Universidad de las Ciencias Informáticas
Facultad 3

Título: *Plugin para la evaluación de la interoperabilidad del subsistema Contabilidad a partir de la modelación de su arquitectura.*

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: *Kirenia Garcia Marisí.*

Tutor: *Ing. Alain Fernández Deronceré.*

La Habana, Cuba
Junio de 2013

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

Lord Kelvin

Declaración de autoría

Declaro ser autora de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Kirenia Garcia Marisí

_____

(Autor)

Ing. Alain Fernández Deronceré

_____

(Tutor)

Datos de contacto

Síntesis del Tutor:

Ing. Alain Fernández Deronceré

Ingeniero en Ciencias Informática 2006.

Profesor Asistente.

Cinco años de experiencia en el desarrollo de software de gestión.

Clasificación: Desarrollo.

A mi abuela por tener tanta paciencia conmigo y acogerme entre tus brazos desde que me dieron a luz.

A mi tía por ser junto a mi abuela otra madre en la que podía confiar.

A mi primo Marlon por ser la persona que me roba todo el amor que una persona es capaz de soportar.

A mi papá por siempre amarme aunque no estuvo cerca de mí en mis estudios sé que cuidó de mí a distancia.

A mi novio por amarme tanto y demostrarme que aún no había conocido el amor hasta que él llegó a mi vida.

A mis hermanos de sangre: Melisa, Yusnier y Lizandra porque a pesar que no nos criamos untos siempre me brindaron su amor.

A mis hermanos de amistad infinita: Milagro, Yuya, Yaylen, Yanisleydis, Eduardo, Liannis, Gladys, Mari, Meylin, Lianet , Yasmany y Rayner por siempre estar ahí cuando los necesité

A mi otra familia: Emilia ,Saralys, Tania ,Carlo y Maide por quererme y apoyarme siempre.

Dedicatoria

Dese lo más profundo de mi corazón dedico esta tesis:
A mi abuela por ser el motor impulsor para que yo llegara donde estoy y espero que esté muy orgullosa por haber hecho su sueño realidad: que me graduara en esta escuela. Mami para ti lo mejor del mundo, te amo mucho.

A mi tía por haber luchado junto a mi abuela y sembrar en mí ese voto de confianza: tía te agradezco que me hayas querido como tu otra hija a pesar de que ya tenías tu hijo cuando yo viene a estudiar a la UCI.

Resumen

La funcionalidad es un atributo de calidad relevante para los Subsistemas de Contabilidad y uno de sus subatributos es la interoperabilidad, el cual comprueba la capacidad de intercambiar y compartir datos entre dos sistemas o componentes informáticos y permitir la compartición de información y conocimientos. Por tal motivo, el presente trabajo tiene como objetivo desarrollar un plugin que, partiendo de la modelación de la arquitectura del subsistema contabilidad en una primera versión, sea capaz de evaluar cuantitativamente la interoperabilidad de la misma, de manera que permita reducir el tiempo de la toma de decisiones en cuanto a la arquitectura. Para la realización de dicho plugin fue importante la selección de una métrica para la evaluación de la arquitectura del subsistema. 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, evaluación, funcionalidad**,** interoperabilidad, métrica.

Tabla de Contenidos

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La contabilidad es una herramienta fundamental para el desarrollo económico de una empresa, su principal objetivo es clasificar, registrar y resumir la actividad financiera de una empresa o negocio. De tal manera que permita la toma de decisiones en las compañías y negocios, es decir, permite entregar una "foto" de lo que es una empresa en términos económicos, la cual dice rápidamente la situación actual del negocio, su estabilidad y asimismo su capacidad financiera. La importancia de la contabilidad va más allá del simple registro y posterior representación de datos, es una herramienta clave para la operación financiera sana de una empresa, para la gestión, la toma de decisiones y para las organizaciones actuales. Así como lo son también los sistemas de contabilidad, los cuales juegan un papel fundamental en el desarrollo económico del país [1].

Los sistemas de contabilidad representan una de las herramientas más importantes y esenciales para lograr el desarrollo de las compañías. A su vez, los sistemas de contabilidad son los encargados de analizar y valorar todos los resultados económicos que suele obtener una empresa mediante la agrupación y la comparación de resultados, permitiendo así, que todas las tareas de la compañía sean ejecutadas bajo control y con la supervisión de un contador. Se debe destacar que la creación de los sistemas contables eficientes surgió de la necesidad de información que los actores que componen la empresa necesitaban. En este sentido, un sistema debe encontrarse estructurado y especialmente diseñado para que sea capaz de clasificar todas las acciones económicas que implica la construcción y formación de una empresa, es decir, debe contar con una arquitectura capaz de representar la estructura del sistema describiendo las partes que lo integran y sobre todo debe ser robusta, pues muchas veces estos subsistemas no están diseñados para los cambios que pueden ocurrir como la misma política de la empresa, la política contable del país o la variación en el precio específico de un bien o servicio [1] [2].

Es de gran importancia desarrollar la arquitectura de cualquier sistema, pues es como el puente que une los requerimientos del sistema y la implementación, es la base para el diseño, facilita el entendimiento del sistema ya que permite ver los componentes en pequeñas piezas, especifica la estructura del sistema, organiza los componentes y sus relaciones así como los atributos de calidad, pero no solo es importante el desarrollo de una arquitectura sino también su evaluación ya que cuando se evalúa la arquitectura se analiza y se identifican riesgos en la estructura y propiedades que

pueden afectar al sistema resultante, así como verificar si los atributos de calidad se pueden desarrollar y si se satisfacen.

La Universidad de las Ciencias Informáticas (UCI) cuenta con un centro de Informatización de Gestión de Entidades (CEIGE), en el cual se desarrolla el Sistema de Planificación de Recursos Empresariales (ERP) denominado Cedrux. La forma de evaluar la arquitectura para lograr una adecuada calidad en Cedrux, es mediante la revisión de diferentes artefactos de forma manual, revisando diferentes tablas que contienen información de la misma, la documentación de toda la arquitectura está registrada en Excel, lo cual no permite un fácil análisis de la información, no existe una métrica hoy en día que se utilice para desarrollar la evaluación del atributo de calidad interoperabilidad, por todo lo antes mencionado conlleva que la toma de decisiones con respecto al rediseño de la arquitectura y los posibles cambios que puedan ocurrir en ella se demore, por lo cual puede incumplir en el tiempo pactado con el cliente, así como el rediseño de la misma.

La problemática planteada permite definir el **problema a resolver** ¿Cómo reducir el tiempo en que se logra evaluar cuantitativamente la interoperabilidad para agilizar el proceso de toma de decisiones relacionadas con la arquitectura del subsistema Contabilidad de Cedrux?

Teniéndose como **objeto de estudio**: Proceso de evaluación cuantitativa de la arquitectura.

Y su **campo de acción** está definido en: Proceso de evaluación cuantitativa de la arquitectura, desde el punto de vista de la interoperabilidad.

Para la solución del problema expuesto se define como **objetivo general:** Desarrollar un plugin para medir cuantitativamente la interoperabilidad del subsistema Contabilidad de Cedrux, a partir de la modelación de su arquitectura, de manera que se reduzca el tiempo en que se lleva a cabo esta actividad y se agilice la toma de decisiones relacionadas con la arquitectura del subsistema de contabilidad de Cedrux.

Luego, para dar cumplimiento al objetivo general planteado, se tienen los siguientes **objetivos específicos**:

1 Realizar el marco teórico de la investigación relativo a los métodos y técnicas para la evaluación cuantitativa de la arquitectura.

2 Modelar la arquitectura del subsistema Contabilidad en un lenguaje de descripción arquitectónica (ADL).

3 Desarrollar un plugin que implemente una técnica de evaluación cuantitativa de la interoperabilidad a partir de la arquitectura modelada.

4 Validar la solución propuesta mediante criterios de expertos.

Se plantea como **idea a defender**: La evaluación cuantitativa de la interoperabilidad de la arquitectura del subsistema de contabilidad permitirá reducir el tiempo en que se lleva a cabo esta actividad y agilizar la toma de decisiones con respecto a la arquitectura del subsistema de contabilidad de Cedrux.

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: se utilizó este método para analizar documentos y teorías, logrando extraer los elementos más importantes que se relacionan con los sistemas de contabilidad. Además para analizar a través de una profunda búsqueda las tecnologías, herramientas y metodologías a utilizar en el desarrollo del plugin.

- ✓ **Métodos empíricos:**

Entrevista: se realiza este método con el objetivo de obtener información sobre la gestión del subsistema contabilidad, del sistema integral de gestión de recursos empresariales (ERP), Cedrux, intercambiando directamente con las personas involucradas en el mismo. Así como entrevistar a aquellas personas que son especialistas en la investigación del atributo de interoperabilidad.

Capítulo 1: Fundamentación teórica

En este capítulo se realiza un análisis de los conceptos y elementos para la evaluación de la interoperabilidad del subsistema contabilidad, desarrollados bajo el Lenguaje de Descripción de la Arquitectura (ADL); el cual será Acme y el lenguaje de programación Java. Además se hace un estudio de las diferentes técnicas de evaluación de la arquitectura que existen actualmente para basar el desarrollo del plugin en una de ellas.

Capítulo 2: Propuesta de solución

En este capítulo se explican las métricas, las técnicas y las herramientas que se van a emplear y se ofrece la propuesta de solución para evaluar la interoperabilidad de los componentes, desarrollados bajo el lenguaje de descripción de la arquitectura y el lenguaje de programación Java, mediante la evaluación de la interoperabilidad. Se realizan los diagramas correspondientes para llevar a cabo una buena implementación del plugin solicitado,

Capítulo 3: Implementación y prueba

En este capítulo se lleva a cabo el desarrollo y validación de la propuesta del plugin, así como el desarrollo de un modelo de prueba para verificar la efectividad del mismo.

Capítulo 1. Fundamentación teórica

1.1 Introducción

En aras de realizar un plugin capaz de evaluar cuantitativamente la interoperabilidad del subsistema de contabilidad, en este capítulo se abordarán las diferentes definiciones que se hacen necesarias para desarrollar dicho plugin. Se expondrán las diferentes técnicas que existen para realizar la evaluación de la arquitectura de un software y las métricas de interoperabilidad.

1.2 Arquitectura de Software

La Arquitectura de Software (AS) es lo primordial a la hora de realizar el desarrollo de un sistema, ya que es en la arquitectura donde se definen los artefactos así como las herramientas que se utilizarán en el desarrollo del sistema. Según [3] es la arquitectura la encargada de unir todas las partes que interactúan. Si una arquitectura no está bien definida el sistema va a tender al fracaso.

1.2.1 Definiciones de Arquitectura de Software

La arquitectura del software es una primera aproximación al diseño de alto nivel donde se identifican los principales componentes, su interacción y sus dependencias, puede decirse que la arquitectura de un programa o sistema de cómputo es la estructura o estructuras del sistema, los cuales comprenden los elementos de software, las propiedades externamente visibles de estos elementos, y las relaciones entre ellos [4] [3].

