sido o no resuelto y N - números de fallos resueltos.

Consideraciones generales de las métricas de diagnosticabilidad descritas

Una vez descritas las métricas de diagnosticabilidad, se ha llegado a la conclusión de que todas poseen como punto común evaluar la diagnosticabilidad de forma cuantitativa. Todas las métricas poseen el mismo método de aplicación, la misma escala y todas proponen medir lo mismo, saber si el usuario puede identificar fácilmente la causa del fallo. Los objetivos principales de estas métricas son comprender mejor la calidad del producto y mejorar la calidad del trabajo realizado en el nivel del proyecto. Para la evaluación de la arquitectura de los subsistemas de Cedrux se llegó a la conclusión de que no es posible utilizarlas en su estado original. Es evidente el hecho de que las métricas propuestas por la ISO/IEC 9126 vistas anteriormente se encuentran enfocadas a productos y resulta más difícil aplicarlas cuando se trata de un modelo estático de la arquitectura .Por lo que es necesario realizar modificaciones o crear una nueva métricas que sea capaz de obtener los valores que brinda el modelado de la arquitectura en la herramienta AcmeStudio.

1.7. Tecnologías y herramientas de desarrollo

La selección correcta de las herramientas y tecnologías que se utilizan en el desarrollo de un software se traduce en ahorro de tiempo y trabajo dentro de cualquier proyecto. A continuación se realiza una breve descripción de las tecnologías y herramientas que serán utilizadas en el desarrollo de la aplicación.

1.7.1. Lenguajes

Un lenguaje es un idioma artificial diseñado para expresar procesos que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana.[14]

1.7.1.1. Lenguaje para el modelado de la arquitectura

Para el modelado de la arquitectura se utilizará el Lenguaje de Descripción arquitectónica (ADL por sus siglas en inglés) esta es un lenguaje o notación para describir una arquitectura de software, que generalmente incluye una descripción de componentes, conectores, enlaces entre ellos y el prototipado rápido.

La definición más simple es la de Tracz que define un ADL como una entidad consistente en cuatro “Cs”: componentes, conectores, configuraciones y restricciones.[7]

Una de las definiciones más tempranas es la de Vestal quien sostiene que un ADL debe modelar o soportar los siguientes conceptos:[7]

- Componentes.
- Conexiones.
- Composición jerárquica, en la que un componente puede contener una sub-arquitectura completa.
- Paradigmas de computación, es decir, semánticas, restricciones y propiedades no funcionales.
- Paradigmas de comunicación.
- Modelos formales subyacentes.
- Soporte de herramientas para modelado, análisis, evaluación y verificación.
- Composición automática de código aplicativo.

Acme es uno de los ADL más usados en la actualidad, este se define como un lenguaje capaz de soportar el mapeo de especificaciones arquitectónicas entre diferentes ADL, o en otras palabras, como un lenguaje de intercambio de arquitectura. Además define siete elementos básicos que son: Componentes, conectores, sistemas, puertos, roles, representaciones y mapas de representación.

Por lo tanto, un ADL es un lenguaje que proporciona características para modelar la arquitectura conceptual de un sistema software, distintiva de la implementación del sistema. Los ADL's proporcionan tanto una sintaxis concreta como un marco conceptual para la caracterización de arquitecturas donde este refleje las características del dominio para los que el ADL está enfocado y/o el estilo arquitectónico.[15]

1.7.1.2. Lenguaje de modelado

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

El Lenguaje Unificado de Modelado (UML) describe un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos y describe además la semántica esencial de lo que estos diagramas y símbolos significan. Existen muchas notaciones y métodos usados para el diseño orientado a objetos, ahora el personal sólo tienen que aprender una única notación.

UML se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware y organizaciones del mundo real. UML ofrece nueve diagramas en los cuales modelar sistemas. Los objetivos de UML son muchos pero se centra en expresar de forma gráfica el sistema que se desee implementar de una forma entendible y logra especificar cada una de sus características. A partir de los modelos especificados se logran construir los sistemas diseñados y el diseño se reutilizaría como parte de la documentación del producto.[14]

Aunque UML es bastante independiente del proceso de desarrollo que se siga, los mismos creadores de UML han propuesto su propia metodología de desarrollo, denominada el Proceso Unificado de Desarrollo. Los aspectos que definen este Proceso Unificado son tres: Es iterativo e incremental, dirigido por casos de uso y centrado en la arquitectura.[16]

Dirigido por casos de uso: Basándose en los casos de uso, los desarrolladores crean una serie de modelos de diseño e implementación que los llevan a cabo. Además, estos modelos se validan para que sean conformes a los casos de uso. Finalmente, los casos de uso también sirven para realizar las pruebas sobre los componentes desarrollados.

Centrados en la arquitectura: En la arquitectura de la construcción, antes de construir un edificio éste se contempla desde varios puntos de vista: estructura, condiciones eléctricas y fontanería. Cada uno de estos aspectos está representado por un gráfico con su notación correspondiente.

Iterativo e incremental: Dividir un proceso en varias fases y ciclos de vida en los que se realizan varios recorridos por todas las fases. Cada recorrido por las fases se denomina iteración en el proyecto en la que se realizan varios tipos de trabajos (denominados flujos). Además, cada iteración parte de la anterior incrementando o revisando la funcionalidad implementada. Se suele denominar proceso.

Para cumplir con los objetivos trazados y darle solución a la problemática anteriormente descrita, se define como lenguaje de modelado UML, centrándose en las características relevantes anteriormente descritas. Permite desarrollar el trabajo fácil y organizado, brindando muchísimas comodidades en el uso de la programación orientada a objetos y permite un fácil entendimiento entre el equipo de desarrollo y el cliente.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.7.1.3. Lenguaje de Programación

En la actualidad existen varios lenguajes de programación que se pueden utilizar en el desarrollo del plugin del presente trabajo. Sin embargo, se decide utilizar el lenguaje de programación Java, atendiendo al hecho de que la herramienta AcmeStudio se basa en este lenguaje. Además de sus ventajas de ser un lenguaje orientado a objetos, multiplataforma, y presentar varias de las características que el equipo de desarrollo necesita que tenga el lenguaje seleccionado, las cuales son: sencillo, familiar, robusto, seguro y alto rendimiento.

Java es un lenguaje de programación por objetos creado por Sun Microsystems, que permite crear programas que funcionan en cualquier tipo de ordenador y sistema operativo. La sintaxis del lenguaje Java heredó características de C y C++, explícitamente eliminando aquellas que para muchos programadores (según los diseñadores) resultan excesivamente complejas e inseguras. En la actualidad su uso es promovido para el desarrollo de aplicaciones empresariales del lado del servidor, especialmente a través del estándar J2EE. En realidad, Java hace referencia a un conjunto de tecnologías entre las cuales el lenguaje Java es solo una de ellas.[17]

Características principales:

- Orientado a Objetos: Java organiza sus programas en una colección de objetos. Esto va a permitir estructurar los programas de una manera más eficiente y en un formato más fácil de comprender.
- Distribuido: Java dispone de una serie de librerías para que los programas se puedan ejecutar en varias máquinas y puedan interactuar entre sí.
- Robusto: Java está diseñado para crear software altamente fiable.
- Seguro: Java cuenta con ciertas políticas que evitan que se puedan codificar virus con este lenguaje, sin olvidar además que existen muchas otras restricciones que limitan lo que se puede o no se puede hacer con los recursos críticos de una máquina.
- Interpretado: La interpretación y ejecución se hace a través de la Máquina Virtual Java (JVM) es el entorno en el que se ejecutan los programas Java, su misión principal es la de garantizar la ejecución de las aplicaciones Java en cualquier plataforma.
- Independiente de la Arquitectura: El código compilado de Java se va a poder usar en cualquier plataforma.
- Multiejecución: Java permite elaborar programas que permitan ejecutar varios procesos al mismo tiempo sobre la misma máquina.[18]

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.7.2. Herramientas

Las herramientas de desarrollo de software desempeñan un papel fundamental en la creación de aplicaciones. Estas han experimentado en los últimos años grandes cambios y siguen avanzando en su desarrollo, evolucionando constantemente. Entre las principales corrientes del software se encuentra el software privativo y el software libre. El privativo es aquel que, por su uso, redistribución o modificación está prohibida, o requiere permiso expreso del titular del software. Software libre trata de mejorar sus herramientas y adaptarlas a las necesidades de cada persona que interviene en el proceso, sin costos por su uso, esto no implica que la adquisición del producto sea gratis.[14]

1.7.2.1. Visual Paradigm

Visual Paradigm es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, generar código desde diagramas y generar documentación. Es muy sencillo de usar, fácil de instalar y actualizar. Otra de las ventajas que posee es que se adecua al entorno de desarrollo permitiéndoles a los diseñadores, analistas y a todo aquel que trabaje con el mismo, una mayor organización y claridad en el trabajo, además presenta soporte multiplataforma. Esta herramienta es bastante común y muy utilizada en el mundo por la variedad de funciones que permite realizar, lo que posibilita su utilización para la evolución del proyecto catalogación y publicación de medias. Algunas ventajas que ofrece para el proyecto son:[19]

- Diseño centrado en casos de uso y enfocado al negocio, permitiendo generar un diseño de software de gran calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo facilitando la comunicación.
- Permite el control de versiones.
- Modelo y código permanecen sincronizados en todo el ciclo de desarrollo.
- Permite integrarse con diferentes IDE.

Se decide tomar esta herramienta como medio para modelar el proyecto ya que brinda grandes facilidades y cuenta con las características necesarias para el diseño y modelado del plugin a implementar. Además, en la universidad se cuenta con la licencia para su uso y tiene la característica de ser multiplataforma.

1.7.2.2. AcmeStudio

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

AcmeStudio es una herramienta de diseño arquitectónico que se ha desarrollado en la Carnegie Mellon University. Proporciona una interfaz gráfica que permite dibujar las arquitecturas de diferentes estilos y para manipular y analizar los diseños.

AcmeStudio se define como una herramienta capaz de soportar el mapeo de especificaciones arquitectónicas entre diferentes ADLs, o en otras palabras, como un lenguaje de intercambio de arquitectura. No es entonces un ADL en sentido estricto, aunque la literatura de referencia acostumbra tratarlo como tal. De hecho, posee numerosas prestaciones que también son propias de los ADLs. En su sitio oficial se reconoce que como ADL no es necesariamente apto para cualquier clase de sistemas, al mismo tiempo que se destaca su capacidad de describir con facilidad sistemas “relativamente simples”.

Objetivo principal: La motivación fundamental de Acme es el intercambio entre arquitecturas e integración de ADLs.[7]

Tipos de arquitectura que soporta AcmeStudio

AcmeStudio soporta la definición de cuatro tipos de arquitectura: la estructura (organización de un sistema en sus partes constituyentes); las propiedades de interés (información que permite razonar sobre el comportamiento local o global, tanto funcional como no funcional); las restricciones (lineamientos sobre la posibilidad del cambio en el tiempo); los tipos y estilos.

La estructura se define utilizando siete tipos de entidades: componentes, conectores, sistemas, puertos, roles, representaciones y mapas de representación.[7]

- **Componentes:** Representan elementos computacionales y almacenamientos de un sistema. Como se verá en el ejemplo, un componente se define siempre dentro de una familia.
- **Interfaces:** Todos los ADLs conocidos soportan la especificación de interfaces para sus componentes. En AcmeStudio cada componente puede tener múltiples interfaces. Igual que en Aesop y Wright los puntos de interfaz se llaman puertos (ports). Los puertos pueden definir interfaces tanto simples como complejas.
- **Conectores:** En su ejemplar estudio de los ADLs existentes, los lenguajes que modelan sus conectores como entidades de primera clase lenguajes de configuración explícitos, en oposición a los lenguajes de configuración in-line. AcmeStudio pertenece a la primera clase, igual que Wright y UniCon. Los conectores representan interacciones entre componentes. Los conectores también tienen interfaces que están definidas por un conjunto de roles. Los conectores binarios son los más

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

sencillos: el invocador y el invocado de un conector RPC, la lectura y la escritura de un conector de tubería, el remitente y el receptor de un conector de paso de mensajes.

- **Semántica:** Muchos lenguajes de tipo ADL no modelan la semántica de los componentes más allá de sus interfaces. En este sentido, AcmeStudio sólo soporta cierta clase de información semántica en listas de propiedades. Estas propiedades no se interpretan y sólo existen a efectos de documentación.
- **Estilos:** Acme posee manejo intensivo de familias o estilos. Esta capacidad está construida naturalmente como una jerarquía de propiedades correspondientes a tipos. Acme considera, en efecto, tres clase de tipos: tipos de propiedades, tipos estructurales y estilos. Así como los tipos estructurales representan conjuntos de elementos estructurales, una familia o estilo representa un conjunto de sistemas. Una familia Acme se define especificando tres elementos de juicio: Un conjunto de tipos de propiedades y tipos estructurales, un conjunto de restricciones y una estructura por defecto, que prescribe el conjunto mínimo de instancias que debe aparecer en cualquier sistema de la familia. El uso del término “familia” con preferencia a “estilo” recupera una idea de uno de los precursores tempranos de la arquitectura de software.

1.7.2.3. Entorno de desarrollo integrado

Un entorno o ambiente integrado de desarrollo Integrated Development Environment (IDE) es un programa compuesto por un conjunto de herramientas para un programador, que puede dedicarse a un solo lenguaje de programación o utilizarse para varios. Estos suelen incluir en una misma suite, un buen editor de código, administrador de proyectos y archivos, enlace transparente a compiladores y debuggers e integración con sistemas controladores de versiones o repositorios.

✓ **Eclipse**

Eclipse es un entorno integrado de desarrollo de código abierto y multiplataforma, creado originalmente por la International Business Machines (IBM) Canadá y actualmente desarrollado por la Fundación Eclipse. Este no es tan solo un IDE, se trata de un marco de trabajo modular ampliable mediante complementos (plugins). El mismo permite la realización tanto de aplicaciones web como de aplicaciones de escritorios y existen complementos que permiten usar Eclipse para programar en el lenguaje PHP, Perl, java, Python y C/C++. Provee soporte para Java y SVN¹.