Hoy en día el término de arquitectura se usa para referirse a varios aspectos relacionados con las Tecnologías de la Información. De acuerdo al Instituto de Ingenieros de Software (SEI), la Arquitectura de Software se refiere a las estructuras de un sistema, compuestos de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos [5].

Esta definición tiene múltiples implicaciones, ya que la arquitectura es una abstracción de un sistema. De hecho, todo sistema tiene una arquitectura, lo cual no significa ni que ésta sea conocida, ni buena, ni apropiada para los propósitos del proyecto.

La arquitectura es una descripción de los componentes de un sistema informático y las relaciones entre ellos. Además constituye un amplio marco que describe su estructura, componentes, su forma y la manera que estos interactúan. Es la base fundamental para lograr el éxito de un sistema [6].

La Arquitectura de Software se ocupa de la organización general de un sistema en términos de sus componentes, sus propiedades visibles externas y las relaciones entre ellas. La arquitectura abarca decisiones significativas sobre la organización del sistema de software, la selección de elementos estructurales y sus interfaces con sus comportamientos, la composición de estos elementos y subsistemas progresivamente mayores [4] [7].

La AS es considerada a grandes rasgos como la encargada de describir los componentes de un cierto y determinado sistema informático además de la relaciones entre ellos, es un marco muy amplio que describe su estructura. Es fundamental para lograr el éxito de un sistema. Por esta razón la AS constituye los pilares fundamentales donde se apoyará todo el proceso de desarrollo de un software.

Evaluar una arquitectura de software sirve para prevenir todos los posibles desastres de un diseño que no cumple con los requerimientos de calidad y para saber qué tan adecuada es la arquitectura de software diseñada para el sistema. Una buena regla para decidir cuándo hay que realizar una evaluación de una AS es cuando el equipo de desarrollo comience a tomar decisiones que dependan de la arquitectura y el costo de deshacer dichas decisiones sea mayor al costo de realizar la evaluación. Cuando se evalúa la AS no se tiene claramente una respuesta concreta donde se pueda decir a ciencia cierta si es buena o mala, ni tampoco se tiene una calificación. Sin embargo se puede identificar el riesgo, es decir fortalezas y debilidades de la arquitectura de software. Después de una evaluación de una AS, se pueden tomar algunas decisiones como: si se puede seguir el proyecto con las áreas de debilidad dadas en la evaluación, si hay que reforzar la Arquitectura de Software o si hay que comenzar de nuevo toda la AS [8] [9].

1.3 Definiciones de la calidad del software.

Según el Diccionario de la Real Academia Española (DRAE), es una propiedad o conjunto de propiedades inherentes de algo en específico, que permiten apreciarlo como igual, mejor o peor que las restantes de su especie. También es totalidad de las

características de un producto o servicio que le confieren su aptitud a satisfacer las necesidades expresadas o implícitas.

De acuerdo a 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 iniciales especificados por el cliente o usuario final [10].

La norma ISO/IEC 9126 define la calidad del software como el grado en que un producto de software satisface las necesidades explícitamente solicitadas y las necesidades implícitas esperadas por el cliente.

Por tanto se concluye que la calidad desde el punto de vista del software es la concordancia de este, producido con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo documentados explícitamente y con las características implícitas que se espera de todo software desarrollado profesionalmente. Teniendo en cuenta además que el uso de estándares y las normas de desarrollo permiten que se consiga la calidad.

1.3.1 Atributos de calidad

A grandes rasgos, se establece una clasificación de los atributos de calidad en dos categorías:

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

Algunos atributos de calidad por los cuales puede ser evaluada la arquitectura de software son:

Disponibilidad: Es la medida de disponibilidad del sistema para el uso.

Modificabilidad: Es la habilidad de realizar cambios futuros al sistema.

Rendimiento: 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.

La interoperabilidad: se define como la capacidad de intercambiar y compartir datos entre dos sistemas o componentes informáticos sin la intervención de un tercer sistema, de modo que la información o datos compartidos puedan ser utilizados sin requerir una comunicación previa [11].

La interoperabilidad: es necesaria para que los sistemas no estén aislados, es necesaria para que cooperen entre sí. Es la capacidad, conocimiento y acuerdo de dos o más partes de un todo para interoperar [12].

Según los criterios antes referenciados en cuanto al atributo de calidad interoperabilidad, se puede decir que es la capacidad que dos o más sistemas de intercambiar y compartir datos sin la intervención de un tercer sistema. La interoperabilidad es necesaria para que el subsistema de contabilidad actúe como tal, como una unidad centrada, brindando y recepcionando información que permite el buen funcionamiento económico de las demás entidades que se comunican constantemente con el subsistema.

1.3.2 Modelos de Calidad

Según el estándar ISO 8402 (1986), un modelo de calidad puede definirse como el conjunto de factores de calidad y de relaciones entre ellos, que proporciona una base para la especificación de requisitos de calidad y para la evaluación de la calidad de los componentes software. Los modelos de calidad se estructuran generalmente como una jerarquía (ya sea un árbol, ya sea un grafo dirigido), donde factores de calidad más genéricos, como eficiencia o usabilidad, se descomponen en otros más particulares, como tiempo de respuesta o facilidad de aprendizaje, probablemente en diversos niveles de descomposición [13].

- **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 estándar ISO-IEC 9126-1 propone un modelo jerárquico compuesto por características y subcaracterísticas (también conocidos como atributos y subatributos) en términos de calidad interna, calidad externa y calidad en el uso [14].
El estándar ISO/IEC 9126 distingue entre calidad interna y calidad externa, e introduce también el concepto de calidad en uso. La calidad interna tiene como objetivo medir la calidad del software mediante factores medibles durante su desarrollo. La calidad externa pretende medir la calidad del software teniendo en cuenta el comportamiento de este software en un sistema del cual forme parte. Finalmente, la calidad en uso corresponde a la calidad del software desde el punto de vista de un usuario [13].

1.3.3. Relación entre arquitectura de software y atributos de calidad

Para alcanzar un atributo específico, es necesario tomar decisiones de diseño arquitectónico que requieren un pequeño conocimiento de funcionalidad. Por ejemplo, el desempeño depende de los procesos del sistema y su ubicación en los procesadores, caminos de comunicación, etc. Por otro lado, establecen que al considerar una decisión de arquitectura de software, el arquitecto se pregunta cuál será el impacto de ésta sobre ciertos atributos; por ejemplo, modificabilidad, desempeño, seguridad, usabilidad, etc. Es por eso que cada decisión incorporada en una arquitectura de software puede afectar potencialmente los atributos de calidad. Cada decisión tiene su origen en preguntas acerca del impacto sobre estos atributos, y el arquitecto puede argumentar cómo la decisión tomada permite alcanzar algún objetivo [15].

La relación entre la arquitectura de software y los atributos de calidad no se encuentra sistemática y completamente documentada, debido a diversas razones:

- Existen atributos de calidad que, luego de ser estudiados durante años, poseen definiciones generalmente aceptadas. Sin embargo, existen algunos que carecen de definición, lo que inhibe el proceso de exploración de la relación en estudio.
- Los atributos no están aislados ni son independientes entre sí. Muchos atributos conforman un subconjunto de otro, es decir, se definen en función de otros atributos que lo contienen. Por ejemplo, el atributo disponibilidad puede ser un atributo por sí mismo; sin embargo, puede ser subconjunto de usabilidad y seguridad.
- El análisis de atributos no se presta a estandarizaciones, puesto que existen diferentes patrones con distintos niveles de profundidad, y resulta complicado establecer cuáles patrones se pueden utilizar para analizar calidad.

- Las técnicas de análisis son específicas para un atributo en particular. Por esta razón es difícil comprender la interacción entre varios análisis de atributos específicos [15].

Por ello es interesante considerar que tanto la arquitectura de un sistema de software como el sistema en sí mismo, se encuentran íntimamente relacionados con su entorno, por lo que es necesario tomarlo en cuenta para efectos del estudio de la calidad y la determinación de los atributos de calidad.

1.4 Evaluación de la arquitectura de software

La arquitectura es primordial para lograr la calidad del software que se desarrolla. Su evaluación permite mitigar los diferentes riesgos asociados con el desarrollo del software. Así como mejorar la visión de los procesos críticos, validar las decisiones de diseño que se tomaron. Además permite valorar los RNF (Portabilidad, Facilidad de Mantenimiento, Eficiencia, Usabilidad, Confiabilidad, Funcionalidad) sin esperar a que el software se construya.

1.4.1 Técnicas para evaluar la arquitectura de software.

Las técnicas utilizadas para la evaluación de atributos de calidad requieren grandes esfuerzos por parte del ingeniero de software para crear especificaciones y predicciones. Estas técnicas requieren información del sistema a desarrollar que no está disponible durante el diseño arquitectónico, sino al principio del diseño detallado del sistema. Las clasificaciones de las técnicas de evaluación de arquitectura se muestran en la figura 1.

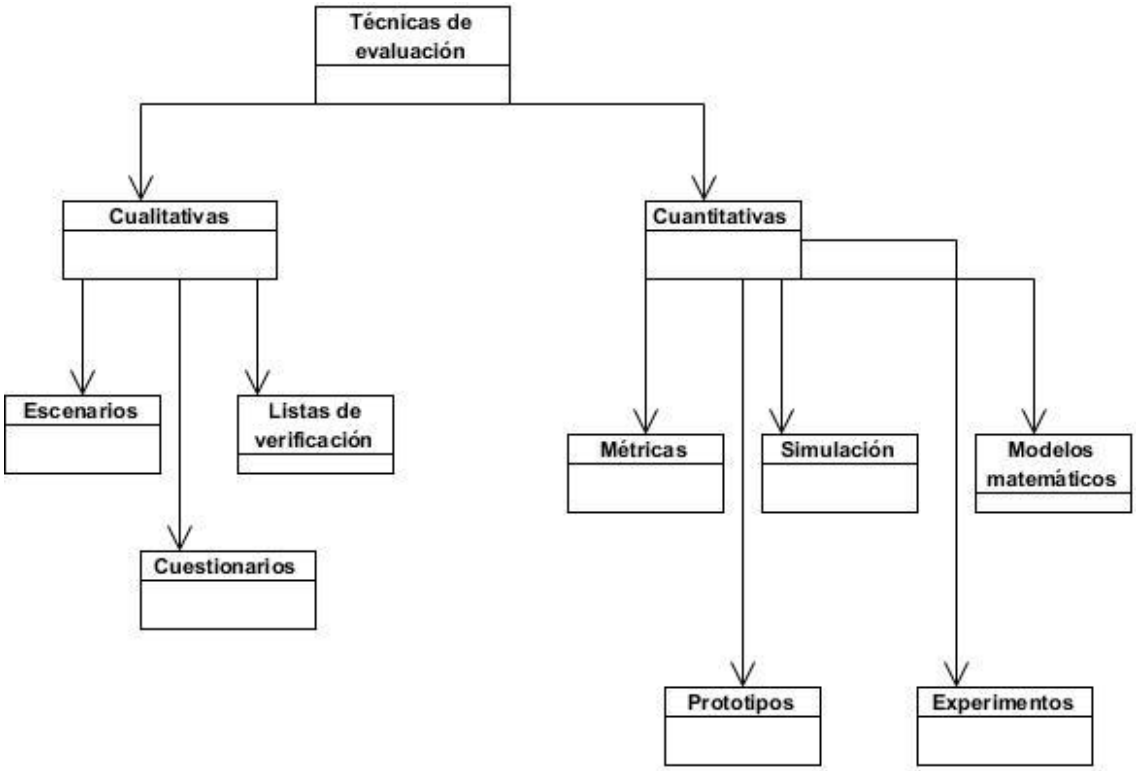


Figura 1: Clasificación de las técnicas de evaluación.

1.4.2 Evaluación basada en simulación.

La evaluación basada en simulación utiliza una implementación de alto nivel de la arquitectura de software. El enfoque básico consiste en la implementación de componentes de la arquitectura y la implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse. La finalidad es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. Una vez disponibles estas implementaciones, pueden usarse los perfiles respectivos para evaluar los atributos de calidad.

En términos de los instrumentos asociados a las técnicas de evaluación basadas en simulación, se encuentran los lenguajes de descripción arquitectónica y los modelos de colas.

1.4.3 Evaluación basada en modelos matemáticos.

La evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico y se presentan como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados, utilizando los

resultados de uno como entrada para el otro. El proceso de evaluación basada en modelos matemáticos sigue los siguientes pasos:

Selección y adaptación del modelo matemático. La mayoría de los centros de investigación orientados a atributos de calidad han desarrollado modelos matemáticos para medir sus atributos de calidad, los cuales tienden a ser muy elaborados y detallados, así como también requieren de cierto tipo de datos y análisis. Parte de estos datos requeridos no están disponibles a nivel de arquitectura, y la técnica requiere mucho esfuerzo para la evaluación arquitectónica, por lo que el arquitecto de software se ve obligado a adaptar el modelo.

Representación de la arquitectura en términos del modelo. El modelo matemático seleccionado y adaptado no asume necesariamente que el sistema que intenta modelar consiste de componentes y conexiones. Por lo tanto, la arquitectura necesita ser representada en términos del modelo.

Estimación de los datos de entrada requeridos. El modelo matemático aun cuando ha sido adaptado, requiere datos de entrada que no están incluidos en la definición básica de la arquitectura. Es necesario estimar y deducir estos datos de la especificación de requerimientos y de la arquitectura diseñada.

Predicción de atributos de calidad. Una vez que la arquitectura es expresada en términos del modelo y se encuentran disponibles todos los datos de entrada requeridos, el arquitecto está en capacidad de calcular la predicción resultante del atributo de calidad evaluado.

Cuando se habla de cómo evaluar siempre se hace referencia a las técnicas de evaluación como una categoría que determina la manera de medir el estado de una arquitectura [15].

1.4.4 Evaluación basada en 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. [15]

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 éstas por parte de las empresas desarrolladoras de software debido a que dicha norma definió métricas para medir cada una de las su características 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.
El beneficio que ofrecen éstas de ser adaptadas al sistema y de emitir un resultado que ayude a la toma de decisiones, es el objetivo que las sitúa como la técnica cuantitativa elegida para medir la interoperabilidad del subsistema Contabilidad.

Después de realizar un análisis de las técnicas de evaluación cuantitativas, se llegó a la conclusión que la técnica que más se ajusta para la realización de un plugin para evaluar la interoperabilidad en el subsistema Contabilidad, son las Métricas, ya que estas son usadas para medir la calidad de un producto a través de la medición de su comportamiento y a la vez emiten un resultado positivo o negativo para los arquitectos o líderes del proyecto. 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.

1.4.4.1 Métricas de interoperabilidad

Las métricas externas de interoperabilidad deben ser capaces de medir un atributo como es el número de funciones o la ocurrencia de la menor incomunicación que involucre a datos y comandos o instrucciones que sean transferidos entre el producto de software y otros sistemas, otros productos de software u otros equipos a los cuales está conectado.

Nombre de la métrica: Intercambiabilidad de datos, en base a su formato [16].

La métrica se propone medir: ¿Cuán correctamente ha sido implementado el intercambio de funciones de interfaces para una transferencia de datos específica?

Método de aplicación: Ejecutar las pruebas a cada registro de salida de las funciones

de interfaces de acuerdo con la especificación de los campos de datos. Cuente el número de formatos de datos que deben ser intercambiados con otro software o sistemas durante las pruebas en comparación con el número total.

Medición_ (fórmula): X = A / B

A- Número de formatos de datos intercambiados exitosamente con otro software o sistemas durante las pruebas del intercambio de datos.

B - Número total de formatos de datos a intercambiar.

Interpretación del valor obtenido: 0 <= X <= 1

A mayor cercanía al 1 resultará mayor intercambiabilidad.

Escala: Absoluta.

Nombre de la métrica: Intercambiabilidad de datos, en base éxito del intento [16].

La métrica se propone medir: ¿Cuán frecuentemente falló el intento de intercambio de datos entre el software objeto de la prueba y otro software?

¿Cuán frecuentemente es satisfactoria la transferencia de datos entre el software objeto de la prueba y otro?

Método de aplicación: Ejecutar las pruebas.

Cuente el número de casos en que las funciones de interfaces fueron usadas y fallaron.

Medición_ (fórmula): 1) X = 1 - A/B

A- Número de casos en que se falló al proceder a un intercambio de datos con otro software o sistemas.

B - Número de casos en que se intentó proceder a un intercambio de datos.

2) Y = A / T

T - Período de tiempo de operación

Interpretación del valor obtenido: 0 <= X <= 1

A mayor cercanía al 1 resultará mejor

0 <= Y

A mayor cercanía al 0 resultará mejor.

Escala: 1) Absoluta 2) Valorativa

1.6 Herramientas y Lenguajes de desarrollo.

La selección correcta de las herramientas y lenguajes 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 herramientas y lenguajes que serán utilizadas en el desarrollo de la aplicación.

1.6.1 Lenguaje de descripción de la arquitectura (ADL)

El lenguaje de descripción de la Arquitectura (ADL) es un lenguaje que proporciona elementos para modelar la arquitectura conceptual de un sistema software. Se encarga de la especificación explícita de componentes, conectores, configuraciones, herramientas del sistema y soporte [17].

1.6.1.2 Lenguaje de programación Java.

Java es un lenguaje de programación y la primera plataforma informática creada por Sun Microsystems en 1995. Es simple, seguro, portable y distribuido. Su principal característica es que es un lenguaje interpretado y compilado. Es un lenguaje orientado a objetos de propósito general. Su sintaxis es muy parecida a la de C y C++ pero hasta ahí llega el parecido. Java no es una evolución ni de C++ ni un C++ mejorado.[18]

Se escogió este lenguaje de programación java ya que es una fuente abierta, java es capaz de realizar la colección de basura de las ayudas, esto permite que la gerencia de memoria sea automática. Además se usó para una mejor integración con la herramienta a la cual será integrada que es AcmeStudio que está programado en este lenguaje, además de que se tiene un previo conocimiento básico del mismo que permite una fácil implementación sin necesidad de estudiarse otro lenguaje que no sea conocido.

Ventajas de Java:

- Fácil de aprender.
- Orientado a objetos.

- Independiente de plataforma.
- Es un lenguaje seguro.
- Es un lenguaje robusto.
- Es portable.

1.6.2 AcmeStudio.

AcmeStudio es una herramienta de diseño arquitectónico. Proporciona una interfaz gráfica que permite dibujar las arquitecturas de diferentes estilos 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 [19].

La motivación fundamental de Acme es el intercambio entre arquitecturas e integración de ADLs. La biblioteca AcmeLib define un conjunto de clases para manipular representaciones arquitectónicas Acme en cualquier aplicación. Su código se encuentra disponible tanto en C++ como en Java y puede ser invocada por lo tanto desde cualquier lenguaje la plataforma clásica de Microsoft [19].

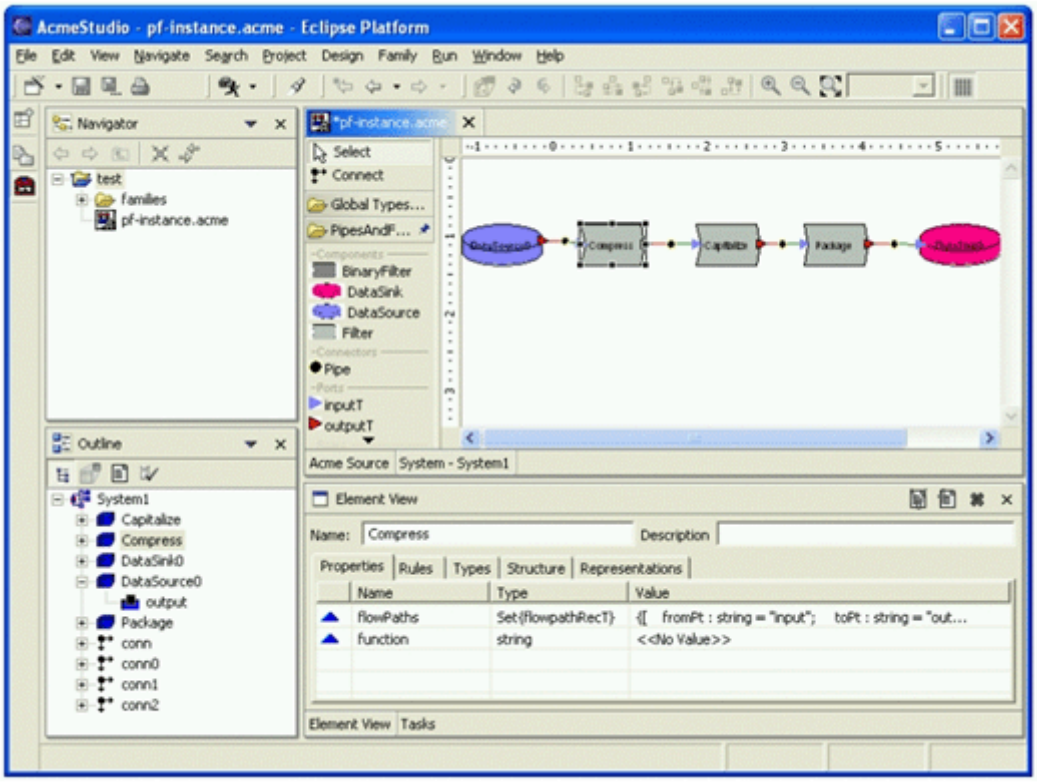


Figura 2: Ambiente de edición de AcmeStudio con diagrama de tubería y filtros.