¹ Subversion: Software de sistema de control de versiones.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Eclipse contiene una serie de perspectivas. Cada perspectiva proporciona una serie de funcionalidades para el desarrollo de un tipo específico de tarea. Por ejemplo la perspectiva Java combina un conjunto de views que permiten ver información útil cuando se está escribiendo código fuente, mientras que la perspectiva de depuración contiene vistas que muestran información útil para la depuración de los programas Java.

Entre sus características principales se encuentran:

- Dispone de un editor de texto con resaltado de sintaxis.
- La compilación es en tiempo real.
- Realiza pruebas unitarias.
- Mediante de su arquitectura de plugins permite añadir soporte para lenguajes adicionales, así como otras aplicaciones, tal es el caso de herramientas UML.[20]

1.8. Metodología de desarrollo de software

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software. En el que se van indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado, indicando además qué personas deben participar en el desarrollo de las actividades y qué papel deben de tener. Además, detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla.[12]

Como base para escoger una metodología que se ajuste a las necesidades específicas del trabajo, el tipo de aplicación y la finalidad del mismo, se realizó un estudio de las principales características de una de las metodologías ágiles usadas en la actualidad para el desarrollo de software.

1.8.1. Extreme Programming

Extreme Programin (XP por sus siglas en inglés) es una metodología ágil basada en la simplicidad, la comunicación e interacción permanente con el cliente (comprobación de requisitos constante) y en el “pair-programming”, que es la técnica de programación por parejas donde uno de los programadores escribe código y el otro lo prueba y después se cambian los papeles. De esta forma ya desde el principio se van probando los programas en cuanto a cumplimiento de requisitos como a funcionalidad. Esta metodología se caracteriza por:[12]

- Permite introducir nuevos requisitos o cambiar los anteriores ágilmente.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

- Adecuado para proyectos pequeños y medianos.
- Adecuado para proyectos con alto riesgo.
- Su ciclo de vida es iterativo e incremental.
- Cada iteración dura entre una y tres semanas.
- No produce demasiada documentación acerca del diseño o la planificación.

XP se basa fundamentalmente en ser ligero, cercano al desarrollo, basado en Historias de Usuario, fuerte comunicación con el cliente, el código pertenece a todos, programación por parejas y pruebas como base de la funcionalidad. Teniendo en cuenta las características y peculiaridades de las metodologías analizadas anteriormente y basándose en las particularidades del desarrollo del plugin a implementar se decidió utilizar XP-Programación Extrema, para el proceso de desarrollo y solución del problema a resolver ya que cuenta con diferentes características que son similares a las condiciones de trabajo y desarrollo. Las demás metodologías, aunque cabe destacar que son muy buenas y ampliamente utilizadas en el mundo, no se ajustan a las necesidades del trabajo y por ende no son las más aconsejables a manejar ya que traerían consecuencias negativas en las posteriores fases de desarrollo del producto.

Puntos claves: pesado, dividido en cuatro fases, las fases se dividen en iteraciones, se definen flujos de trabajo, los artefactos son el objetivo, se basa en roles, UML, muy organizativo, mucha documentación.

Teniendo en cuenta las características y peculiaridades de la metodología estudiada y basándose en las particularidades del desarrollo del plugin a implementar se decidió utilizar XP-Programación Extrema, para el proceso de desarrollo y solución del problema a resolver ya que cuenta con diferentes características que son similares a las condiciones de trabajo y desarrollo. Debido que el plugin a desarrollar no necesita mucha documentación y es un proyecto pequeño.

1.9. Conclusiones del capítulo

A lo largo de este capítulo se ha hecho referencia a la importancia que encierra una evaluación arquitectónica y al impacto que esta provoca durante el proceso de desarrollo de un software. Se han tratado además conceptos básicos muy relacionados con el tema, así como una serie de técnicas que pueden resultar muy útiles a la hora de evaluar un diseño arquitectónico.

Con la realización de este capítulo se arribaron a las siguientes conclusiones:

- La diagnosticabilidad es un atributo de calidad que desempeña un papel fundamental en los Sistemas de Gestión Integral. A medida que estos sistemas crecen y se hacen complejos, crecen

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

también las probabilidades de fallos, de ahí la importancia de una evaluación exitosa de la diagnosticabilidad.

- Los modelos de calidad son elementos esenciales en el proceso de evaluación, el modelo que guía la investigación es el modelo ISO/IEC 9126. Este modelo cuenta con un alto prestigio en el mundo, además de ser el definido por la Oficina Nacional de Normalización para la producción de sistemas de software.
- La evaluación de la diagnosticabilidad mediante métricas puede brindar una visión del estado de este atributo en la arquitectura y a su vez retribuir el proceso de desarrollo del software.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

2.1. Introducción

El presente capítulo se centrará en enfatizar y exponer la métrica de diagnosticabilidad para realizar la evaluación, con el fin de describir las prácticas que se adecúen al sistema, así como las características que no se adapten a este. En consecuencia, durante el desarrollo del capítulo se detalla el análisis desarrollado al modelado de la arquitectura del subsistema Inventario el cual fue la muestra seleccionada, para determinar los elementos significativos que pueden ayudar en el diagnóstico a desarrollar, en busca de la realización de un proceso de evaluación de la facilidad de prueba precisa y concreta. También se presentan los requisitos funcionales y no funcionales con los que debe cumplir el plugin a desarrollar. Se describe el modelo de conceptual para el plugin, las historias de usuario así como el procedimiento a ejecutar por el usuario para el uso de las funcionalidades de la herramienta.

2.2. Propuesta de solución

La obtención de un plugin que permita evaluar a través del atributo de calidad mantenibilidad y su subatributo diagnosticabilidad la arquitectura de los subsistemas de Cedrux, que forma parte del Sistema Integral de Gestión de Recursos Empresariales (ERP), Cedrux. Además permitirá fortalecer el proceso de desarrollo que se lleva a cabo en el centro CEIGE para el desarrollo del ERP Cedrux.

La presente tesis aportará:

- La especificación de la arquitectura del subsistema Inventario en el lenguaje Acme.
- La selección o creación de una métrica capaz de medir la facilidad de pruebas de la arquitectura de un subsistema.
- El desarrollo de un plugin que implementa la métrica seleccionada facilitando su uso en los distintos momentos que se necesite ponerla en práctica.

2.3. Modelo Conceptual

Dada la propuesta de la creación de un plugin para evaluar la diagnosticabilidad de la arquitectura en los subsistemas de Cedrux a través de su modelado en la herramienta AcmeStudio se propone la definición de un modelo conceptual el cual mostrará los principales conceptos a utilizar en el desarrollo de este complemento.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

El modelo conceptual contempla la representación de conceptos u objetivos que son importantes dentro de un problema. Estos conceptos representan y simbolizan objetos del mundo real que se encuentran dentro del sistema y ofrecen a su vez un entendimiento del problema en cuestión.

2.3.1. Representación del modelo de dominio

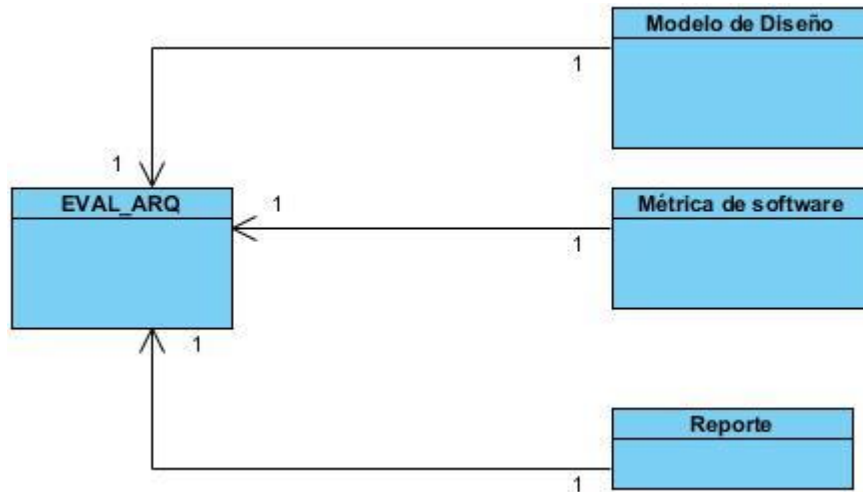


Ilustración 2. Modelo de Dominio

EVAL_ARQ: Plugin integrado a la herramienta “AcmeStudio”, el cual evalúa la facilidad de prueba que tiene una arquitectura a partir del modelado generado en la herramienta AcmeStudio.

Modelo del Diseño: Representa el modelo de la Arquitectura a la cual se le realizará las evaluación de la métrica.

Métrica de software: Métrica a ser utilizada en el plugin para efectuar la evaluación.

Reporte: Archivo generado por el plugin, el cual una vez que es evaluado el diseño se genera para su comprensión, ya que está conformado por información relevante respecto a la solución.

2.4. Desarrollo del modelado de la arquitectura en la herramienta AcmeStudio

Para el desarrollo de la solución se realizó el modelado de la arquitectura a través de los componentes por los que está compuesto el subsistema Inventario. Para la realización del modelado es necesario guiarse por la Matriz de Integración en su totalidad la cual contiene toda la información referente a los componentes que consumen y brindan los servicios. También es necesario conocer las pautas definidas en el estilo arquitectónico para CedruX propuesto en el documento: Estilo arquitectónico para el sistema integrado de gestión CedruX.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Un estilo arquitectónico en lo adelante (EA) típicamente define un vocabulario de tipos para componentes, conectores, interfaces y propiedades, además de un conjunto de reglas que rigen cómo los elementos de dichos tipos pueden componerse.[21]

A continuación se presenta una imagen de una parte de la Matriz de Integración del subsistema Inventario con algunos de los componentes que lo componen y las relaciones que existen entre ellos. Además de la tabla de clasificación de los componentes y conectores.

Componentes	Brindan			
	Lote	Documento	Configuracion	Despacho
Ajuste	LogLotInv06, LogLotInv05	LogDocInv100, LogDocInv26, LogDocInv126, LogDocInv45, LogDocInv127, EncabezadoAddAjuste, AdicionarDocAjuste, Setalias, ModificarDocAjuste, DevolverCronologia, ContabilizarDocAjuste, ConfirmarDocAjuste, EliminarDocAjuste, CancelarEstadoDocAjuste, AprobarDocAjuste, NoAprobarDocAjuste, AprobadoPorAjuste, creadoPor, getOperacion, EncabezadoModAjuste	LogConfInv01	LogDesInv01
Apertura	LogLotInv08	creadoPor, AprobadoPorDocInventario, desbloqueardocByldusuario, BuscarDocInventario, ObtenerDocInventario, AdicionarDocInventario, EncabezadoAdicionarApertura, EncabezadoModInventario, Setalias, ModificarApertura, EliminarDocInventario, verificarSalDOS, DevolverCronologia, AprobarDocInventario, NoAprobarDocInventario, GetDocumento, ConfirmarDocInventario, CancelarEstadoDoc, CambiarUsuario, BuscarDocFac	LogConfInv01, LogConfInv02	

Ilustración 3. Parte de la Matriz de Integración de subsistema Inventario[22]

Tabla 1. Clasificación de los componentes

Tipo Comp.	Componentes
Negocio	Apertura
Negocio	Recepción
Negocio	Baja
Negocio	Conciliación
Negocio	Inventario
Negocio	Ajuste
Negocio	Despacho

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Negocio	Generar Doc. Salida
Negocio	Combustible
Negocio	Submayor
Dominio	Recuperaciones
Dominio	Documentos
Dominio	Movimientos
Dominio	Productos
Negocio	Nro. Serie
Dominio	Acceso Datos
Negocio	Lote
Negocio	Nomencladores
Dominio	Configuración
Dominio	Útiles Municiones
Negocio	Cierre
Negocio	Generar Comprobante

Estilo Arquitectónico de Cedrux utilizados en el modelado de la arquitectura

Componentes: Los componentes son herramientas que automatizan las tareas y agrupan los objetos.

Negocio: Están directamente relacionados con funcionalidades de negocio dado que implementan los requerimientos del mismo.

Dominio: Implementan funcionalidades suplementarias para el negocio. Estos componentes pueden tener varias funciones. No obstante, no se dividen en otras categorías puesto que el comportamiento en el sistema es prácticamente el mismo. Su diferencia radica fundamentalmente en el tipo de información con que trabajan. Pero la forma de recibirla, brindarla, así como la manera en que la procesan es la misma. Su función fundamental es llevar a cabo cálculos o permitir configuraciones requeridas para realizar las funcionalidades de negocio. Estos cálculos y configuraciones tienen un comportamiento horizontal para el sistema, por lo que se encapsulan en componentes especializados.

Conector: Son los elementos que unen a los componentes.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

la toma de decisiones para aplicar reparaciones en el caso que sea necesario. La reparabilidad se ve afectada por la cantidad y tamaño de los componentes o piezas:

- Un producto software que consiste en componentes bien diseñados es más fácil de analizar y reparar que uno monolítico.
- Pero el incremento del número de componentes no implica un producto más reparable, ya que también aumenta la complejidad de las interconexiones entre los componentes.[23]

A la hora de evaluar una arquitectura de software es importante conocer el tamaño del sistema resultante, donde resulta probable que el tamaño y la complejidad del diseño estén directamente relacionados. A continuación se muestra y se explica una de las métricas de diagnosticabilidad que proporciona al diseñador una mejor visión interna y así el diseño evolucionará a un mejor nivel de calidad.

McCabe identifica un número importante de usos para las métricas de complejidad, donde pueden emplearse para predecir información sobre la fiabilidad y mantenimiento de sistemas software, también realimentan la información durante el proyecto de software para ayudar a controlar la actividad de diseño, en las pruebas y mantenimiento, proporcionan información sobre los módulos software para ayudar a resaltar las áreas de inestabilidad.[23]

Métrica: Facilidad de pruebas.