1.6.2.1 Arquitecturas que soporta Acme.

Acme soporta la definición de cuatro tipos de arquitecturas: 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. [19]

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

Componentes: Representan elementos computacionales y almacenamientos de un sistema, un componente se define siempre dentro de una familia.

Interfaces: Todos los ADLs conocidos soportan la especificación de interfaces para sus componentes. En Acme cada componente puede tener múltiples interfaces. los puntos de interfaz se llaman puertos (ports). Los puertos pueden definir interfaces tanto simples como complejas, desde una asignatura de procedimiento hasta una colección de rutinas a ser invocadas en cierto orden.

Conectores: Los conectores representan interacciones entre componentes. Tienen interfaces que están definidas por un conjunto de roles. Los conectores binarios son los más 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, Acme 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 [19].

1.6.3 Entorno de desarrollo integrado (IDE)

Un entorno o ambiente integrado de desarrollo (del inglés Integrated Development Environment) 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 debugueadores, como se le conoce en inglés (debuggers) e integración con sistemas controladores de versiones o repositorios.

1.6.3.1 Eclipse

Eclipse es un entorno integrado de desarrollo de código abierto y multiplataforma, creado originalmente por la 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. El mismo permite la realización tanto de aplicaciones web como de aplicaciones de escritorios y existen complementos que permiten usar

Eclipse para programar en PHP, Perl, java, Python y C/C++. Además provee soporte para Java.

Se escogió como IDE de desarrollo el Eclipse ya que el entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés plugin) para proporcionar toda su funcionalidad al frente de la Plataforma de Cliente, rico a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. La arquitectura plugin permite escribir cualquier extensión deseada en el ambiente, como sería Gestión de la configuración. Eclipse provee al programador con Frameworks muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de Software, Aplicaciones web, etc. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de metadata en un espacio para archivos plano, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente. Además que la herramienta a la cual se integrará esta herramienta está basado en eclipse y así queda una mejor integración en la funcionalidad de la misma.

1.7 Metodología de desarrollo de software

Como metodología de desarrollo se utilizará Extreme Programming (XP) ya que es una metodología ágil para equipos de desarrollo de software pequeños o medianos. Elimina la gran documentación del sistema como un todo y ataca mínimas partes del sistema que son rápidamente implementadas. De esta forma el sistema va creciendo junto con el nuevo conocimiento del cliente y sus nuevas necesidades.

XP se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes. La Programación Extrema asume que la planificación nunca será perfecta, y que varía en función de cómo varíen las necesidades del negocio. Como es lógico, la planificación es iterativa: un representante del negocio decide al comienzo de cada iteración qué características concretas se van a implementar.

En la práctica se observa que la comunicación basada en metodologías ágiles potencia el desarrollo de los proyectos software, gracias a que el cliente casi siempre está en el sitio de trabajo, se genera una retroalimentación apoyada con reuniones diarias. Al estar diariamente con el equipo cliente se aprende mucho sobre el manejo de la empresa, procesos, temas relacionados y se realizan conexiones o relaciones laborales importantes para ambas partes.

Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se conoce en tiempo real. Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en los que es más importante. El código también es una fuente de retroalimentación, por ejemplo, las pruebas unitarias informan sobre el estado de salud del código. Ejecutar las pruebas unitarias frecuentemente permite descubrir fallos debidos a cambios recientes en el código.

Dentro de sus características principales se encuentran:

- **Integración del equipo de desarrollo con el cliente:** es recomendable que un representante del cliente trabaje junto al equipo de desarrollo.
- **Propiedad del código compartida:** no busca la división de las responsabilidades durante el desarrollo de los módulos en diferentes grupos de trabajo, sino que promueve que todo el personal pueda corregir y extender cualquier parte del proyecto.
- **Pruebas unitarias continuas:** Frecuentemente repetidas y automatizadas [20].

Ventajas de XP:

- Apropiado para entornos volátiles.
- Estar preparados para el cambio, significa reducir su coste.
- Planificación más transparente para nuestros clientes, conocen las fechas de entrega de funcionalidades ya que son vitales para su negocio.
- Permitirá definir en cada iteración cuales son los objetivos de la siguiente.
- Permite tener realimentación de los usuarios muy útil.
- La presión está a lo largo de todo el proyecto y no en una entrega final.

La metodología XP genera un conjunto de artefactos que son necesarios para lograr tener una información detallada del proyecto que se está realizando, en este caso en específico se desarrollarán las Historias de Usuario que son creadas a partir de las funcionalidades definidas, las pruebas de aceptación, pruebas unitarias y el plan de la entrega.

El ciclo de vida ideal consta de 6 fases:

Exploración: En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas y tecnologías.

Planificación de Entregas: En esta fase el cliente establece la prioridad de cada historia de usuario.

Iteraciones: Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado.

Producción: La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente.

Mantenimiento: Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento.

Muerte: Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura.

1.8 Ingeniería de Software Asistida por Computadora (CASE por sus siglas en inglés)

La herramienta CASE que se utilizará es el Visual Paradigm. La misma soporta el ciclo de vida completo del desarrollo de una aplicación informática, desde la fase de análisis hasta el despliegue del mismo, es una herramienta libre que permite el modelado de varios lenguajes de programación y otras tecnologías de forma fácil y asequible. Soporta todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Entre sus más recientes características se incluyen el modelado colaborativo con el sistema de control de versiones CVS4 y Subversión, así como la interoperabilidad con modelos UML2 (meta modelos UML 2.x para plataforma Eclipse) a través de XML de intercambio de metadatos (XMI o XML Metadata Interchange por sus siglas en inglés) [21].

1.9 Conclusiones del capítulo.

Tras el análisis de los elementos que caracterizan a la evaluación de arquitectura de software y las métricas estudiadas se puede arribar a las siguientes conclusiones:

1. Evaluar la arquitectura de software es recomendable para lograr una mejor calidad del producto.

2. La interoperabilidad es un atributo de calidad que desempeña un papel fundamental en los sistemas de contabilidad. A medida que estos sistemas crecen y se hacen complejos, crecen también las probabilidades de comunicación e intercambio de datos con otros sistemas, de ahí la importancia de una evaluación exitosa de la interoperabilidad.

3. Los modelos de calidad son elementos fundamentales para el proceso de evaluación, el modelo por el cual se va a guiar la investigación es el modelo ISO/IEC 9126. Este modelo cuenta con un prestigio a nivel internacional, además de ser el definido por la Oficina Nacional de Normalización para la producción de sistemas de software.

4. La evaluación de la interoperabilidad a través de métricas puede brindar una visión del estado de este atributo en la arquitectura y a su vez retribuir el proceso de desarrollo de dicha arquitectura, ya que las métricas se usan para evaluar un determinado atributo y proveen un valor que brinda una visión de su comportamiento.

5. El desarrollo del plugin se basará en la metodología XP, utilizando como lenguaje de programación Java, en el entorno de desarrollo Eclipse para una mayor integración con la herramienta donde se integrará el plugin.

Capítulo 2. Propuesta de solución

2.1 Introducción

El presente capítulo se centrará en enfatizar y aplicar la métrica de interoperabilidad identificada en el capítulo anterior, con el fin de describir si el sistema es interoperable o no, es decir si tiene buena transferencia de datos. Durante el desarrollo del capítulo se detalla el análisis desarrollado al modelado de la arquitectura del subsistema Contabilidad, para determinar los elementos significativos que pueden ayudar en el plugin a desarrollar, en busca de la realización de un proceso de evaluación de la interoperabilidad precisa y concreta. También se presentan los requisitos funcionales y no funcionales con los que debe cumplir la herramienta a desarrollar, así como la descripción del modelo de dominio para dicha herramienta

Descripción de la propuesta de solución.

Para el desarrollo del plugin, primeramente se debe modelar la arquitectura del subsistema contabilidad en un lenguaje de descripción de la arquitectura, el cual será AcmeStudio. Después se implementará la métrica, se emitirá un reporte y se integrará ese plugin al AcmeStudio. Se debe destacar que los valores de la métrica se obtendrán del modelo arquitectónico, puesto que el mismo además del modelo genera un código mediante clases y funciones el cual será objeto de información para obtener los valores de dicha métrica. Cabe destacar que para la implementación de la métrica se le agregarán propiedades al modelo obtenido en el AcmeStudio, ya que la métrica cuenta con indicadores que para poderlos obtener del modelo hay que agregárselos al mismo. Finalmente como se mencionaba anteriormente luego de la implementación de la métrica se emitirá un reporte que servirá de ayuda para valorar la arquitectura del subsistema y ese plugin obtenido se integrará al AcmeStudio.

2.2 Propuesta de solución

La obtención de un plugin que permita evaluar la arquitectura del subsistema de Contabilidad que forma parte del sistema de Planificación de Recursos Empresariales ERP, permitirá fortalecer el proceso de desarrollo que se lleva a cabo en el centro CEIGE para el desarrollo del ERP Cedrux.

El plugin a desarrollar tendrá como entrada el modelo de la arquitectura realizado en el AcmeStudio, una vez obtenido estos datos se trabaja en la implementación de la métrica utilizando dichos datos.

La presente tesis aportará:

- La especificación de la arquitectura del subsistema Contabilidad en un lenguaje ADL.
- La selección de una métrica capaz de medir la interoperabilidad de la arquitectura de dicho 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 de Dominio

Dada la propuesta de la realización del plugin para evaluar la interoperabilidad de la arquitectura del subsistema Contabilidad, en el AcmeStudio se propone la definición de un modelo de dominio el cual mostrará los principales conceptos a utilizar en el desarrollo de este complemento web.

2.3.1 Representación del modelo de dominio.

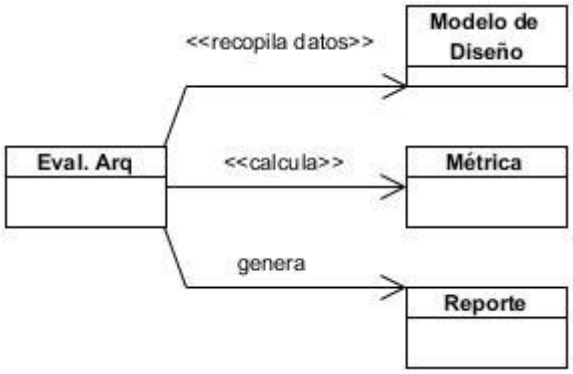


Figura 3: Modelo de Dominio.

2.4 Relación entre componentes.

Como parte de la solución, se diseñaron los componentes que integran el subsistema de contabilidad. A continuación se presenta una pequeña representación de la matriz de integración del subsistema contabilidad donde se evidencia la relación entre los componentes del mismo. A partir de esta tabla se desarrolló el modelo de la arquitectura del subsistema contabilidad realizado con la herramienta AcmeStudio, donde aparecen en la parte izquierda de la tabla los componentes que consumen servicios y en la parte superior los que brindan servicios como se muestra en la ilustración.

Tabla 1: Relación entre componentes

Brindan			
Consumen	Componentes	Comprobante Operaciones	Nomenclador Cuentas
	Comprobante Operaciones	ContCuent38 ContCuent15 ContCuent40	ContCuent08 ContCuent03 ContCuent10
	Nomenclador Cuentas	Contcuent29	

Componentes que ayudan a la realización del modelado.

Negocio: Están directamente relacionados con funcionalidades de negocio dado que implementan los requerimientos del mismo.[22]

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.[22]

Tecnológico: implementan funcionalidades puramente tecnológicas. Por lo general esta parte del sistema es responsabilidad de un marco de trabajo (del inglés *framework)* que establece los mecanismos de comunicación, caché, transacciones.[22]

Modelado de la arquitectura realizado en la herramienta AcmeStudio.

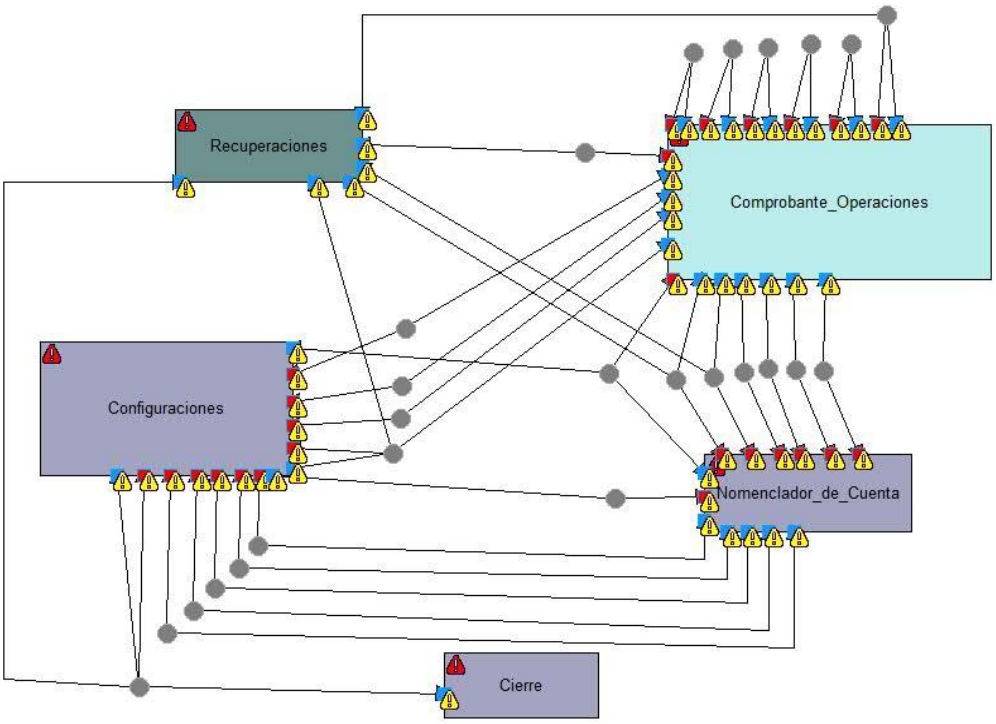


Figura 4: Modelo arquitectónico en el AcmeStudio.

En este modelado se representa la arquitectura del subsistema de contabilidad donde Cierre, Comprobante-Operaciones, Recuperaciones, Nomenclador-Cuentas, y Configuraciones son los componentes que conforman dicha arquitectura, la relación entre ellos está dada por los servicios que brindan y los servicios que consumen cada uno de esos componentes, en la tabla de relación de los componentes vista en el epígrafe anterior se representa la relación de estos componentes.

Ejemplo del código del modelo arquitectónico realizado en el AcmeStudio.

```
Component Comprobante_Operaciones: Domain = new Domain extended with {
    Port p0: InPut = new InPut extended with {
    }
    Port Contcuent38 : OutPut = new OutPut extended with {
    }
    Port p1: InPut = new InPut extended with {
    }
    Port ContCuent15: OutPut = new OutPut extended with {
    }
    Port p2: InPut = new InPut extended with {
    }
```

```
Port ContCuent40: OutPut = new OutPut extended with {
}
Port Contcuent37: OutPut = new OutPut extended with {
}
Port p4: InPut = new InPut extended with {
}
Port ContCuent39: OutPut = new OutPut extended with {
}
Port p5: InPut = new InPut extended with {
}
```

En este código se muestra uno de los componentes del modelado especificando que es de tipo dominio, donde por los puertos de entrada (InPut) consumen servicios y por los puertos de salida (Output) brindan un servicio. Este código es importante para poder calcular la métrica seleccionada, así como identificar los indicadores explicados en la misma.

2.5 Identificar puntos clave de la modelación para la realización de la herramienta con métricas.

Las métricas para software, como otras métricas, no son perfectas, sin embargo el diseño sin medición es una alternativa inaceptable. A continuación se mostrará y explicará una de las métricas de interoperabilidad que proporciona al diseñador una mejor visión interna y así el diseño evolucionará a un mejor nivel de calidad.

Nombre de la métrica: Intercambiabilidad de datos, en base a su formato

X=A/B.

Donde A es la cantidad de componentes de la arquitectura que son interoperables y B es la cantidad de componentes que tiene la arquitectura. Para la arquitectura mostrada en la ilustración 5.

Una vez modelada la arquitectura se realiza una evaluación necesaria para determinar las medidas claves para el cálculo de la métrica. Se deben comprobar los siguientes valores para calcular la métrica.

- Formato de datos de intercambio.
- Cantidad de datos recibidos.

- Componente sin conexión con otro componente.

A se hallará sumando los componentes que cumplan con el formato de datos de intercambio, mientras que los demás indicadores se utilizarán para emitir el reporte ayudando al arquitecto a encontrar algunos de los problemas en su arquitectura.

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

Si el valor de 0<=X<=0.6 implica que la arquitectura no es interoperable.

Si el valor de 0.7<=X<=1 implica que la arquitectura es interoperable.

En el desarrollo de esta métrica debemos destacar que los valores de A y de B cambiaron ya que las métricas investigadas en el capítulo anterior son para la evaluación de productos y en este caso lo que se tendrá es un modelo, se escogió una métrica de las propuestas y se le realizaron los cambios necesarios con ayuda de diferentes investigaciones del atributo interoperabilidad, así como entrevistas a especialistas en el tema.

2.6 Diagrama de clases del diseño.

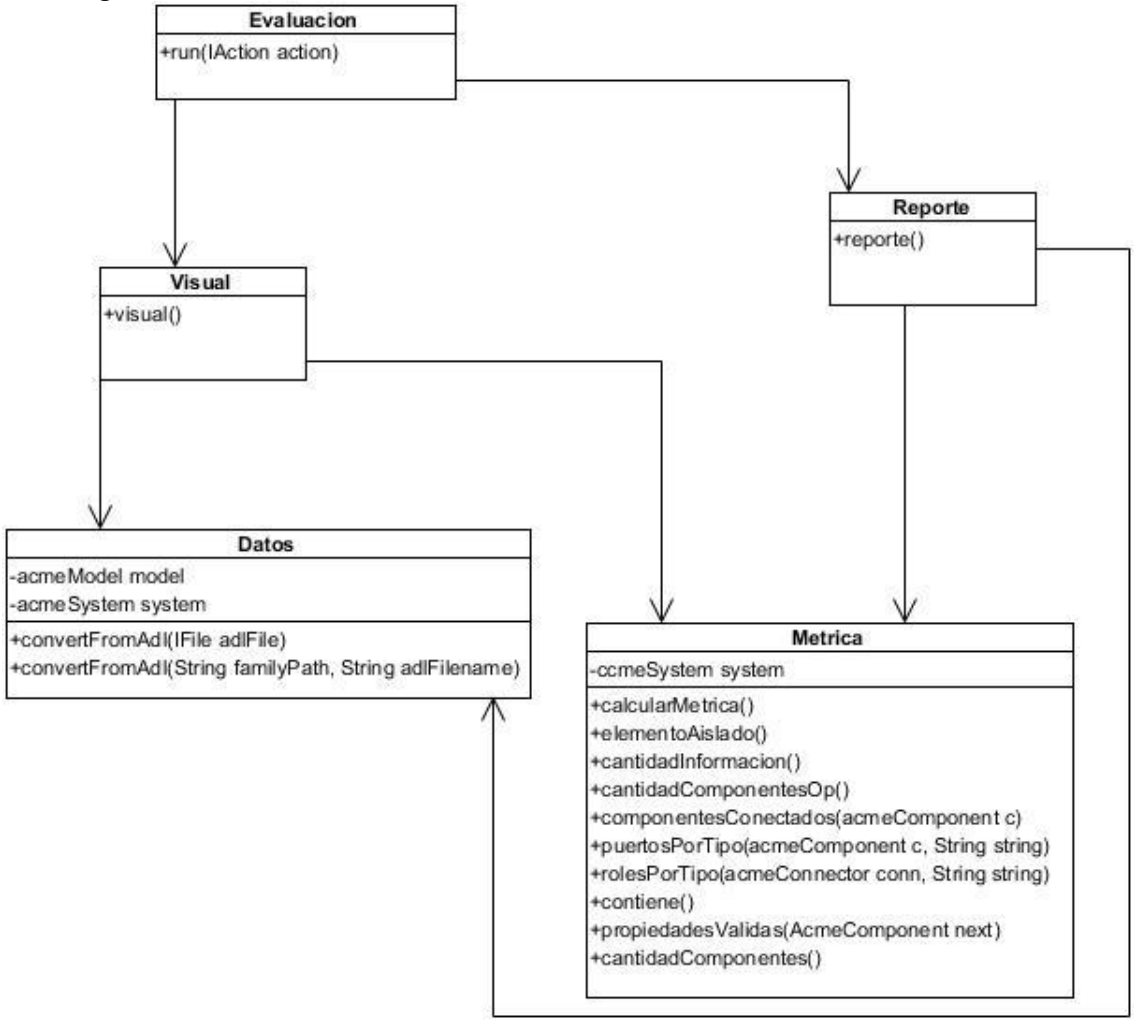


Figura 5: Diagrama de clases del diseño.

2.7 Patrones de diseño.

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

2.7.1 Patrones GRASP

Los Patrones Generales de Software de Asignación de Responsabilidades (GRASP por sus siglas en inglés) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. A continuación se describen los patrones básicos de asignación de responsabilidades utilizados en el desarrollo del sistema:

Experto: En la clase Evaluación se estableció la asignación de responsabilidades específicas por cada una de las clases del sistema, cuenta con las clases que poseen funcionalidades específicas atendiendo a la actividad que realizan y los procesos que gestionan.

Bajo Acoplamiento: En las clases Visual, Reporte, Métrica y Datos se evidencia este patrón ya que el intercambio de información se realiza a través de servicios implementados que se encargan de distribuir la información para la realización de procesos dependientes. Además están diseñadas para la menor dependencia posible entre ellas de manera que si se realiza una modificación en alguna no afecte el funcionamiento de las otras.