Esta se concentra en las características de la estructura del programa dándole énfasis a la estructura arquitectónica y en la eficiencia de los módulos. Card y Glass manifiestan que a medida que aumenten la cantidad de componentes y conectores, la complejidad arquitectónica o global del sistema también aumenta. Esto lleva a una mayor probabilidad de que aumente el esfuerzo necesario para la integración y las pruebas (21). Por lo que se define una métrica para evaluar la arquitectura de los subsistemas de Cedrux a través de la característica de diagnosticabilidad de la forma:

$$X = \frac{1}{(CantComp + CantConec) * CS}$$

Donde X es la capacidad de análisis de fallos, CantComp es la cantidad de componentes por la que está compuesta la arquitectura del subsistema, CantConect es la cantidad total de conectores y ComplSist es la complejidad del sistema, que se calcula a través de la unión de varias fórmulas que se expondrán a continuación.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Una causa de problemas son los fallos en los componentes el origen último de este tipo de fallos es muchas veces difícil de determinar. En algunas ocasiones, los fallos en los componentes pueden deberse a errores en el dimensionamiento de los mismos, que provocan su sobrecarga y su posterior destrucción. Otras veces es la corrosión o el desgaste natural el que provoca fallos en los componentes; por otra parte, hay componentes defectuosos en origen, pero cuyo defecto no se manifiesta hasta pasado cierto tiempo de funcionamiento.[23]

Para determinar la complejidad del sistema se tienen en cuenta un conjunto de elementos: tamaño, complejidad y criticidad de cada uno de los componentes que lo forman. Para ello se medirán los siguientes aspectos:

- Cantidad de servicios que consume el componente.
- Complejidad promedio de los servicios que se brinda.
- Cantidad de conectores del diseño.
- Complejidad promedio de los conectores.
- Cantidad de servicios que brinda el componente.
- Cantidad de componentes dependientes del mismo.

Con todos estos datos se pueden calcular las tres propiedades a medir mencionadas con anterioridad:[24]

Tamaño del componente:

$$TC = \frac{Kr * 2 + Ks}{100}$$

Donde: TC es el tamaño del componente, Kr la cantidad de servicios que consume el componente, Ks cantidad de servicios que brinda el componente.

Complejidad del componente:

$$CoC = \frac{Cr * 2 + TC}{100}$$

Donde: Coc es la complejidad del componente, Cr es la complejidad promedio de servicios que se consume, TC es el tamaño del componente.

Criticidad del Componente:

$$CrC = \frac{Kcd * 2 + \sum KcdD}{100}$$

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Donde: Crc es la criticidad del componente, Kcd es la cantidad de componentes dependientes directos, KcdD es la cantidad de componentes dependientes de los dependientes directos del componente.

Complejidad del sistema:

$$CS = \frac{\sum C_{oc}}{CantComp}$$

CS: Complejidad del Sistema.

Para la Interpretación del valor obtenido va a estar contenido en un intervalo: $0 \leq X \leq 1$ y mientras más cercano al 1, mejor.

Si el valor de $0 < X \leq 0.3$ implica una arquitectura difícil de probar y se le deben hacer modificaciones.

Si el valor de $0.4 \leq X \leq 0.6$ implica una arquitectura de una complejidad media.

Si el valor de $0.7 \leq X \leq 1$ implica una arquitectura fácil de probar.

Se puede determinar y comparar el valor de X de diseños anteriores con un diseño actualmente en desarrollo. Si X es significativamente inferior a la medida, se aconseja seguir trabajando en el diseño y revisión. Igualmente, si hay que hacer grandes cambios a un diseño ya existente, se puede calcular el efecto de estos cambios en X.[23]

2.5. Requisitos del software

Los requisitos del software forman parte de la documentación asociada al software que se está desarrollando, por tanto, se deben definir correctamente todos los requisitos, pero no más de los necesarios. Esta documentación no debería describir ningún detalle de diseño, modo de implementación o gestión del proyecto, ya que los requisitos se deben describir de forma que el usuario pueda entenderlos. Al mismo tiempo, se da una mayor flexibilidad a los desarrolladores para la implementación.[14]

2.5.1. Requisitos Funcionales

Los requisitos funcionales se definen como capacidades o condiciones que el sistema debe cumplir, o sea, son las características que el cliente solicita del sistema y espera que cumpla el mismo, en vista de solucionar su problema o conseguir su objetivo.

La herramienta que se pretende construir debe cumplir los siguientes requisitos funcionales:

- **RF1.Calcular la diagnosticabilidad.**

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Este requisito a través del modelado devuelve un valor positivo, un resumen del tamaño, criticidad y complejidad de los componentes los cuales tienen una suma importancia para la toma de decisiones de la arquitectura de software y para la retroalimentación del proceso de desarrollo.

➤ **RF2.Exportar reporte detallado.**

Devuelve la fecha en que se realizó la evaluación y además todas las funciones que se realizaron para evaluar la arquitectura, para que los arquitectos tomen sus decisiones con respecto a la evaluación emitida.

2.5.2. Requisitos no Funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe cumplir. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Después de determinar lo que el software debe hacer debe establecerse cómo el mismo ha de comportarse y cuáles serán sus cualidades.[20]

Son aquellas exigencias de cualidades que se imponen al proyecto: exigencias de usar un cierto lenguaje de programación o plataforma tecnológica, por ejemplo. Un requisito no funcional es una característica ya sea del sistema, del proyecto o del servicio de soporte, que nos es requerida junto con la especificación del sistema pero que como ya dije, no se satisface añadiendo código, sino cumpliendo con esta como si de una restricción se tratara.

Software: Para ejecutar y correr el programa solo se necesita contar con una computadora con la herramienta AcmeStudio instalada.

Hardware: Como requerimiento mínimo el sistema debe contar con procesador Pentium II en adelante con no menos de 256 de RAM. Se recomienda para su funcionamiento óptimo máquinas con procesadores Pentium IV y 512 de RAM.

Usabilidad: Facilidad de uso por parte de los usuarios: La interfaz debe ser lo más descriptiva posible, permitiendo que las operaciones a realizar por los usuarios estén bien descritas, de manera que se puedan entender claramente.

Apariencia o interfaz: La interfaz deberá ser sencilla de usar, amigable y aunque su ámbito principal es para el desarrollo de software, cualquier usuario podría hacer uso de esta sin dificultad.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Seguridad: El plugin debe mantener la integridad y confiabilidad ante la información manejada, ya que la ausencia de estos aspectos puede conllevar a grandes riesgos pues se ve afectado el manejo de los datos de las pruebas en el laboratorio. Para lograr lo antes mencionado se definirán roles y el acceso a la aplicación será según los permisos con que cuente cada rol definido, evitando así acciones que se realicen no autorizadas.

Confiabilidad: El plugin deberá tener un 100% de disponibilidad por lo que podrá ser usado siempre.

Requerimientos del diseño e implementación:

Como característica específica se utilizarán estas herramientas en las fases de construcción del software:

- Se debe implementar en el lenguaje Java.
- Para el modelado de la arquitectura se usará el lenguaje de descripción arquitectónica ADL y como herramienta para llevar a cabo el modelado AcmeStudio
- Para el análisis y el diseño del plugin se utilizará la metodología XP, usando el lenguaje de modelado UML y como herramienta para llevar a cabo el modelado Visual Paradigm.
- Para la implementación del componente se utilizará el IDE Eclipse.

2.6. Modelado del sistema

Se muestran los actores del sistema además de la representación de las historias de usuario definidas en la metodología XP y su análisis correspondiente.

Tabla 2. Actores del sistema

Nombre del Actor	Descripción
Probador	Estos usuarios pueden configurar todos los datos necesarios para la gestión exitosa de las pruebas.

2.6.1. Lista de reserva del producto

La tabla que a continuación se muestra, relaciona las Historias de Usuario con la prioridad que tienen, la estimación del tiempo para efectuarlas en días y los usuarios que se encargan de desarrollarlas.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Tabla 3. Lista de reserva del producto

Prioridad	Ítem	Descripción	Estimación (días)	Estimado por:
Alta	1	Analizar el estilo arquitectónico por el cual se determinará el tipo de componente, conectores y restricciones.	3	PROB
Alta	2	Modelar la arquitectura del software en la herramienta AcmeStudio.	5	PROB
Alta	3	Extraer la información del modelo realizado en la herramienta ACME	2	PROB
Alta	4	Calcular la diagnosticabilidad	10	PROB
Alta	5	Mostrar los resultados obtenidos	5	PROB

2.6.2. Descripción de las historias de usuarios

En este epígrafe se realiza una descripción de las Historias de Usuarios y las tareas ingenieriles que cada una tiene asignada además del responsable del cumplimiento de dicha tarea.

Tabla 4. HU1: Calcular la diagnosticabilidad

Historia de Usuario	
Código: HU1	Nombre Historia de Usuario: Calcular la diagnosticabilidad
Modificación de Historia de Usuario Número: ninguna	
Referencia: RF1	
Programador: Greter Encinosa Hernández	Iteración Asignada: primera
Prioridad: Alta	Puntos Estimados: 10 días
Riesgo en Desarrollo: Alta	Puntos Reales: 2 semanas

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Descripción: Calcula matemáticamente la diagnosticabilidad, de los subsistemas de Cedrux, utilizando la métrica Facilidad de prueba a partir de la obtención del modelo de la arquitectura con AcmeStudio, para obtener valores cuantitativos y saber qué tan fácil de probar es la arquitectura de dicho subsistema.

Observaciones:

Prototipo de interface:

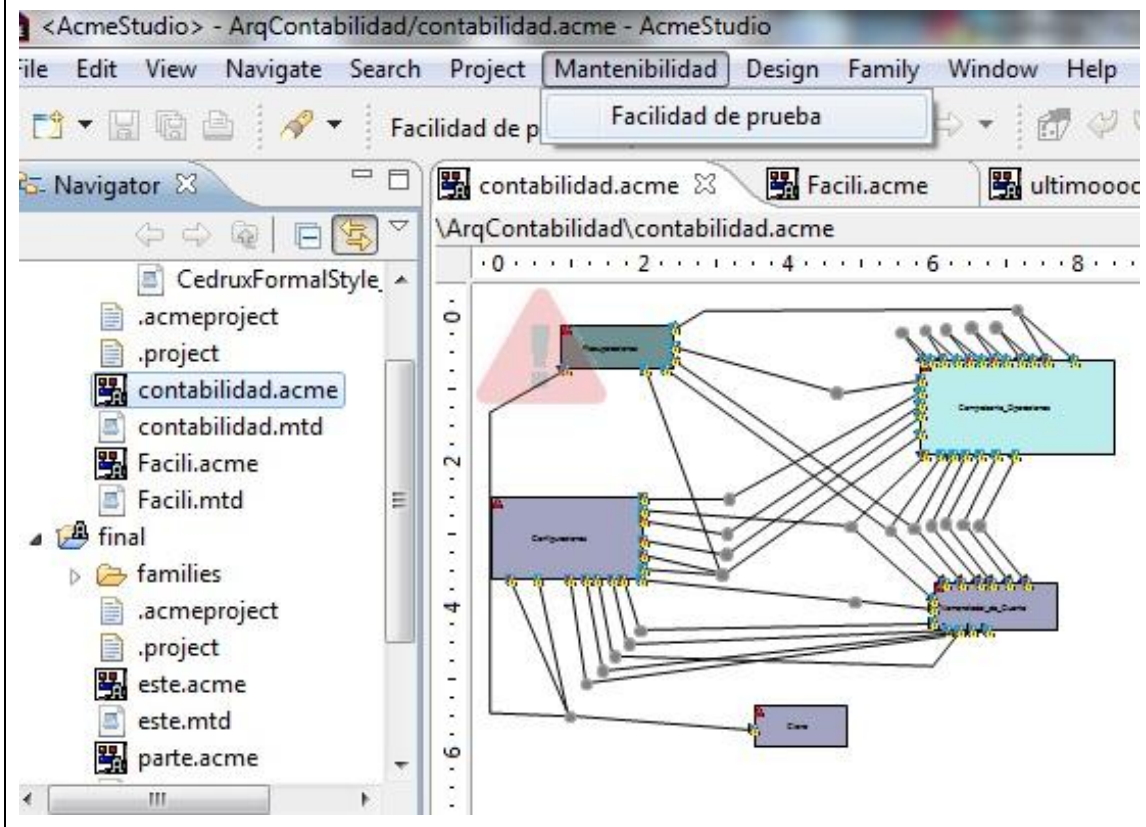
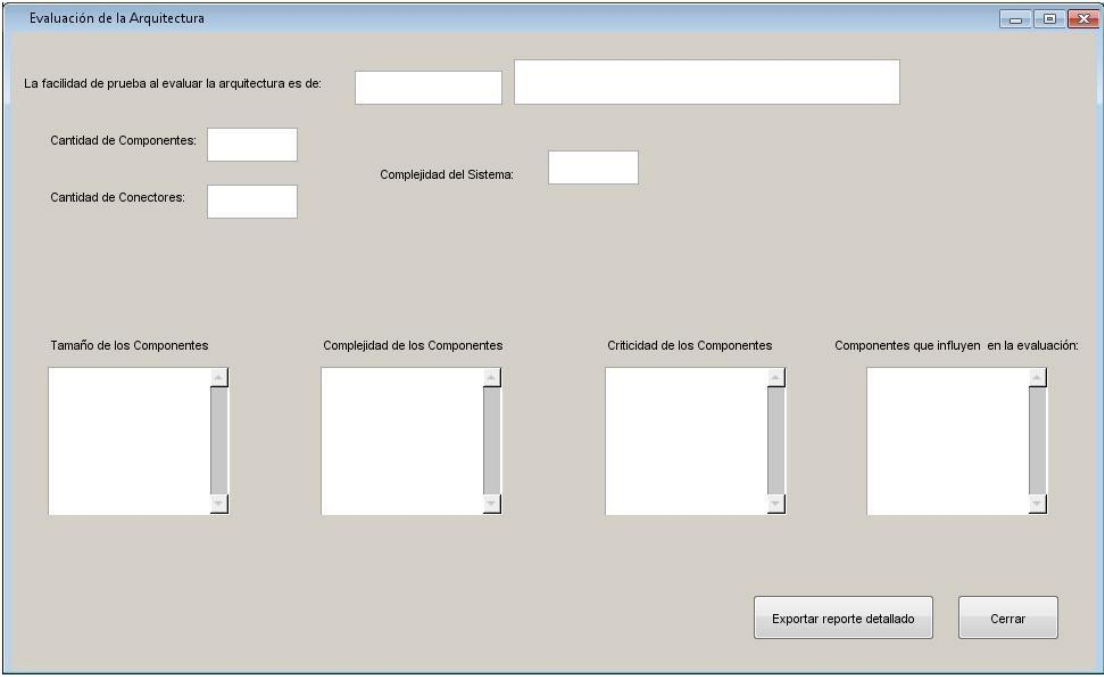


Tabla 5. HU2.Exportar los resultados obtenidos

Historia de Usuario	
Código: HU2	Nombre Historia de Usuario: Exportar los resultados obtenidos
Modificación de Historia de Usuario Número: ninguna	

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Referencia: RF2	
Programador: Greter Encinosa Hernández	Iteración Asignada: primera
Prioridad: Alta	Puntos Estimados: 10 días
Riesgo en Desarrollo: Alta	Puntos Reales: 2 semana
Descripción: El sistema genera un reporte constituido por las evaluaciones referentes a la métrica que fue utilizada en la evaluación, donde se muestran los valores como: El tamaño, complejidad, criticidad de los componentes y la facilidad de prueba del subsistema en general.	
Observaciones:	
Prototipo de interface:	
	

2.7. Modelo de Diseño

El diseño constituye una representación ingenieril significativa en la construcción del software. Es un proceso que tributa en demasía al proceso de desarrollo de software para transformar los requisitos de los

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

usuarios en un producto de software finalizado. Durante esta fase, cuando la arquitectura es estable y los requisitos están bien entendidos, el centro de atención se desplaza a la implementación siendo de vital importancia la especificación de la estructura del sistema, la cual es definida a través del modelo de diseño.