Alta Cohesión: El diseño y la dependencia entre clases están elaborados a partir de las funcionalidades que realizan, presentando alta relación de afinidad en las operaciones que controlan. En el caso de las actividades de alta complejidad comparten relación con otros objetos, disminuyendo la carga de transacciones entre ellas.

2.8 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.

2.8.1 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 interoperabilidad.
- RF2. Mostrar los resultados.

La descripción de cada requisito la podemos ver en las descripciones de las historia de usuario que se especifican más adelante.

2.8.2 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. [23]

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 Máquina Virtual de Java 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.

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 ésta sin dificultad.

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 de la herramienta se utilizará la metodología XP, usando el lenguaje de modelación UML y como herramienta para llevar a cabo el modelado Visual Paradigm.
- ✓ Para la implementación del componente se utilizará el IDE Eclipse.

2.10 Descripción de las historias de usuarios.

Representan una breve descripción del comportamiento del sistema, emplean terminología del cliente sin lenguaje técnico, se realiza una por cada característica principal del sistema, se emplean para hacer estimaciones de tiempo y para el plan de lanzamientos, reemplazan un gran documento de requisitos y presiden la creación de las pruebas de aceptación. [24]

Tabla 2: Historia de Usuario (Calcular Interoperabilidad).

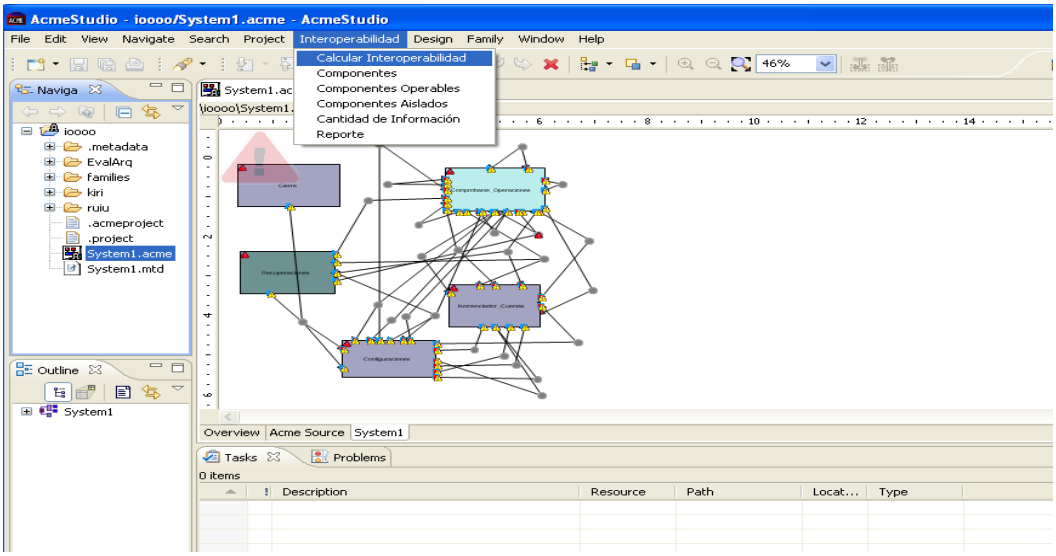
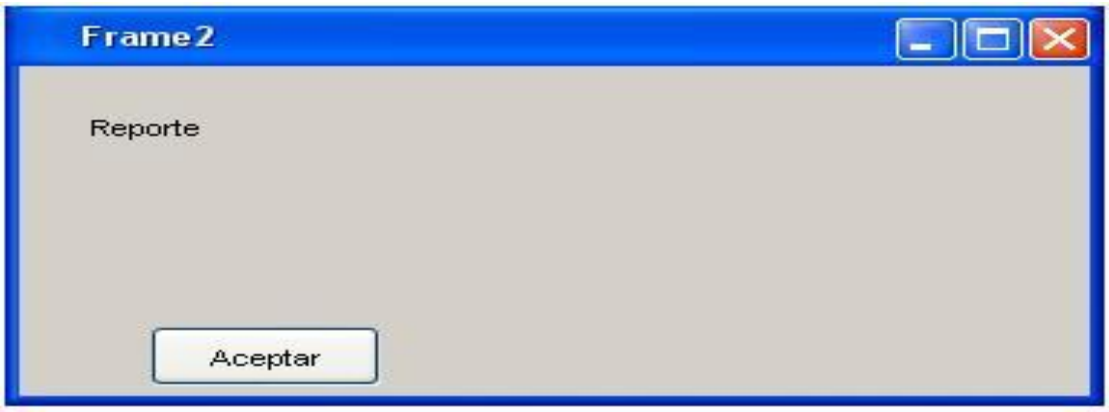
Historia de Usuario	
Código: *HU1*	**Nombre historia de usuario:** Calcular la interoperabilidad
Modificación de historia de usuario número: *ninguna*	
Referencia: *RF1*	
Programador: *Kirenia Garcia Marisi*	**Iteración asignada:** *primera*
Prioridad : *Alta*	**Puntos estimados:** *10 días*
Riesgo en desarrollo: *Alta*	**Puntos reales:** *2 semana*
Descripción: se calcula el atributo mediante la implementación de la métrica seleccionada.	
Prototipo de interface: 	

Tabla 3: Historia de Usuario (Mostrar los resultados).

Historia de Usuario	
Código: *HU2*	**Nombre historia de usuario:** Mostrar los resultados
Modificación de historia de usuario número: *ninguna*	
Referencia: *RF2*	
Programador: *Kirenia Garcia Marisi*	**Iteración asignada:** *primera*
Prioridad : *Alta*	**Puntos estimados:** *6 días*
Riesgo en desarrollo: *Alta*	**Puntos reales:** *1 semana*
Descripción: se muestra el resultado del cálculo realizado.	
Observaciones:	
Prototipo de interface: 	

2.9 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.

Tabla 4: Reserva del producto.

Prioridad	Ítem	Descripción	Estimación (días)	Estimado por:
Alta	1	Calcular la interoperabilidad.	10	PROG
Alta	2	Mostrar los resultados obtenidos.	6	PROG

2.11 Conclusiones del capítulo.

Durante el desarrollo de este capítulo se resaltan entre los principales resultados que:

1. Se llevó a cabo el desarrollo de los artefactos correspondientes a la metodología que se seleccionó, así como la descripción de estos.

2. Quedó definido los requisitos tanto funcionales como no funcionales que posibilitará su posterior implementación.

3. Se realizó el modelo arquitectónico en la herramienta AcmeStudio que brindará información para el desarrollo de la herramienta.

4. Para la fundamentación de los requisitos, se definieron los prototipos de interfaz de usuario, mostrándose así en las historias de usuario.

5. Este capítulo posibilitó fundamentar las bases para la realización de la propuesta de solución.

Capítulo 3. Implementación y prueba.

3.1 Introducción

En el presente capítulo se describen las fases de implementación y pruebas, propias de la metodología de desarrollo XP. Esta metodología plantea que la implementación de un software debe realizarse de forma iterativa, obteniendo al final de cada iteración un producto funcional que debe ser probado y mostrado al cliente para retroalimentar a los desarrolladores con la opinión de este. Se detallan las dos iteraciones realizadas durante la etapa de codificación del proyecto, así como las tareas de programación desarrolladas en cada una de estas y se hace referencia a las pruebas realizadas sobre el sistema. Además se realizan los artefactos generados por esta metodología como las historias de usuarios, pruebas de aceptación, pruebas unitarias y plan de entregas.

3.2 Implementación

Durante el inicio de cada iteración, se lleva a cabo una revisión del plan de iteraciones y se modifica de ser necesario. Las HU seleccionadas para cada entrega, son desarrolladas y probadas en un ciclo de iteración. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable. Así mismo, para cada HU se establecen las pruebas de aceptación. Estas pruebas se realizan al final del ciclo en el que se desarrollan, pero también al final de cada uno de los ciclos siguientes.

Por todo lo antes mencionado se decide realizar la aplicación en una iteración la cual se describe a continuación:

Iteración 1

En esta iteración se implementarán las historias de usuarios restantes, las cuales son: 2 y 3. Una vez que se termine de implementar estas funcionalidades ya se liberará la primera versión de la aplicación.

3.3 Plan de duración de la iteración

El plan de duración de cada una de la iteración se crea para una mayor organización del trabajo y como parte del ciclo de vida de un proyecto que utiliza la metodología XP. Este plan como objetivo tiene mostrar la duración de la iteración, así como el orden en

que serán implementadas las Historias de Usuarios en la iteración, esto ayuda a tener una aproximación del tiempo en que se desarrollará el plugin en su totalidad.

Tabla 5: Distribución de historias de usuario por iteración.

Iteraciones	**Orden de las HU a implementar**	**Duración total de las iteraciones**
1	Calcular interoperabilidad.	12 semanas
	Mostrar los resultados.	

3.4 Plan de entregas.

Como propuesta del plan de entregas se hicieron versiones al sistema en las fechas aproximadas que se indican a continuación:

Producto

Final 1ra Iteración 4ta semana de Mayo.

3.5 Pruebas.

Uno de los factores fundamentales de la metodología XP es el proceso de pruebas, el cual prueba constantemente tanto como sea posible. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. 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.

3.5.1 Pruebas unitarias.

En el transcurso de la implementación de un sistema, cada uno de los desarrolladores tiene que ir probando constantemente lo que va obteniendo para garantizar que las funcionalidades exigidas por el cliente estén siendo implementadas correctamente. A estas pruebas se le llama Pruebas Unitarias. Aunque estas no generan artefactos y no son directamente palpables para el cliente, son de vital importancia para el desarrollo de un proyecto.

Pruebas de Caja Blanca

Las pruebas de caja blanca se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente.

Mediante la prueba de la caja blanca el ingeniero del software puede obtener casos de prueba que:

- Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
- Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
- Ejecuten todos los bucles en sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

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

Existen diversas técnicas para el diseño de pruebas de caja blanca, como pruebas de camino básico y pruebas de estructura de control. Para diseñar una prueba utilizando la técnica de camino básico es necesario tomar el código del programa que se desea probar y generar un grafo, posteriormente se debe obtener la complejidad ciclomática del grafo y finalmente diseñar los casos de prueba con la información obtenida.

Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico.

La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática. Los pasos que se siguen para aplicar esta técnica de diseños de casos de prueba de caja blanca son:

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

Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Complejidad ciclomática.

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

Esta complejidad se pude 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 Ciclomática (CC) son:

CC= número de arcos – números de nodos + 2.

CC= número de nodos predicados + 1.

CC= número de regiones del grafo.

```
public String elementoAislado(){
 Set<AcmeComponent>componentes = system.getComponents(); ---------//1
 Iterator<AcmeComponent>it = componentes.iterator(); -------------//1
 String aislados = new String(); ---------------------------------//1

 while(it.hasNext()){---------------------------------------------//2
 AcmeComponent c = it.next(); ------------------------------------//3

  if(c.getPorts().size() == 0)-----------------------------------//4
      aislados += c.getName()+",";-----------------------------//5
  }

  return aislados;-----------------------------------------------//6
```

Figura 6: Representación del método elemtoAislado de la clase Métrica.

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.

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.

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

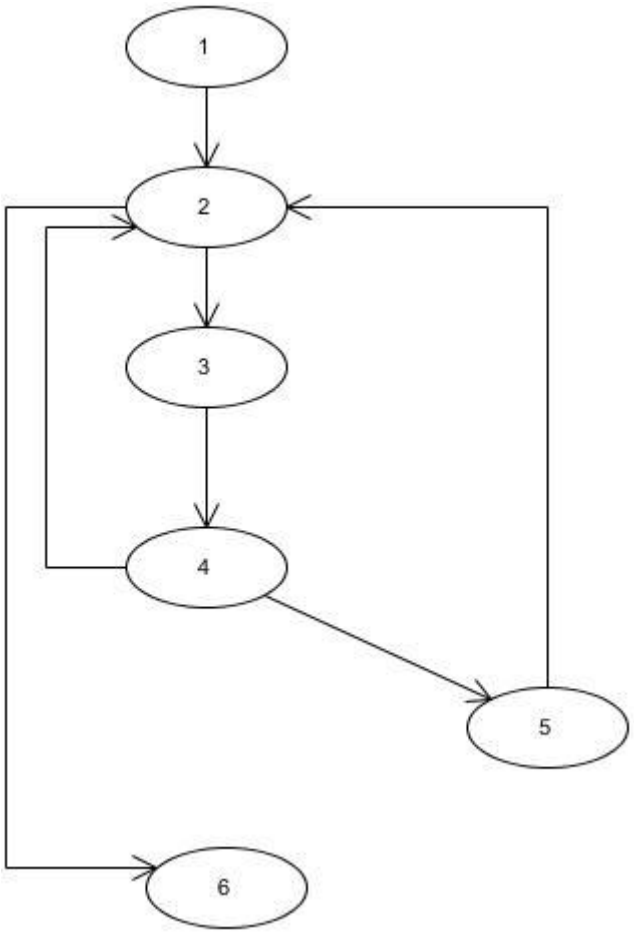


Figura 7: Grafo de flujo asociado al método elementoAislado.

CC=Aristas – Nodos + 2= 7-6+2=3.

CC=Número de regiones = 3.

CC=Nodos de predicado + 1=2+1= 3.

Caminos.

1-2-3-4-5-2-6.

1-2-3-4-2-6.

1-2-6

Casos de prueba.

Tabla 6: Caso de prueba para los caminos descritos anteriormente.

No. de caminos	Casos de prueba	Objetivo	Resultados esperados
1	elementoAislado ()	Si un componente no tiene puertos.	Se agrega el nombre al componente y se analiza el próximo.
2	elementoAislado ()	Si un componente tiene puertos	Continúa para el otro componente a analizar.
3	elementoAislado ()	Si no hay más componentes a analizar.	Se muestran los componentes aislados.

3.5.2 Pruebas de aceptación.

Las pruebas de aceptación son las realizadas por el cliente y usuarios finales de la aplicación. En estas fueron probadas las funcionalidades exigidas por el cliente y descritas en las historias de usuario. 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 del plugin y así la satisfacción del cliente con el desarrollo del mismo.

A continuación se muestran las pruebas de aceptación para cada una de las iteraciones definidas.

Tabla 7: 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 interoperabilidad.	
Condiciones de ejecución: Debe seleccionar la opción calcular interoperabilidad.	
Escenario de prueba 1	
Entradas: Selección de la opción calcular interoperabilidad.	
Resultado esperado: Se muestra el resultado de la operación.	
Evaluación: El resultado luego de ejecutada la prueba.	

Tabla 8: Caso de prueba de aceptación HU2.

Casos de Prueba	
Número de caso de prueba :CP_2	**Número de Historia de Usuario:HU2**
Nombre del caso de prueba: Mostrar los resultados	
Condiciones de ejecución: Debe haber una opción que diga reporte	
Escenario de prueba	
Entradas: Presionar la opción que dice reporte.	
Resultado esperado: Se muestra un reporte correspondiente con el resultado obtenido de calcular la interoperabilidad.	
Evaluación: El resultado luego de ejecutada la prueba.	

3.6 Métodos de expertos.

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 crean los métodos expertos, estos se basan en la consulta a un grupo de expertos para la validación de la propuesta. Los métodos 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.

Como pronóstico visionario es una profecía que usa ideas y juicios personales, vinculados entre sí.

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 ésta sea errónea. Así, un experto puede renunciar a la defensa de su opinión ante la persistencia del grupo en rechazarla.

3.6.1 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 sólo 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 años de experiencia.
- Conocimientos sobre proceso de pruebas.
- Que esté vinculado al desarrollo del software.
- Conocimientos de las aplicaciones Web.

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 Anexo 8)

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 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** = Σ de los valores que se obtienen de sustituir las cruces de la tabla 2 (Ver Anexo 8) por los valores correspondientes en su posición de la tabla de valores para el coeficiente de argumentación (Ver Anexo 9).

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:

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 método 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 6 fueron seleccionado para formar parte del grupo de validación de la propuesta, pues fueron aquellos cuyos resultados arrojaron un coeficiente de competencia alto y medio (Ver anexo 1).
A continuación se muestra una representación gráfica del coeficiente de competencia de los expertos.

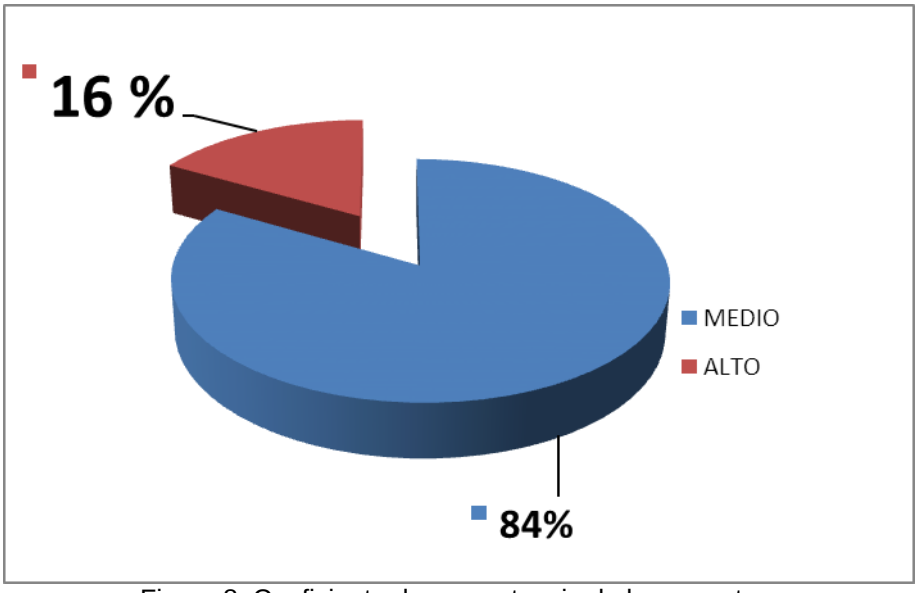


Figura 8: Coeficiente de competencia de los expertos.

3.7 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. Con las respuestas dadas por los expertos se podrá obtener la concordancia entre ellos y la aceptación y validez de la propuesta (Ver Anexo 11).

Indicadores a evaluar:

- La métrica propuesta en la solución.
- Valor científico de la propuesta.
- El tiempo de evaluación del atributo interoperabilidad mediante la realización del plugin.
- Grado de utilidad del plugin en cuanto a la toma de decisiones.

3.7.1 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 6.

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

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

$\overline{Rj}$ 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:

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

Donde **S** es la suma de los cuadrados de las desviaciones observadas de la media de

Rj (rangos) y se calcula:

$$\mathbf{S} = \sum_{j-1}^{n} (R_j - \overline{Rj})^2$$

Y $\overline{Rj} = \frac{\sum_{j=i}^{n} Rj}{N}$ es la suma de los rangos divido entre la cantidad de preguntas (Ver Anexo 1).

Sustituyendo $\overline{Rj} = \frac{\sum_{j=i}^{n} Rj}{N}$ se obtiene el valor: 54 / 4 = 13,5.

Luego se determina la desviación media **S** = 17.

Sustituyendo los valores en la ecuación Kendall se obtiene el valor de **W**= 0.0944. Este valor expresa el grado de concordancia entre los seis expertos al dar un orden evaluativo a los aspectos valorados. Este coeficiente siempre será positivo y su valor estará comprendido en el rango de 0 a 1.

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 se obtiene a través de la fórmula: $X^2 = K\ (N - 1)W = 1.7000$.

Después de calcular el Chi Cuadrado se procede a comparar el valor con el de la tabla estadística (Ver Anexo 9). Si se cumple que X^2 real < X^2 (α, N -1) entonces quiere decir que existe concordancia entre los expertos.

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

1.7000 < X2 (0.05, 5) = 1.7000 < 11.0705 lo cual afirma el cumplimiento de la comparación y por tanto la concordancia entre los expertos.

Desarrollo práctico y análisis de los resultados.

Los expertos seleccionados recibieron un resumen de la propuesta como documentación para contestar las preguntas, de igual manera recibieron la encuesta con un total de 4 preguntas, estas documentaciones fueron entregadas por vía e-mail, luego se confeccionaron tablas donde se recogen los resultados aportados por los expertos (Ver Anexo 3) .

Par la frecuencia absoluta acumulada (Ver Anexo 4).

Para la frecuencia relativa acumulada (Ver Anexo 5).

Para los puntos de corte (Ver Anexo 6).

La suma obtenida de las dos primeras columnas da los puntos de corte (Ver Anexo 6). 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:

Tabla 9: Categoría de cada aspecto.

Bastante adecuado	adecuado	No adecuado
0.25	1	

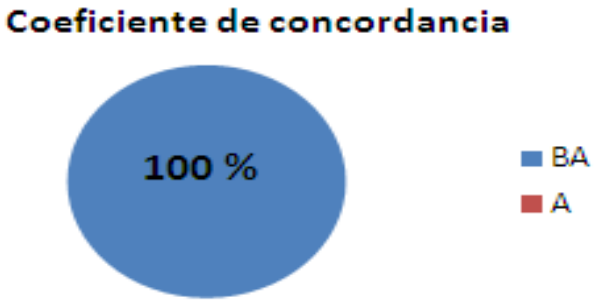


Figura 9: Coeficiente de concordancia.

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

- Menor o igual que 0,25 el nivel de adecuación es Bastante adecuado.
- Mayor que 0,25 menor o igual que 1 el nivel de adecuación es adecuado.
- Mayor que 1 el nivel de adecuación es No adecuado.