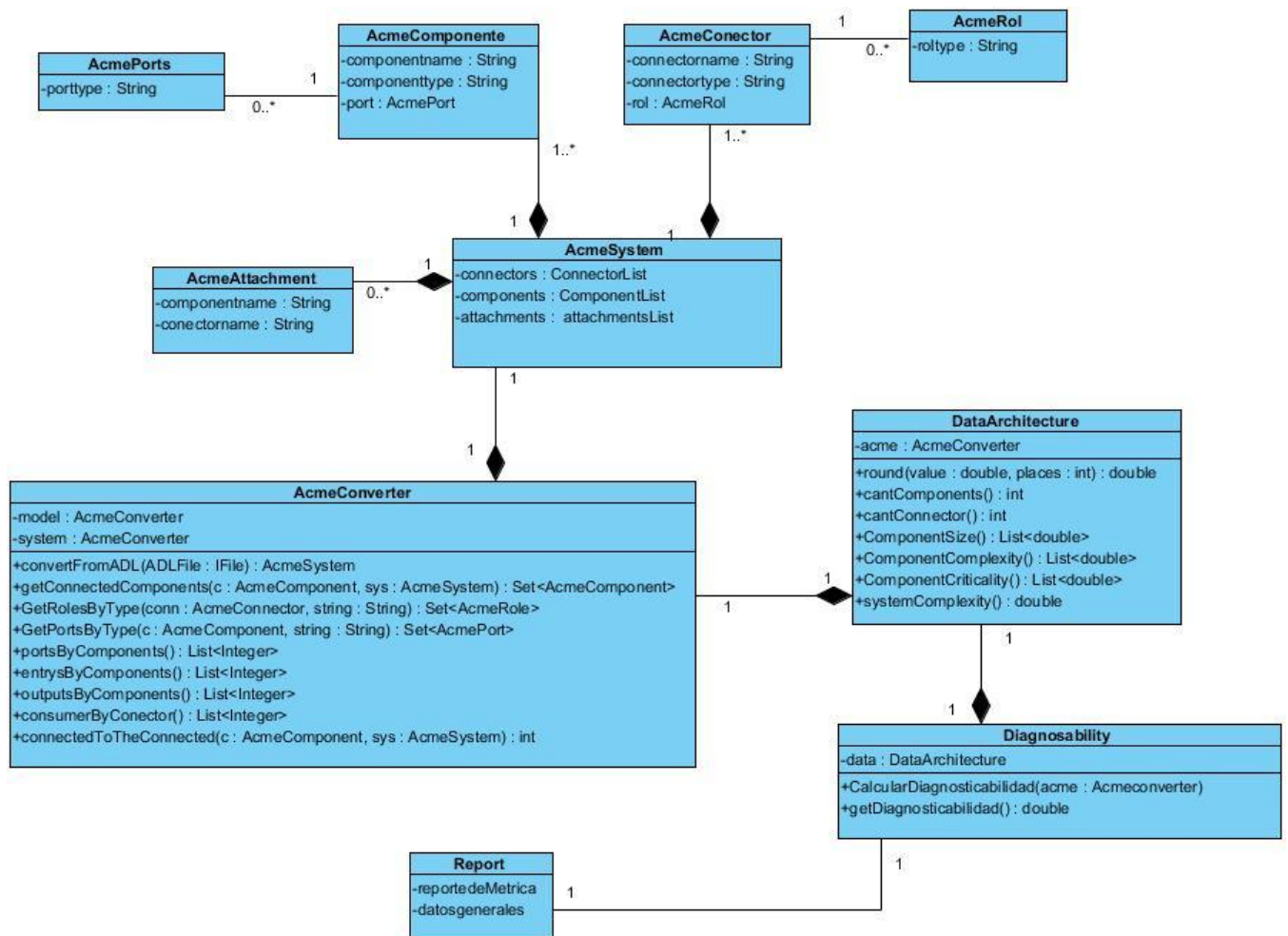


Ilustración 5. Diagrama de Clases del Sistema

Tabla 6. Descripción de las Clases del Sistema

Clase	Descripción
AcmeSystem	Es la clase que contiene todo lo referente a los componentes, conectores, puertos, rol y attachment. Esta clase es generada por la herramienta AcmeStudio luego de haber modelado la arquitectura.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

AcmeConverter	Es la clase que permite la captura en el contexto de la acción de realizar la evaluación y contiene la mayoría de los métodos necesarios para la evaluación.
Diagnosability	Es la clase que mantiene el control del tamaño, criticidad, complejidad, criticidad y complejidad del sistema.
DataArchitectura	Es la clase que mantiene el control de todas las acciones que se realizan. Clase que contiene la métrica.
Report	Formulario que contiene y manipula los datos del reporte a generar una vez que se realiza la evaluación.
AcmeRol	Clase que indica el tipo de rol.
AcmeConnector	Posee la información referente a los conectores, nombre, tipo y el rol representado.
AcmeComponent	Posee la información referente a los componentes, nombre, tipo y el puerto representado.
AcmePort	Indica el tipo de puerto.
AcmeAttachment	Clase encargada proporcionar elementos de la relación entre un componente y un conector.

2.7.1. Patrones de asignación de responsabilidades utilizados en el diseño

Para la realización del diseño de la aplicación informática a implementar, se utilizaron los patrones GRASP, estos son Patrones Generales de Software para Asignación de Responsabilidades. Se considera que más que patrones son una serie de "Buenas Prácticas" de aplicación recomendable en el diseño de software. Entre los patrones GRASP se encuentran los siguientes:

Controlador: Este patrón funciona como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos y la que los envía a las distintas clases según el método llamado.

Experto: El GRASP de Experto en información es el principio básico de asignación de responsabilidades. Indica que la responsabilidad de la creación de un objeto debe recaer sobre la clase que conoce toda la información necesaria para crearlo, lo cual permite que se conserve el encapsulamiento, soportando un bajo acoplamiento y una alta cohesión.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Creador: Ayuda a identificar quién debe ser el responsable de la creación de nuevos objetos o clases. La nueva instancia deberá ser creada por la clase que: tiene la información necesaria para realizar la creación del objeto, usa directamente las instancias creadas del objeto, o almacena o maneja varias instancias de la clase. Este patrón brinda soporte de bajo acoplamiento, lo cual supone menos dependencias entre clases y posibilidades.

El patrón Experto y Creador se ponen de manifiesto en la clase AcmeConverter porque es la encargada de la creación de las instancias de las diferentes clases utilizadas en la realización de la evaluación.

Alta cohesión y bajo acoplamiento: Se pueden separar, aunque están íntimamente ligados, de hecho si se aumenta mucho la cohesión del sistema software, se tiene un alto acoplamiento entre las clases y por el contrario si reduce mucho el acoplamiento, se verá mermada la cohesión.

Alta cohesión: Este patrón propone asignar la responsabilidad de manera que la complejidad se mantenga dentro de límites manejables asumiendo solamente las responsabilidades que deben manejar, evadiendo un trabajo excesivo. Su utilización mejora la claridad y facilidad con que se entiende el diseño, simplifica el mantenimiento y las mejoras de funcionalidad, generan bajo acoplamiento, soporta mayor capacidad de reutilización.

Este patrón se pone de manifiesto en la mayoría de las clases porque cada una es capaz de realizar sus responsabilidades sin la utilización de las demás.

Bajo acoplamiento: Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización y disminuyendo la dependencia entre las clases.[25]

2.8. Conclusiones del capítulo

Luego del estudio del presente capítulo se expusieron:

- Los conceptos esenciales en el contexto del sistema a través del modelo de dominio, arrojando con posterioridad a una propuesta de solución transcurriendo por 2 procesos esenciales tales como “Cálculo de la métrica y Evaluación de métricas”.
- Se realizó el levantamiento de requisitos del sistema concluyendo con la identificación de 2 requisitos funcionales agrupados por las Historias de Usuarios.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

- Luego de un análisis de los requisitos identificados se obtuvo el modelo de diseño como vía principal para visualizar las relaciones entre las clases involucradas en el sistema.
- Fueron definidos para la implementación del plugin los patrones Generales de Asignación de Responsabilidades (Grasp).

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

CAPÍTULO 3: VALIDACIÓN Y PRUEBAS

3.1. Introducción

En el presente capítulo se exponen el proceso de validación a través del criterio de expertos y las pruebas realizadas al software desarrollado, con el objetivo de comprobar las funcionalidades del plugin en los diferentes escenarios, verificando en todos los casos que los resultados de las pruebas sean los esperados.

Las pruebas tienen gran importancia en el desarrollo de un software, ya que mediante estas se pueden detectar y corregir errores tempranamente. Antes de entregar el producto al cliente final se debe garantizar que el software cumpla con todos los requerimientos y se han corregido todos los errores, pues cada vez que el programa se ejecuta, el cliente lo está probando, por tanto, debemos hacer un intento especial para que se sienta satisfecho.

3.2. Importancia del desarrollo de la propuesta de solución

La arquitectura de software es un artefacto fundamental dentro del desarrollo de sistemas de calidad. El no cuidar aspectos relacionados con el desarrollo de la arquitectura puede resultar en sistemas que no cubren las expectativas de los clientes y de la organización de desarrollo; la evaluación del diseño de la arquitectura es, por lo tanto, una actividad fundamental dentro de las actividades de desarrollo. El alto costo que tienen los defectos relacionados con la arquitectura en etapas tardías del desarrollo justifica plenamente que se invierta en la realización de esta práctica como parte del desarrollo. Con el desarrollo del plugin para evaluar cuantitativamente la diagnosticabilidad de los subsistemas de Cedrux a partir del modelado de sus arquitecturas en la herramienta AcmeStudio. A la hora de la toma de decisiones con respecto al rediseño de la arquitectura, el tiempo y los costos para el desarrollo del subsistema disminuiría satisfactoriamente, permitiendo así cumplir con el tiempo pactado con el cliente.

También se permitirá una retroalimentación en el proceso de desarrollo de las evaluaciones sucesivas, donde estas tendrán presente el resultado de las evaluaciones pasadas. El plugin desarrollado ayuda en la toma de decisiones con respecto al rediseño de la arquitectura ya que emite y permite guardar un reporte que contiene toda la evaluación realizada y la fecha en que se realizó. Enfatizando en los componentes que influyen notoriamente en el análisis.

3.3. Proceso de validación de la métrica

Métodos de expertos

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

Desde el momento en el que se procede al desarrollo de una investigación, el principal problema que surge es la posibilidad de comprobar y demostrar la fidelidad de la propuesta resultante. Para erradicar este problema es que se crea el Método de expertos, estos se basan en la consulta a un grupo de expertos para la validación de la propuesta. Los métodos de expertos tienen las siguientes ventajas:

Se basa en la suposición de que varios expertos pueden llegar a un mejor pronóstico que una sola persona. El experto se siente involucrado plenamente en la solución del problema y facilita su implantación. De ello es importante el principio de voluntariedad del experto en participar en la investigación.

De igual manera este método también posee desventajas y estas son:

- No siempre el argumento más válido es el que triunfa, en ocasiones es el más citado.
- Estos grupos son vulnerables a la posición y personalidad de algunos de los individuos.
- La presión social que el grupo ejerce sobre sus participantes puede provocar acuerdos con la mayoría, aunque la opinión de esta sea errónea. Así, un experto puede renunciar a la defensa de su opinión ante la persistencia del grupo en rechazarla.[26]

3.4. Elección de expertos

Se considerará un experto aquel individuo, grupo de personas u organizaciones que sean capaces de brindar valoraciones conclusivas sobre la propuesta, además de hacer recomendaciones con un determinado coeficiente de competencia. Los mismos no solo deben ser conocedores del tema, sino que deben presentar gran diversidad en sus planteamientos. Los criterios que se tuvieron en cuenta para la selección de los posibles expertos son reflejados a continuación:

- Graduado de nivel superior.
- Más de 2 años de experiencia.
- Conocimientos sobre proceso de pruebas.
- Que esté vinculado al desarrollo del software.
- Que tenga conocimientos medio sobre la evaluación de la arquitectura de software.

Una vez conocidos los requisitos a cumplir por los expertos, se realiza una encuesta de autoevaluación para medir la fuente y el grado de conocimiento de cada uno, con el objetivo de seleccionar los que conozcan más del tema que se está investigando. (Ver Anexo5)

Para seleccionar los expertos se tiene en cuenta la valoración de sus competencias, es por esta razón que se hace necesario calcular el coeficiente de competencia (**K**) que es basado en los resultados obtenidos en las encuestas aplicadas anteriormente. Su objetivo es demostrar si el nivel de conocimientos que

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

poseen los expertos así como las fuentes de argumentación son las adecuadas, de manera tal que garantice la confiabilidad en los resultados.

El coeficiente de competencia se calcula mediante la fórmula: $K = 0.5 (Kc + Ka)$, donde **Kc** es el coeficiente de conocimientos y **Ka** el coeficiente de argumentación. El **Kc** es la información que posee la persona acerca del problema (sobre la base de su autovaloración, es el resultado de la primera pregunta de la encuesta); sus valores están en una escala de 0 a 10, que para ajustarla a la teoría de las probabilidades se multiplica por 0.1, el cero indica que la persona no posee absolutamente ningún conocimiento de la problemática en estudio, mientras que el 10 expresa pleno conocimiento.

El coeficiente de argumentación o fundamentación de los criterios de la persona se obtiene de la siguiente forma $Ka = \Sigma$ de los valores que se obtienen de sustituir las cruces del (Anexo6) por los valores correspondientes en su posición de la tabla de valores para el coeficiente de argumentación.[26]

Una vez calculado el **Kc** y el **Ka**, se puede calcular **K** mediante la fórmula descrita anteriormente. El coeficiente **K**, teóricamente, se encuentra siempre entre 0.25 y 1. Mientras más cercano esté el valor de **K** a uno, mayor es el grado de competencia de la persona. Este resultado se debe interpretar a través de la siguiente escala brindada por el método Delphi:[26]

- Si $0.8 < K < 1.0$, el coeficiente de competencia es alto.
- Si $0.5 < K < 0.8$, el coeficiente de competencia es medio.
- Si $K < 0.5$, el coeficiente de competencia es bajo.

Para obtener mejores resultados en la aplicación del Delphi es conveniente utilizar los expertos cuyo coeficiente de competencia sea alto o medio.

Luego de realizada la encuesta de autoevaluación a los posibles expertos, solo 5 fueron seleccionados para formar parte del grupo de validación de la propuesta, pues fueron aquellos cuyos resultados arrojaron un coeficiente de competencia alto y medio (Tabla 7). Los resultados del análisis del coeficiente de competencia se muestran a continuación:

Tabla 7. Coeficiente de expertos

Experto	Ka	Kc	K	Competencia
1	0.5	0.8	0.65	Medio
2	0.8	0.9	0.85	Alto
3	0.7	0.6	0.65	Medio

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

4	0.7	0.8	0.75	Alto
5	0.8	0.8	0.8	Alto

En el (Anexo 7) se observa un grafo donde se detalla el porcentaje del coeficiente de competencia de los expertos.

3.5. Elaboración y lanzamiento del cuestionario

Una vez seleccionados los expertos, se elabora el cuestionario de validación de la propuesta, que tiene como objetivo principal la validación de los elementos fundamentales que conforman la propuesta. El mismo cuenta con cuatro preguntas orientadas a evaluar aspectos críticos de la solución, el cual fue realizado por los cinco expertos seleccionados pidiéndosele que evaluaran en las categorías ordinales de Bastante Adecuado (BA), Adecuado (A) y No Adecuado (NA). Donde la categoría de BA sería el valor más alto, la categoría A sería un valor medio y la categoría NA sería el valor más bajo. Con las respuestas dadas por los expertos se podrá obtener la concordancia entre ellos, la aceptación y validez de la propuesta. El cuestionario de validación utilizado en la investigación se muestra en el (Anexo 8).

Establecimiento de la concordancia entre los expertos

Cuando se tienen datos de tipo ordinal el coeficiente de Kendall, el cual toma en consideración el orden, es usualmente un instrumento estadístico muy apropiado para indicar el grado de asociación de las evaluaciones ordinales hechas por evaluadores múltiples cuando se evalúa la misma muestra. Los valores del coeficiente deben oscilar entre 0 y 1. Entre mayor sea el valor Kendall, más fuerte será la asociación. Un coeficiente Kendall (**W**) significativo implica que los evaluadores están aplicando esencialmente el mismo estándar cuando evalúan la muestra.

Valores para calcular el coeficiente de Kendall.

K Es el número de expertos que intervienen en el proceso de validación, por tanto va ser 5.

N Cantidad de preguntas a validar. En este caso N = 4.

R_j Es la suma de los rangos asignados a cada pregunta por parte de los expertos.

\bar{R}_j Es la media de los rangos y se determina mediante la fórmula.

Ya con todos estos datos se puede decir que **W** se calcula mediante la siguiente fórmula:

$$W = \frac{12 * S}{k^2(N^3 - N)}$$

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

Donde **S** es la suma de los cuadrados de las desviaciones observadas de la media de **R_j** (rangos) y se calcula:

$$S = \sum_{j=1}^n (R_j - \bar{R}_j)^2 \quad \bar{R}_j = \frac{\sum_{j=1}^n R_j}{N}$$

Y

es la suma de los rangos dividido entre la cantidad de preguntas

A continuación se muestran los cálculos realizados para determinar la concordancia de los expertos:

Tabla 8. Datos obtenidos de los expertos

	Exp1	Exp2	Exp3	Exp4	Exp5	R _j
P1	3	3	2	3	2	13
P2	2	3	1	3	2	11
P3	2	3	3	3	3	14
P4	3	3	2	2	3	13

Sustituyendo los valores en la ecuación para calcular la media de los rangos se obtiene el valor $51/4 = 12,75$.

Luego se determina la desviación media $S = ((13-12,75)^2 + (11-12,75)^2 + (14-12,75)^2 + (13-12,75)^2) = 4,75$

Sustituyendo los valores en la ecuación Kendall se obtiene el valor de $W = 0,038 \cdot 12 \cdot 4,75 / 5^2 (4^3 - 4) = 57/25(60) = 0,038$

El coeficiente de Kendall obtenido permite calcular el Chi cuadrado real, el cual tiene el objetivo de medir si existe o no concordancia entre los expertos y se obtiene a través de la fórmula:

$$X^2 = K (N - 1) W = 5(4-1) 0,038 = 0,57$$

Después de calcular el Chi Cuadrado se procede a comparar el valor con el de la tabla estadística (Anexo 9). Si se cumple que:

$$X_{real}^2 < X^2_{(\alpha, N-1)}$$

entonces quiere decir que existe concordancia entre los expertos.

Teniendo en cuenta la probabilidad de error de un 0,05 se realizar los cálculos:

$$X_{real}^2 < X^2_{(\alpha, N-1)}$$

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

$0,57 < X_2(0.05, 3) = 0,57 < 7,8147$ lo cual afirma el cumplimiento de la comparación y por tanto la concordancia entre los expertos.

3.6. Desarrollo práctico y análisis de los resultados

Los expertos seleccionados recibieron un resumen de la propuesta como documentación para contestar los criterios encuestados, de igual manera recibieron la encuesta con un total de 4 preguntas, estas documentaciones fueron entregadas personalmente o enviadas por vía e-mail, además se realizó una sola ronda de encuesta y luego se procedió al análisis de los resultados.

Paso 1: Se realiza la construcción de la siguiente tabla de frecuencias absoluta en la que se encuentran los resultados obtenidos de las encuestas evaluadas en las categorías de bastante adecuado (BA), adecuado (A) y no Adecuado (NA).

Tabla 9. Frecuencia absoluta de las preguntas de las encuesta

Tabla de frecuencias absolutas				
No.	BA	A	NA	Total
1	3	2	0	5
2	2	2	1	5
3	4	1	0	5
4	2	3	0	5
Total	11	8	1	

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

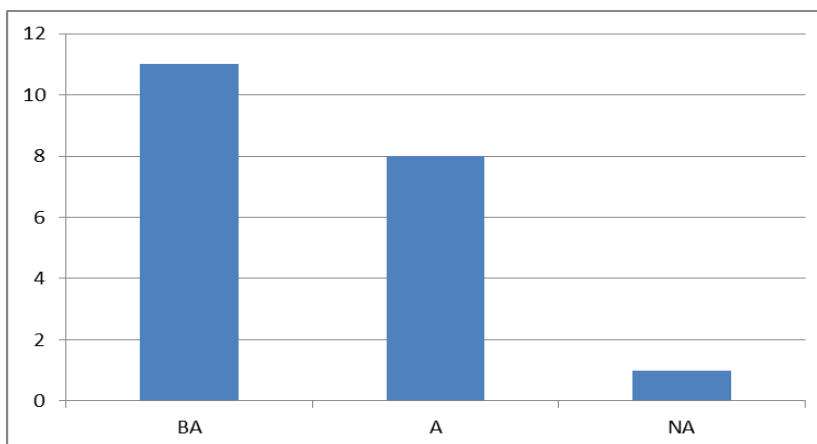


Ilustración 6. Análisis de las frecuencia absoluta de las preguntas de la encuestas

Paso 2: Construir una tabla de frecuencias absoluta acumulada. Esto se realiza la sumándole a cada número, el numero anterior de la fila, exceptuando al primer número de la fila.

Tabla 10. Frecuencia absoluta acumulada

Tabla de frecuencias absolutas acumuladas			
No.	BA	A	NA
1	3	5	5
2	2	4	5
3	4	5	5
4	2	5	5

Paso 3: Partiendo de tabla anterior se construye una nueva pero de frecuencias relativas acumuladas. En esta, los datos se obtienen mediante la división de los valores numéricos de la tabla de frecuencia absoluta acumulada entre el número total de expertos, en este caso serian cinco.

Tabla 11. Frecuencia relativa acumulada.

Tabla de frecuencias relativa acumulada			
No.	BA	A	NA

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

1	0.6	1	1
2	0.4	0.8	1
3	0.8	1	1
4	0.4	1	1

Paso 4: Finalmente, se buscan las imágenes de los elementos de la tabla anterior por medio de la función. Estas imágenes se representan en la misma tabla anterior, sólo que se le adicionan 3 columnas y una fila para representar los siguientes resultados:

Suma de las columnas: Esta nueva fila recoge en cada una de sus celdas la suma de todos los valores de la columna correspondiente.

Suma de las filas: Esta nueva fila recoge en cada una de sus celdas la suma de todos los valores de la fila correspondiente.

Para hallar el promedio General (N): Se divide la suma de las sumas de las filas (la cual tiene que ser igual a la suma de las sumas de las columnas) entre el resultado de multiplicar el número de aspectos que se están evaluando por el número de preguntas, en este caso se divide por 2 porque quedan sólo dos categorías, ya que la última ha sido eliminada. $N = 6 / 4 * 2 = 0,75$

Tabla 12. Punto de cortes

Puntos de cortes				N=0,75	
No.	BA	A	Suma	P	N-P
1	0,6	1	1,6	0,8	0,05
2	0,4	0,8	1,2	0,6	-0,15
3	0,8	1	1,8	0,9	0,15
4	0,4	1	1,4	0,7	-0,05
Suma	2,2	3,8	6		
Puntos de corte	0.55	0.95			

La suma obtenida de las 2 primeras columnas da los puntos de corte. Estos puntos de corte se utilizan para determinar el grado de adecuación o categoría de cada aspecto encuestado según los expertos. El grado de adecuación se muestra en la tabla siguiente:

Bastante Adecuado	Adecuado	No Adecuado
-------------------	----------	-------------

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

0.55	0.95	
------	------	--

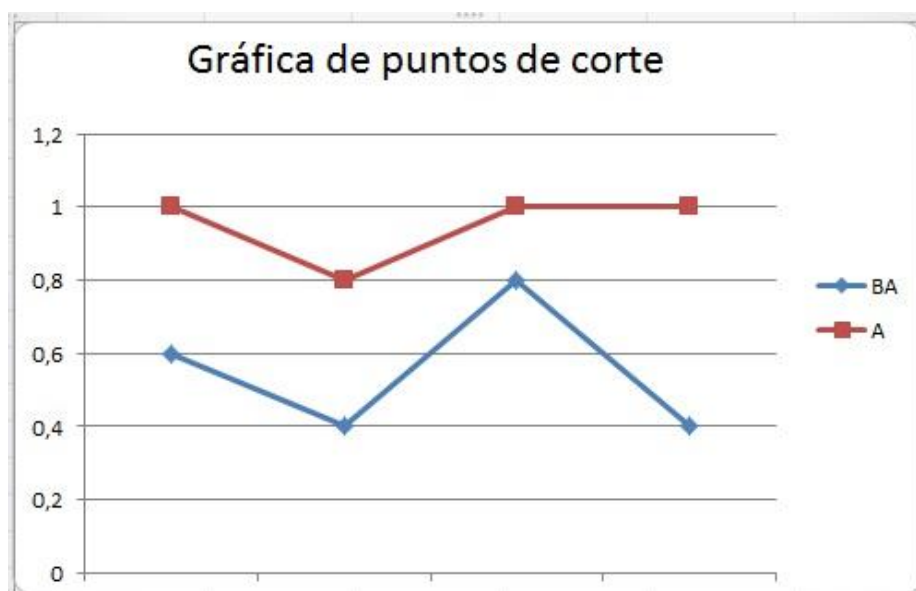


Ilustración 7. Puntos de cortes

Si el valor promedio de adecuación del elemento a evaluar es:

- Menor o igual que 0,55 el nivel de adecuación es Muy adecuado.
- Mayor que 0,55 menor o igual que 0,95 el nivel de adecuación es Adecuado.
- Mayor que 0,95 el nivel de adecuación es No adecuado

Una vez aplicado el método Delphi, las encuestas realizadas a los 5 expertos arrojaron resultados satisfactorios:

- Todas las preguntas fueron valoradas por los expertos de Bastante adecuada.

El 100% de los expertos coinciden en la utilidad que tiene la puesta en práctica de la propuesta en los subsistemas de CedruX.

3.7. Pruebas de Software

Para determinar la calidad de un producto de software se deben efectuar medidas y desarrollar actividades que permitan comprobar el grado de cumplimiento de las especificaciones iniciales del sistema, lo cual es aquí donde las pruebas de software desempeñan un papel fundamental. Estas se definen como una actividad en la cual un sistema o uno de sus componentes se ejecutan en

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

circunstancias previamente especificadas, los resultados se observan y registran con el fin de realizar una evaluación de algún aspecto.

Objetivos de las pruebas:

La prueba de software es una etapa imprescindible durante todo el proceso de desarrollo, pues una vez que se genera código fuente, el software debe ser probado para descubrir y corregir el máximo de errores posibles antes de su entrega al cliente. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces, de ahí que se hace necesario el cumplimiento de sus objetivos, expuestos a continuación, ya que este proceso posibilita no solo la detección de errores en el código sino la documentación necesaria que facilita que este código pueda ser reutilizado:

- Planificar las pruebas necesarias en cada iteración.
- Diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar, creando los procedimientos de prueba que especifican cómo realizar las pruebas y creando, si es posible, componentes de prueba ejecutables para automatizar las pruebas.
- Realizar las diferentes pruebas y manejar los resultados de cada prueba sistemáticamente.

XP divide las pruebas del sistema en dos grupos: pruebas unitarias, diseñada por los programadores con el objetivo de verificar el código y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas por el cliente final.

3.7.1. Pruebas de Caja Blanca

Con la realización de **pruebas de caja blanca** se comprueban los caminos lógicos del software y se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado, este método de pruebas se realiza sobre el código, por lo que requiere del conocimiento de la estructura interna del programa y debe garantizar como mínimo que:

- Se ejerciten por lo menos una vez todos los caminos independientes para cada módulo.
- Se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsa.
- Ejecuten todos los bucles en sus límites y con sus límites operacionales.
- Se ejerciten las estructuras internas de datos para asegurar su validez.

A continuación se presenta la realización de esta prueba utilizando la técnica del camino básico, la cual consta de varios pasos:

- A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

- Se calcula la complejidad ciclomática del grafo.
- Se determina un conjunto básico de caminos independientes.
- Derivación de casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Para llevar a cabo la prueba de camino básico se tomó una parte del código correspondiente a uno de los métodos de la clase AcmeConverter como se evidencia en la siguiente figura.

Complejidad Ciclométrica

La complejidad ciclométrica es una complejidad lógica de una rutina, permite obtener la cantidad de caminos independientes.

Esta complejidad se puede calcular de tres maneras distintas, cualquiera de ellas es válida y todas darán el mismo resultado es recomendable que se calcule de las tres formas para asegurar que el grafo se generó correctamente, entonces si con las tres fórmulas el resultado es el mismo, quiere decir que no hubo problemas al generar el grafo del método analizado.

Las fórmulas para el cálculo de la Complejidad Ciclométrica (CC) son:

CC= número de arcos – números de nodos + 2

CC= número de nodos predicados + 1

CC= número de regiones del grafo.

```
public List<Integer> entradasPorComponentes() {  
    1 { Iterator<AcmeComponent> itr = system.getComponentes().iterator();  
      List <Integer> entradas = new ArrayList<Integer>();  
      while (itr.hasNext()) { 2  
          3 { int result = 0;  
              Set<AcmePort> puertos = itr.next().getPorts();  
              Iterator<AcmePort> itr1 = puertos.iterator();  
              while (itr1.hasNext()) { 4  
                  5 { if (itr1.next().getDeclaredTypeRef("InPut") != null)  
                      6 { result ++;  
                  }  
              }  
          7 { entradas.add(result);  
          }  
      }  
      8 { return entradas;  
      }  
}
```

Ilustración 8. Código fuente del método entradasPorComponentes de la clase AcmeConverter

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

Al término de la identificación de cada línea de código, es necesario representar el grafo de flujo asociado, en el cual se representan los distintos puntos que lo componen.

En el mismo se pueden identificar:

Nodo: Cada círculo representado se denomina nodo del Grafo de Flujo. Un solo nodo puede corresponder a una secuencia de procesos o a una sentencia de decisión. Puede ser también que hallan nodos que no se asocian, se utilizan principalmente al inicio y final del grafo.

Aristas: Las flechas del grafo se denominan aristas y representan el flujo de control. Una arista debe terminar en un nodo, incluso aunque el nodo no represente ninguna sentencia procedimental.

Regiones: Las regiones son las áreas delimitadas por las aristas y nodos. También se incluye el área exterior del grafo, contando como una región más. Las regiones se enumeran. La cantidad de regiones es equivalente a la cantidad de caminos independientes del conjunto básico de un programa.

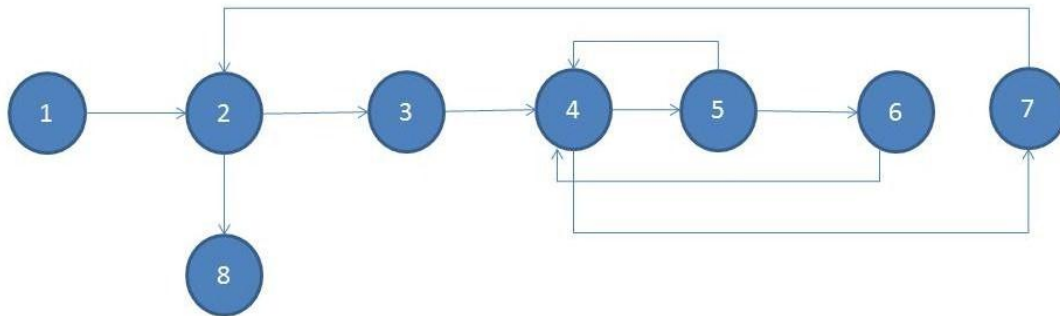


Ilustración 9. Grafo de flujo asociado al método entradasPorComponentes

Nodo de predicado: Son los nodos que contienen una condición y se caracterizan porque de ellos salen dos o más aristas.

$$CC = \text{Aristas} - \text{Nodos} + 2 = 10 - 8 + 2 = 4$$

$$CC = \text{Número de regiones} = 4$$

$$CC = \text{Nodos de predicado} + 1 = 3 + 1 = 4$$

Caminos: Camino 1 (1-2-8); Camino 2 (1-2-3-4-5-6-7-4-7-2-8); Camino 3 (1-2-3-4-7-2-8); Camino 4 (1-2-3-4-5-4-7-2-8).

Acto seguido se derivan algunos casos de prueba que obligan a la ejecución de cada camino, siendo:

Tabla 13. Casos de pruebas de Caja Blanca por caminos

No. de	Caso de prueba	Objetivo	Resultado esperado

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

caminos			
1	While (! itr.hasNext)	Verifica que el iterador no contenga ningún componente que conforma el system.	Retorna el valor 0.
2	While (itr.hasNext) While (itr1.hasNext) If(itr1.next().getDeclaredTupeRef("Input")!= null)	Verifica que el iterador contenga al menos un componente. Si el componente existe. Verifica que contenga algún puerto. Se comprueba que el tipo de puerto sea Input, se aumenta el contador.	Se agrega el valor obtenido por el contador a la lista de entradas. Devuelve la lista de entradas por componentes con todos los puestos que contiene.
3	While (itr.hasNext) While (itr1.hasNext) If(itr1.next().getDeclaredTupeRef("Input")!= null)	Verifica que el iterador contenga al menos un componente. Si el componente existe. Verifica que contenga algún puerto. Si no contiene puertos no se aumenta el contador.	No se agrega nada a la lista de entradas. Devuelve la lista de entradas por componentes vacía.
4	While (itr.hasNext) While (itr1.hasNext) If(itr1.next().getDeclaredTupeRef("Input")!= null)	Verifica que el iterador contenga al menos un componente. Si el componente existe. Verifica que contenga algún puerto. Si contiene puertos, verifica que sea un Input, si no es Input no aumenta el contador.	No se agrega nada a la lista de entradas. Devuelve la lista de entradas por componentes vacía.

3.7.2. Pruebas de Aceptación

Las pruebas de aceptación son las realizadas por el cliente y usuarios finales de la aplicación. En estas serán probadas las funcionalidades exigidas por el cliente y descritas en las historias de usuario, además

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

de los aspectos de seguridad requeridos. Luego de haber superado las pruebas de aceptación podrá considerarse que la aplicación es apta para el uso y despliegue dentro del proyecto. Estas pruebas son más importantes que las pruebas unitarias ya que significan la satisfacción del cliente con el producto desarrollado al final de una iteración. Por esta razón se decidió utilizar este tipo de prueba para garantizar el buen funcionamiento de la aplicación y así la satisfacción del cliente con el desarrollo del mismo. Se escogió un modelo a probar el cual fue la arquitectura de software del subsistema Inventario, está compuesta por 21 componentes y 341 conectores. La siguiente tabla muestra todos los elementos necesarios para realizar la evaluación y que son visualizados en el reporte detallado que se exporta.

Tabla 14. Evaluación de los componentes

Componentes	Servicios que Consume	Servicios que Provee	Tamaño $TC = \frac{Kr * 2 + Ks}{100}$	Complejidad $CoC = \frac{Cr * 2 + TC}{100}$	Criticidad $CrC = \frac{Kcd * 2 + \sum KcdD}{100}$
Ajuste	44	0	0,88	0,5383	0,34
Apertura	52	0	1,04	0,5399	0,37
Documento	19	163	2,01	0,5496	0,33
Baja	47	0	0,94	0,5389	0,49
Combustible	44	1	0,89	0,5384	0,23
Conciliación	23	0	0,46	0,5341	0,12
Configuración	6	5	0,17	0,5312	0,21
Despacho	76	5	1,57	0,5452	0,42
Generar_Comprobante	2	5	0,04	0,5299	0,08
Generar_Doc	53	5	1,06	0,5401	0,4

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

Inventario	36	1	0,73	0,5368	0,42
Nomencladores	21	33	0,75	0,5370	0,36
Nro_Serie	6	5	0,17	0,5312	0,12
Productos	14	22	0,5	0,5345	0,21
Recepción	74	0	1,48	0,5443	0,37
Recuperaciones	10	0	0,2	0,5315	0,3
Submayor	13	0	0,26	0,5321	0,29
Lote	2	11	0,15	0,5310	0,04
Utiles_Municiones	10	5	0,25	0,5320	0,22
Cierre	1	1	0,03	0,5298	0,03
Movimiento	3	89	0,95	0,5390	0,09
Total	556	349	14,53	11,2652	5,4399

Complejidad del Sistema =0,5364

Facilidad de Prueba =0,0051

La Arquitectura del subsistema es difícil de probar.

3.8. Casos de pruebas

Un caso de prueba se diseña según las funcionalidades descritas en los requisitos funcionales. El propósito que se persigue con este artefacto es lograr una comprensión común de las condiciones específicas que la solución debe cumplir. Se parte de la descripción de las Historias de Usuarios del sistema, como apoyo para las revisiones. Cada planilla de caso de prueba recoge la especificación de un requisito funcional, dividido en secciones y escenarios, detallando las funcionalidades descritas en él y describiendo cada variable que recoge el caso de uso en cuestión. A continuación se muestran las pruebas de aceptación para cada una de las iteraciones definidas.

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

Para la realización de las pruebas debe existir el modelo al cual se desee evaluar, al seleccionar el nombre del modelo con la extensión .acme se activará una pestaña con la funcionalidad que lleva el nombre Facilidad de prueba. Seguidamente dar clic en la opción Facilidad de prueba.

Tabla 15. Caso de prueba de aceptación: HU1

Casos de Prueba	
Número de caso de prueba :CP_1	Número de Historia de Usuario:HU1
Nombre del Caso de prueba: Calcular la Diagnosticabilidad.	
Condiciones de ejecución: Debe existir el modelo al cual se desee evaluar.	
Escenario de prueba 1	
Entradas: El modelo de la arquitectura del subsistema Inventario, modelado en la herramienta AcmeStudio, que está compuesto por 21 componentes y 341 conectores. (Ilustración 10)	
Resultado esperado: Se muestra una ventana con el siguiente mensaje: La facilidad de prueba de la Arquitectura es 0.0051, la Arquitectura del subsistema es difícil de probar.	
Evaluación: Una vez ejecutado el caso de prueba el sistema mostró los resultados esperados.	

Tabla 16. Caso de prueba de aceptación HU2

Casos de Prueba	
Número de caso de prueba :CP_2	Número de Historia de Usuario:HU3
Nombre del Caso de prueba: Exportar reporte detallado	
Condiciones de ejecución: Debe salir una ventana donde expone el resultado de la evaluación.	
Escenario de prueba1	
Entradas: Presionar el botón Exportar reporte detallado.	
Resultado esperado: Se exporta un reporte detallado correspondiente al cálculo de la métrica. Especificando para cada componente el tamaño, la criticidad y la complejidad con los valores que se muestran en la Tabla 8.	

CAPÍTULO 3. VALIDACIÓN Y PRUEBAS

Evaluación: El resultado luego de ejecutada la prueba.
Escenario de prueba2
Entradas: Presionar el botón que dice Cancelar .
Resultado esperado: Se cierra la ventana y no se guarda el reporte detallado de la evaluación realizada.

Resultado obtenidos en las pruebas

Una vez que se aplican las pruebas funcionales se arrojan diferentes resultados. La calidad del sistema está dada en los errores que a partir de dichas pruebas se pueden encontrar y a la vez corregir, minimizándole tiempo y costo al producto y obteniendo un software satisfactorio. El plugin desarrollado, después de aplicarle las pruebas no detectó ninguna no conformidad significativa, por lo cual se culminó el presente trabajo con resultados satisfactorios.

3.9. Conclusiones del capítulo

Con el desarrollo de este capítulo se llegó a la conclusión siguiente.

- Se describieron las fases de validación y prueba.
- Las características de los expertos seleccionados para la validación del procedimiento, garantiza la calidad de las opiniones emitidas por los mismos, demostrado que los objetivos propuestos se cumplieron, como quedo demostrado en el análisis estadístico de los resultados obtenidos.
- Fueron detalladas además cada una de las tareas que se realizaron en las dos iteraciones del sistema.
- Se abordó acerca de la importancia de las pruebas en la metodología XP, en especial las pruebas de aceptación, ya que éstas miden la satisfacción del cliente con el producto desarrollado.
- Se realizó el plan de duración de las iteraciones para darle prioridad a las de mayor necesidad.
- Una vez concebida la estructura y la implementación del sistema, se realizaron los casos de pruebas para validar la completitud de los requerimientos, obteniéndose resultados satisfactorios

CONCLUSIONES

CONCLUSIONES

1. Luego de un estudio profundo a cerca de las métricas de calidad de software para evaluar una arquitectura de software, se expusieron las características de las métricas de calidad, su importancia y se realizaron transformaciones para el desarrollo de una métrica capaz de obtener los valores para la evaluación de una arquitectura de software realizada en la herramienta AcmeStudio, para el desarrollo del plugin.
2. Se estudiaron 3 métricas las cuales no cumplían con los datos necesarios para evaluar el modelado de la arquitectura y se decidió la realización de una nueva métrica.
3. A partir del análisis y diseño realizado en el plugin, se expusieron los conceptos esenciales del sistema los cuales ayudaron a la comprensión del mismo, así como la definición de la estructura principal del plugin, proporcionando la entrada apropiada y el punto de partida para las actividades de implementación.
4. Se realizó la implementación correspondiente obteniendo un plugin con las funcionalidades necesarias para garantizar que se realicen correctamente las evaluaciones de los diseños realizados por los arquitectos de un sistema y ayudar en la toma de decisiones.
5. Se realizaron las pruebas funcionales al plugin para validar las funcionalidades que fueron implementadas, las cuales demostraron que los indicadores de calidad cumplieron satisfactoriamente con los requisitos propuestos, garantizando su correcto funcionamiento.

RECOMENDACIONES

RECOMENDACIONES

Luego de haber analizado los resultados del presente trabajo de diploma, surgen algunas ideas que podrían ser incorporadas en un futuro con el objetivo de fortalecer el sistema desarrollado, por lo que se recomienda:

1. Realizar la implementación de nuevas métricas para evaluar el diseño de una Arquitectura de Software.
2. Realizar un estudio de métricas de calidad para el diseño convencional e identificar métricas que puedan ser incorporadas en futuras versiones del plugin.
3. Investigar otras técnicas cuantitativas para la evaluación de la diagnosticabilidad de la arquitectura en subsistemas de CedruX.

BIBLIOGRAFÍA

BIBLIOGRAFÍA

1. Gastón Escobar y Passerini , A.y.N., Arquitectura de Proyectos de IT. Evaluación de Arquitecturas. 2005, Universidad Tecnológica Nacional. Facultad Regional de Buenos Aires - Departamento de Sistemas: Buenos Aires.
2. Martínez Ortega, C., Procedimiento para evaluar la Mantenibilidad de componentes de software. mayo del 2011, Universidad de las Ciencias Informáticas: La Habana
3. González y Grimán , A.y.A., Método de Evaluación de Arquitecturas de Software Basadas en Componentes (MECABIC). Vol. 1. 2005, Colimia, Mexico.
4. Pérez , K., Modelo matemático para la evaluación cuantitativa de la confiabilidad en la arquitectura de CedruX. Junio 2012, Universidad de las Ciencias Informáticas: La Habana, Cuba.
5. Scalone y García Martínez , F.y.R., ESTUDIO COMPARATIVO DE LOS MODELOS Y ESTANDARES DE CALIDAD DEL SOFTWARE. Junio de 2006.
6. Barbacci, M., Steps in an Architecture Tradeoff Analysis Method: Quality Attribute Models and Analysis. TECHNICAL REPORT CMU/SEI-97-TR-029 ESC-TR-97-029. May 1998.
7. Camacho y Cardeso , E.y.F., ARQUITECTURAS DE SOFTWARE. Abril 2004.
8. Palafox Antonio Marcial , P., Modelado de tácticas de atributos de calidad para la generación de arquitecturas ejecutables. 30 de enero de 2009.
9. Fábrega Souchay, D., Propuesta de procedimiento para el Mantenimiento de Software. 2009, Universidad de las Ciencias Informáticas: Ciudad de La Habana.
10. Tecnología de la Información. Características de Calidad y Métricas del Software, NC ISO/IEC 9126-1. Edición 2005.
11. Chaviano Gómez, E., Evaluación de arquitecturas de software., La Habana Universidad de las Ciencias Informáticas.
12. Regalado Vigil, Y., Metodología de evaluación de arquitectura de software. 2009, Universidad de las Ciencias Informáticas: La Habana.
13. NC-ISO-IEC 9126-4 Las tablas de métricas de calidad. .
14. Peláez Rodríguez y Alvarez Hernández, A.y.H., Plugin para la gestión de pruebas en la herramienta REDMINE. Ciudad de la Habana : s.n., Junio de 2011.
15. Posso Neptalí Rivera, A., ESTUDIO DE LA ARQUITECTURA DE SOFTWARE. DICIEMBRE DE 2010, UNIVERSIDAD TÉCNICA DEL NORTE.
16. Jacobson y Rumbaugh, J.y.I., El Proceso Unificado de Desarrollo. 2000, Addison Wesley.

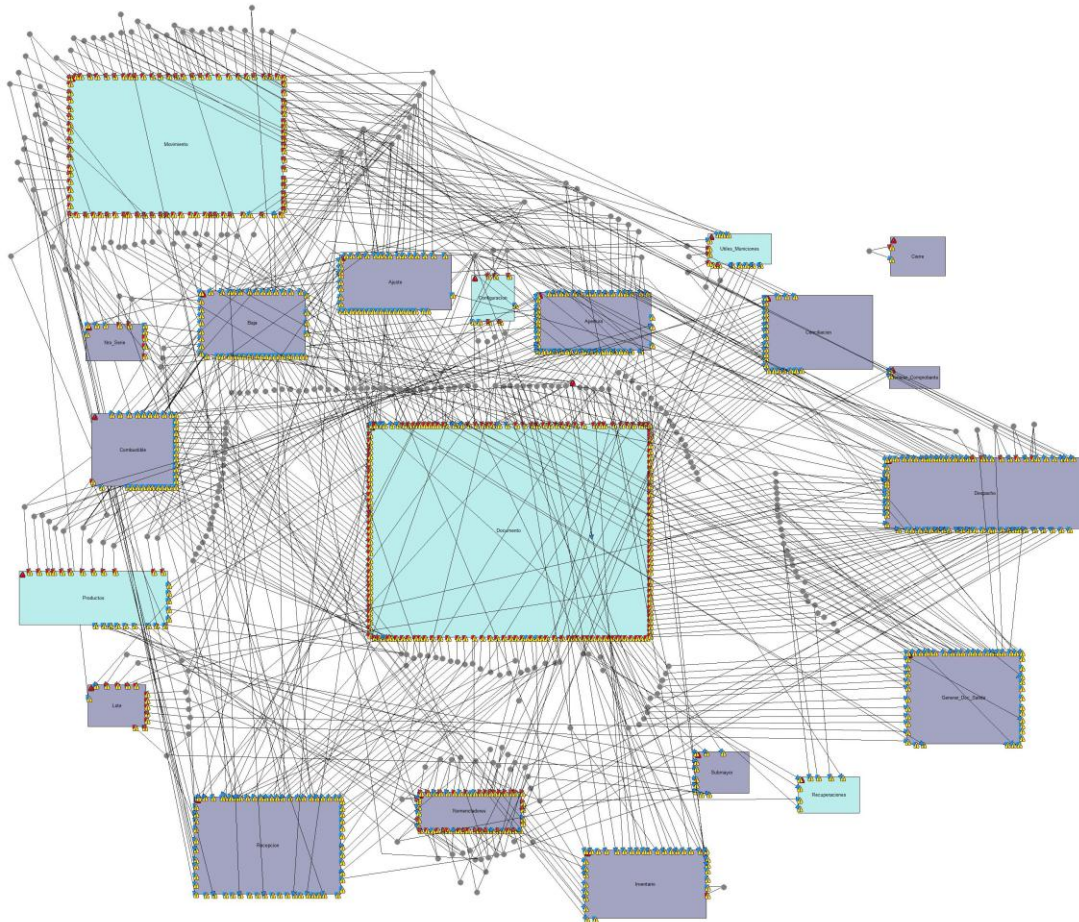
BIBLIOGRAFÍA

17. Sitio oficial Java-Lenguaje de programación.
[http://www.java.com/es/download/faq/whatis_java.xml\(Oracle\)](http://www.java.com/es/download/faq/whatis_java.xml(Oracle)).
18. Hernández Rabelo, J.C.y.M.D., Análisis y Diseño del Modelo de Factibilidad para la Evaluación de Proyectos de Software en la Universidad de las Ciencias Informáticas. 2010, Universidad de las Ciencias Informáticas.
19. Sierra, D.S., Visual Paradigm For Uml. 15 de Noviembre de 2007.
20. Batista González y Bozán, D.e.I., Desarrollo de una aplicación para la automatización de la evaluación de la Arquitectura de Software en los Proyectos Productivos de la UCI. Junio 2010, Universidad de las Ciencias Informáticas, : Ciudad de La Habana.
21. Matos Arias y Silega Martínez, Y.y.N., ESTILO ARQUITECTÓNICO PARA EL SISTEMA INTEGRADO DE GESTIÓN CEDRUX, Universidad de las Ciencias Informáticas (UCI),.
22. Ramírez, J., Arquitectura de software,CIG-CDX-N-INV-i3406 Matriz_Integración_Inventario Versión: 1.0. 31 de enero del 2011., Universidad de las Ciencias Informáticas La Habana.
23. Simón Caridad, S., METRICAS Y MODELOS DE ESTIMACION DEL SOFTWARE. 2010.
24. Fernández Leyet, O., Propuesta metodológica para la obtención de los componentes de software en los proyectos del sistema Cedrux. 2011, Universidad de las Ciencias Informáticas: La Habana.
25. Libro de UML y Patrones de Craig Larman.
26. Novella, A.I., Procedimiento para la validación y verificación de los Requerimientos No Funcionales en aplicaciones web de gestión. 2011, Universidad de las Ciencias Informáticas.

ANEXOS

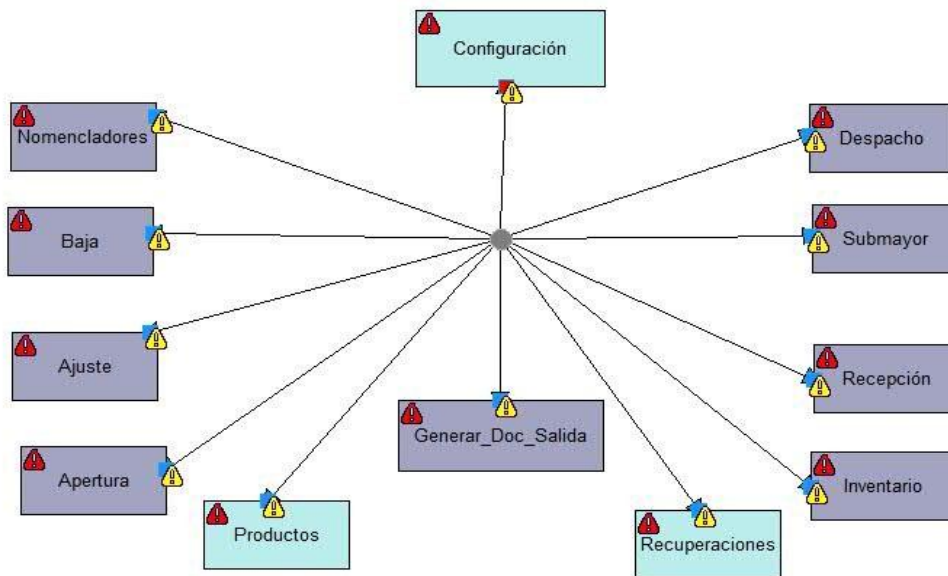
ANEXOS

Anexo 1: Arquitectura del Subsistema Inventario



ANEXOS

Anexo 2: Ejemplo de un servicio que brinda el componente Configuración a los demás componentes



Anexo 3: Cuestionario para la selección de expertos

Compañero (a):

En la presente tesis, se desea someter a la valoración de un grupo de expertos una propuesta de evaluación de la diagnosticabilidad en los subsistemas de Cedrux, a través del modelado de su arquitectura, para garantizar una mejor calidad del software que se produce en el CEIGE. La evaluación de la diagnosticabilidad se realiza para obtener la capacidad que tiene el producto de ser objeto de un diagnóstico para detectar deficiencias o causas de los fallos totales en el software, o para identificar las partes que van a ser modificadas. Para ello necesitamos conocer el grado de dominio que usted posee sobre el tema de arquitectura y evaluación de la misma y con ese fin se desea que responda lo que se le pide a continuación.

Nombre y Apellidos: _____

Rol que desempeña: _____

Años de experiencia: _____

Especialidad: _____

ANEXOS

Categoría docente: _____

1.- Marque con una cruz (x) el grado de conocimiento que usted tiene sobre el tema que se lleva a cabo en esta investigación.

1	2	3	4	5	6	7	8	9	10

Tabla 1: Grado de conocimiento del experto

2.- Marque con una cruz (x) las fuentes que le han servido para argumentar el conocimiento que tiene usted del tema que se lleva a cabo en esta investigación. El 1 indica que no posee absolutamente ningún conocimiento de la problemática en estudio, mientras que el 10 expresa pleno conocimiento.

Grado de influencia			Fuentes de argumentación
Alto	Medio	Bajo	
			Análisis teóricos realizados por usted.
			Experiencia.
			Trabajos de autores nacionales.
			Trabajos de autores extranjeros.
			Su propio conocimiento del tema.
			Su intuición.

Tabla 2: Fuentes para argumentar el conocimiento de los expertos

Anexo 4: Tabla con los valores asignados por el método Delphi

Grado de influencia			Fuentes de argumentación
Alto	Medio	Bajo	
0,3	0,2	0,1	Análisis teóricos realizados por usted.
0,5	0,4	0,2	Experiencia.
0,05	0,05	0,05	Trabajos de autores nacionales.
0,05	0,05	0,05	Trabajos de autores extranjeros.
0,05	0,05	0,05	Su propio conocimiento del tema.
0,05	0,05	0,05	Su intuición.

Tabla: Patrón para el cálculo del coeficiente de argumentación (Ka) del conocimiento de los expertos (25).

ANEXOS

Anexo 5: Coeficiente de competencia de los expertos

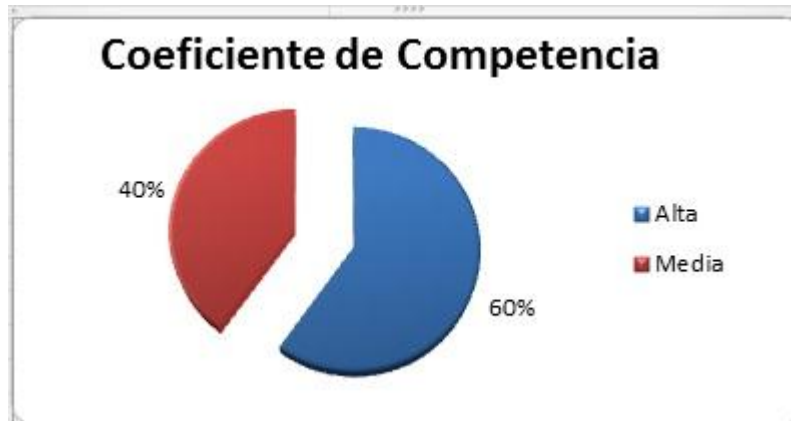


Ilustración: Coeficiente de competencia de los expertos.

Anexo 6: Encuesta realizada a los expertos

Dentro de la Calidad de Software como característica fundamental a tener en cuenta se encuentra la **mantenibilidad** la cual se define 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.

Para que un sistema tenga la calidad requerida, confiabilidad, seguridad de la información y una buena repercusión en el mercado es de suma importancia aplicar las propiedades de mantenibilidad entre ellas la **Facilidad de prueba**, para realizar evaluaciones a la arquitectura del mismo, ya que con ella se comprueba la capacidad del producto de ser objeto de un diagnóstico para detectar deficiencias o causas de los fallos totales en el software, o para identificar las partes que van a ser modificadas.

Para evaluar la arquitectura se realizará a través de las técnicas de evaluación cuantitativa y dentro de esta las métricas.

Las métricas para software, como otras métricas, no son perfectas, sin embargo el diseño sin medición es una alternativa inaceptable. Las métricas examinan el modelo que realizado en la herramienta AcmeStudio se analiza con el fin de predecir lo fácil que se puede probar una arquitectura y así ayudar en la toma de decisiones para aplicar reparaciones en el caso que sea necesario. La reparabilidad se ve afectada por la cantidad y tamaño de los componentes o piezas:

- Un producto software que consiste en componentes bien diseñados es más fácil de analizar y reparar que uno monolítico.

ANEXOS

- Pero el incremento del número de componentes no implica un producto más reparable, ya que también aumenta la complejidad de las interconexiones entre los componentes.

A la hora de evaluar una arquitectura de software es importante conocer el tamaño del sistema resultante, donde resulta probable que el tamaño y la complejidad del diseño estén directamente relacionados. A continuación se muestra y se explica una de las métricas de diagnosticabilidad que proporciona al diseñador una mejor visión interna y así el diseño evolucionará a un mejor nivel de calidad.

McCabe [Pressman '93] identifica un número importante de usos para las métricas de complejidad, donde pueden emplearse para predecir información sobre la fiabilidad y mantenimiento de sistemas software, también realimentan la información durante el proyecto de software para ayudar a controlar la actividad de diseño, en las pruebas y mantenimiento, proporcionan información sobre los módulos software para ayudar a resaltar las áreas de inestabilidad.

Métrica: Capacidad de análisis de fallos.

Esta se concentra en las características de la estructura del programa dándole énfasis a la estructura arquitectónica y en la eficiencia de los módulos. Card y Glass manifiestan que a medida que aumenten la cantidad de componentes y conectores, la complejidad arquitectónica o global del sistema también aumenta. Esto lleva a una mayor probabilidad de que aumente el esfuerzo necesario para la integración y las pruebas. Por lo que se define una métrica para evaluar la arquitectura de los subsistemas de CedruX a través de la característica de diagnosticabilidad de la forma:

$$X = \frac{1}{(CantComp + CantConec) * ComplSist}$$

Donde X es la facilidad de prueba de una arquitectura, CantComp es la cantidad de componentes por la que está compuesta la arquitectura, CantConec es la cantidad de conectores y ComplSist es la complejidad del sistema. Para el cálculo de la complejidad del sistema se necesita otras ecuaciones como son:

$TC = \frac{Kr*2+Ks}{100}$. Donde TC es el tamaño de un componente, Kr cantidad de servicios que consume un componente y Ks cantidad de servicios que brinda un componente.

ANEXOS

$Coc = \frac{Cr*2+TC}{100}$. Donde Coc es la complejidad de un componente, Cr promedio de servicios que se consume y TC tamaño del componente.

Encuesta

Las preguntas se clasifican en Bastante Adecuado (BA), Adecuado (A), No Adecuado (NA).

No.	Preguntas	BA	A	NA
1	¿Cómo considera usted la ecuación $CS = \frac{\sum Coc}{CantComp}$ para el cálculo de la complejidad total de un sistema?			
2	Partiendo que la propuesta del presente trabajo es: $X = \frac{1}{(CantComp + CantConsc)*ComplSist}$. ¿Qué nivel de correspondencia le brinda usted a la fórmula propuesta con relación al cálculo de la facilidad de pruebas de un subsistema de CedruX?			
3	¿Qué grado de utilidad usted le concede al cálculo de la facilidad de prueba para la toma de decisiones durante el desarrollo de los subsistemas que se desarrollan de CedruX?			
4	¿Cómo valora usted el valor científico de la propuesta?			

ANEXOS

Anexo 7: Tabla Distribución Chi Cuadrado

TABLA 3-Distribución Chi Cuadrado χ^2

P = Probabilidad de encontrar un valor mayor o igual que el chi cuadrado tabulado, v = Grados de Libertad

v/p	0,001	0,0025	0,005	0,01	0,025	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5
1	10,8274	9,1404	7,8794	6,6349	5,0239	3,8415	2,7055	2,0722	1,6424	1,3233	1,0742	0,8735	0,7083	0,5707	0,4549
2	13,8150	11,9827	10,5965	9,2104	7,3778	5,9915	4,6052	3,7942	3,2189	2,7726	2,4079	2,0996	1,8326	1,5970	1,3863
3	16,2660	14,3202	12,8381	11,3449	9,3484	7,8147	6,2514	5,3170	4,6416	4,1083	3,6649	3,2831	2,9462	2,6430	2,3660
4	18,4662	16,4238	14,8602	13,2767	11,1433	9,4877	7,7794	6,7449	5,9886	5,3853	4,8784	4,4377	4,0446	3,6871	3,3567
5	20,5147	18,3854	16,7496	15,0863	12,8325	11,0705	9,2363	8,1152	7,2893	6,6257	6,0644	5,5731	5,1319	4,7278	4,3515
6	22,4575	20,2491	18,5475	16,8119	14,4494	12,5916	10,6446	9,4461	8,5581	7,8408	7,2311	6,6948	6,2108	5,7652	5,3481
7	24,3213	22,0402	20,2777	18,4753	16,0128	14,0671	12,0170	10,7479	9,8032	9,0371	8,3834	7,8061	7,2832	6,8000	6,3458
8	26,1239	23,7742	21,9549	20,0902	17,5345	15,5073	13,3616	12,0271	11,0301	10,2189	9,5245	8,9094	8,3505	7,8325	7,3441
9	27,8767	25,4625	23,5893	21,6660	19,0228	16,9190	14,6837	13,2880	12,2421	11,3887	10,6564	10,0060	9,4136	8,8632	8,3428
10	29,5879	27,1119	25,1881	23,2093	20,4832	18,3070	15,9872	14,5339	13,4420	12,5489	11,7807	11,0971	10,4732	9,8922	9,3418
11	31,2635	28,7291	26,7569	24,7250	21,9200	19,6752	17,2750	15,7671	14,6314	13,7007	12,8987	12,1836	11,5298	10,9199	10,3410
12	32,9092	30,3182	28,2997	26,2170	23,3367	21,0261	18,5493	16,9893	15,8120	14,8454	14,0111	13,2661	12,5838	11,9463	11,3403
13	34,5274	31,8830	29,8193	27,6882	24,7356	22,3620	19,8119	18,2020	16,9848	15,9839	15,1187	14,3451	13,6356	12,9717	12,3398
14	36,1239	33,4262	31,3194	29,1412	26,1189	23,6848	21,0641	19,4062	18,1508	17,1169	16,2221	15,4209	14,6853	13,9961	13,3393
15	37,6978	34,9494	32,8015	30,5780	27,4884	24,9958	22,3071	20,6030	19,3107	18,2451	17,3217	16,4940	15,7332	15,0197	14,3389
16	39,2518	36,4555	34,2671	31,9999	28,8453	26,2962	23,5418	21,7931	20,4651	19,3689	18,4179	17,5646	16,7795	16,0425	15,3385
17	40,7911	37,9462	35,7184	33,4087	30,1910	27,5871	24,7690	22,9770	21,6146	20,4887	19,5110	18,6330	17,8244	17,0646	16,3382
18	42,3119	39,4220	37,1564	34,8052	31,5264	28,8693	25,9894	24,1555	22,7595	21,6049	20,6014	19,6993	18,9679	18,0860	17,3379
19	43,8194	40,8847	38,5821	36,1908	32,8523	30,1435	27,2036	25,3289	23,9004	22,7178	21,6891	20,7638	19,9102	19,1069	18,3376
20	45,3142	42,3358	39,9969	37,5663	34,1696	31,4104	28,4120	26,4976	25,0375	23,8277	22,7745	21,8265	20,9514	20,1272	19,3374
21	46,7963	43,7749	41,4009	38,9322	35,4789	32,6706	29,6151	27,6620	26,1711	24,9348	23,8578	22,8876	21,9915	21,1470	20,3372
22	48,2676	45,2041	42,7957	40,2894	36,7807	33,9245	30,8133	28,8224	27,3015	26,0393	24,9390	23,9473	23,0307	22,1663	21,3370
23	49,7276	46,6231	44,1814	41,6383	38,0756	35,1725	32,0069	29,9792	28,4288	27,1413	26,0184	25,0055	24,0689	23,1852	22,3369
24	51,1790	48,0336	45,5584	42,9798	39,3641	36,4150	33,1962	31,1325	29,5533	28,2412	27,0960	26,0625	25,1064	24,2037	23,3367
25	52,6187	49,4351	46,9280	44,3140	40,6465	37,6525	34,3816	32,2825	30,6752	29,3388	28,1719	27,1183	26,1430	25,2218	24,3366
26	54,0511	50,8291	48,2898	45,6416	41,9231	38,8851	35,5632	33,4295	31,7946	30,4346	29,2463	28,1730	27,1789	26,2395	25,3365
27	55,4751	52,2152	49,6450	46,9628	43,1945	40,1133	36,7412	34,5736	32,9117	31,5284	30,3193	29,2266	28,2141	27,2569	26,3363
28	56,8918	53,5939	50,9936	48,2782	44,4608	41,3372	37,9159	35,7150	34,0266	32,6205	31,3909	30,2791	29,2486	28,2740	27,3362
29	58,3006	54,9662	52,3355	49,5878	45,7223	42,5569	39,0875	36,8538	35,1394	33,7109	32,4612	31,3308	30,2825	29,2908	28,3361

Anexo 8: Listado de Experto

Nombre de los expertos

Msc. Larisa González Álvarez

Ing. Willian González

Ing. Javier Ramírez Hernández

Msc. Osmar Leyet Fernández

Ing. Omar A Díaz

GLOSARIO

GLOSARIO DE TERMINOS

Arquitectura de software: No es más que tener una vista global y abstracta del sistema que se desea desarrollar, que nos permita tomar las decisiones adecuadas, definir los parámetros y delimitación de dicho sistema tomando en cuenta los requerimientos funcionales y no funcionales.

Calidad del software: Grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario.

Mantenibilidad: La facilidad con la que un sistema o componente software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno.

Diagnosticabilidad: Capacidad del producto de ser objeto de un diagnóstico para detectar deficiencias o causas de los fallos totales en el software, o para identificar las partes que van a ser modificadas.

Herramientas: Es un objeto elaborado a fin de facilitar la realización de una tarea mecánica que requiere de una aplicación correcta de energía.

Stakeholder: Cualquier persona o entidad que es afectada por las actividades de una organización.

Metodología: Se refiere a los métodos de investigación que se siguen para alcanzar una gama de objetivos.

UCI: Universidad de las Ciencias Informáticas creada en la ciudad de la Habana, Cuba, año 2002.

UML: Lenguaje Unificado de Modelado (Unified Modeling Language). Notación gráfica utilizada para describir sistemas de software.

Requisitos: Una condición o capacidad para un usuario para resolver un problema o alcanzar un objetivo.

RF: Requerimientos Funcionales.

RNF: Requerimientos No Funcionales.

Artefactos: productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.

GLOSARIO

ADL: Lenguaje de Descripción Arquitectónica.

AcmeStudio: Herramienta de diseño arquitectónico.

TC: Tamaño del componente.

Kr: La cantidad de servicios que consume el componente.

Ks: Cantidad de servicios que brinda el componente.

Coc: Es la complejidad del componente.

Cr: Complejidad promedio de servicios que se consume.

Crc: Criticidad del componente.

Kcd: Cantidad de componentes dependientes directos.

KcdD: Cantidad de componentes dependientes de los dependientes directos del componente.

Proyecto: esfuerzo temporal que se lleva a cabo para crear un producto o un servicio con un resultado único.