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

- Todas las preguntas fueron valoradas por los expertos de Bastante adecuado.
- El 100% de los expertos coinciden en la utilidad que tiene la puesta en práctica en el subsistema de Contabilidad de Cedrux.

En la siguiente gráfica quedó especificado claramente la reducción del tiempo en que se empleaba manualmente la evaluación del atributo de calidad interoperabilidad, ya que ésta evaluación demoraba alrededor de 60 minutos aproximadamente y con la realización del plugin ese tiempo logró reducirse a no más de un minuto.

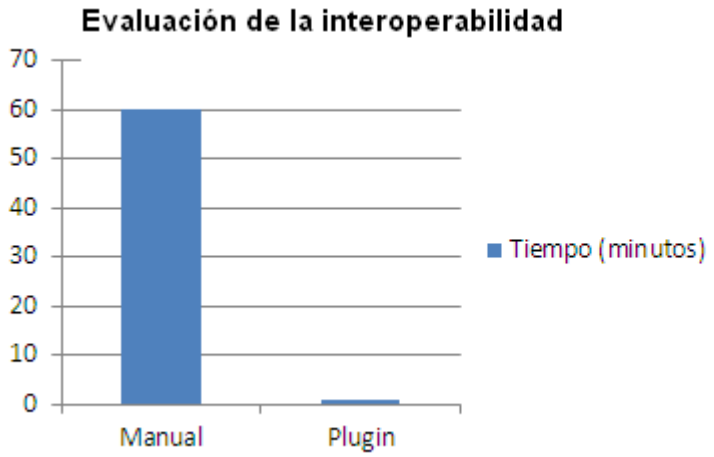


Figura 10: Comparación de tiempo en la evaluación de la interoperabilidad.

3.8 Conclusiones del capítulo.

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

1. Se realizó el plan de duración de las iteraciones para mostrar la duración de cada iteración, sirviendo de ayuda para tener una aproximación del tiempo en que se desarrollará el plugin en su totalidad.

2. Se desarrollaron las pruebas de aceptación mediante las que se pudo detectar las no conformidades detectadas en el sistema durante su desarrollo y reflejar la satisfacción del cliente con el producto desarrollado al final de una iteración.

3. Se realizó el plan de entregas para darle una mayor confianza y seguridad al cliente con respecto a las fechas en las que se harán las entregas del desarrollo de las iteraciones.

4. Se realizó el criterio de expertos el cual permitió validar si la propuesta de solución estaba adecuada para la problemática planteada, lo cual emitió un resultado satisfactorio deduciéndose así que la propuesta cumplía con el problema a resolver.

Conclusiones

Con la investigación que se realizó en este trabajo, luego de un estudio realizado para realizar un plugin para evaluar la interoperabilidad del subsistema de contabilidad, se arribó a las siguientes conclusiones:

1. Se realizó el marco teórico de la investigación relativo a las técnicas de validación de la arquitectura, lo cual permitió enfocar la realización del plugin en la técnica basada en métrica.

2. Mediante la modelación de la arquitectura del subsistema contabilidad en el lenguaje ADL, se pudo obtener los datos necesarios para el desarrollo del plugin.

3. Se realizó la implementación correspondiente, obteniendo un plugin con las funcionalidades necesarias para garantizar que se realicen correctamente las evaluaciones de los diseños arquitectónicos realizados por los arquitectos de un sistema desde el punto de vista de la interoperabilidad y reducir el tiempo en que se realizaba esta evaluación.

4. Se realizaron las pruebas funcionales al plugin para validar las funcionalidades que fueron implementadas, las cuales expusieron que los indicadores descritos en la descripción de la métrica cumplieron satisfactoriamente con los requisitos propuestos, garantizando su correcto funcionamiento.

5. Mediante el criterio de experto basado en el método Delphi se llegó a la conclusión que, la propuesta de solución es bastante adecuada para darle cumplimiento al problema planteado en esta investigación.

Sobre el plugin propuesto se concluye que:

1. El plugin permite evaluar la interoperabilidad del subsistema de contabilidad sin necesidad de hacerlo manualmente.

2. El plugin permite reducir el tiempo de evaluación de la arquitectura del subsistema de contabilidad, correspondiente al atributo interoperabilidad.

3. Genera un reporte con información referente al cálculo de la métrica, el cual permite detectar anomalías en el diseño, lo cual puede afectar el futuro desarrollo del sistema.

Recomendaciones

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

1. Realizar la implementación de nuevas métricas de interoperabilidad para evaluar el diseño de la arquitectura de software.
2. Generar nuevos reportes que tributen al levantamiento de información necesaria para los arquitectos.

Referencias bibliográficas

1. Amat, O.S., Pilar, *Contabilidad y gestión de costes*. 6ta ed. 2011, Barcelona: Profit Editorial.
2. Espinel, A.M.A. (2009) *Sistema de Gestión Financiera y Contable para optimizar los procesos de Administración en América Latina*
3. Breivold, H.P.C., I ; Larsson, M. (2012) *A systematic review of software architecture evolution research.*
4. Silvia, M. (2011) *Arquitectura de Software Parte 1*.
5. Billy, C. (2009) *Introducción a la Arquitectura de Software*
6. Pérez, J.A.A., Campanero *Desarrollo de Software y de las Arquitecturas Software.* 2010: p. 284.
7. Díaz, A.M., Leandro ; Sirerol, Daniel ; Oviedo, Sandra ;Ibáñez, Francisco S. (2012) *Software industrial flexible*. 5.
8. Rocha , A. (2009) *Innovación, Calidad e Ingeniería de Software.* **5**.
9. Germán, M.C., Diego (2010) *Evaluación de Arquitecturas de Software*. 49.
10. Dávila Nicanor ;Leticia ; Mejía Álvarez, P. *Evaluación de la Calidad de Software en Sistemas de Información en Internet.*
11. Manso, A.M., *Modelo de Interoperabilidad Basado en Metadatos (MIBM)*. 2010, Dirección general del Instituto Geográfico Nacional: Madrid. p. 20.
12. Monsalve, M. (2009) *Métricas para la interoperabilidad de la información en el Gobierno electrónico en paises en vía de desarrollo.*, 6.
13. Carvallo, J.P.F., Xavier ;Quer, Carmen (2007) *CALIDAD DE COMPONENTES SOFTWARE [pdf]. Versión preliminar capítulo 10*.
14. Galster, M.E., A; Moussavi, M. (2010) *Systematic selection of software architecture styles. [pdf]*
15. Camacho, E.C., Fabio (2004) *Arquitecturas de Software Guia de estudio*.
16. ISO/IEC, *Anexo B TABLAS CONTENTIVAS DE LAS MÉTRICAS*.
17. Gónzález, M. (2012) *Evaluacoón de la Arquitectura de Software*.
18. Belmonte , O. (2008) *Introducción al lenguaje de programación Java. Una guía básica*.
19. Reynoso , C.K., Nicolás (2012) *Lenguajes de de descripción de arquitectura*.
20. Acosta, D.G., Isael. Ciudad de La Habana : , *Desarrollo de una aplicación para la automatización de la evaluación de la Arquitectura de Software en los Proyectos Productivos de la UCI*. 2010, Universidad de las Ciencias Informáticas.

21. Nuviola, A., *Sistema para la Gestión de préstamos de implementos deportivos en la Universidad de las Ciencias Informáticas.* 2012, Universidad de las Ciencias Informáticas. La Habana.

22. Matos, Y.S., Nemury *ESTILO ARQUITECTÓNICO PARA EL SISTEMA INTEGRADO DE GESTIÓN CEDRUX [pdf]*.

23. Acosta, D.B., Isael . , *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 .

24. Villegas, A. (2007) *A propósito de programación extrema XP (eXtreme Programming).*

Anexos

Anexo 1

Experto	Kc	Ka	K	Competencia
1	1	1	1	ALTO
2	0,7	0,8	0,75	MEDIO
3	0,7	0,8	0,75	MEDIO
4	0,3	0,8	0,55	MEDIO
5	0,7	0,7	0,7	MEDIO
6	0,5	0,5	0,5	MEDIO

Coeficiente de expertos.

Anexo 2

	Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Rj
P1	2	2	2	2	2	2	12
P2	2	3	2	2	2	2	13
P3	3	3	3	3	3	2	17
P4	2	2	2	2	2	2	12

Datos obtenidos por los expertos.

Anexo 3

No.	BA	A	NA	Total
1	0	6	0	6
2	1	5	0	6
3	5	1	0	6
4	0	6	0	6

Frecuencia absoluta de las preguntas de las encuestas.

Anexo 4

No.	BA	A	NA
1	0	6	6
2	1	5	6
3	5	1	6
4	0	6	6

Frecuencia absoluta acumulada.

Anexo 5

No.	BA	A	NA
1	0,00	1,00	1,00
2	0,17	1,00	1,00
3	0,83	1,00	1,00
4	0,00	1,00	1,00

Frecuencia relativa acumulada.

Anexo 6

Puntos de cortes				**N**=0.625	
No.	**BA**	**A**	**Suma**	**P**	**N-P**
1	0,00	1,00	1.00	0.5	0.13
2	0,17	1,00	1.17	0.59	0.04
3	0,83	1,00	1.83	0.91	-0.29
4	0,00	1,00	1.00	0.5	0.13
Suma	1	4	5		
Puntos de corte	0.25	1			

Puntos de corte.

Anexo 7

		Brindan		
	Componentes	Comprobante Operaciones	Nomenclador Cuentas	Concisten
Consumen	Comprobante Operaciones	ContCuent38	ContCuent08	
		ContCuent15	ContCuent03	
		ContCuent40	ContCuent10	
		ContCuent39	ContCuent02	
		ContCuent41	ContCuent11	
		ContCuent37	ContCuent01	
	Nomenclador Cuentas	ContCuent29		

Anexo 8

Compañero (a):

En la presente tesis, se desea someter a la valoración de un grupo de expertos una propuesta de evaluación de la interoperabilidad en el subsistema de contabilidad de Cedrux, a través del modelado de su arquitectura, para garantizar una mejor calidad del software que se produce en el Centro de Informatización de Gestión de Entidades (CEIGE). 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: _____
Categoría docente: _____
Categoría científica: _____

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

El número 1 indica que no posee ningún conocimiento de la problemática en estudio, y el 10 indica que posee pleno conocimiento.

1	2	3	4	5	6	7	8	9	10

Tabla 1: Grado de conocimiento del experto

- 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.

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 9

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 de valores asignados por el Método Delphi.

Anexo 10

TABLA 3-Distribución Chi Cuadrado χ^2

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

ν/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	18,8679	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 11

La Universidad de las Ciencias Informáticas (UCI) cuenta con un centro de Informatización de Gestión de Entidades (CEIGE), en el cual se desarrolla el Sistema de Planificación de Recursos Empresariales (ERP) denominado Cedrux. La forma de evaluar la arquitectura para lograr una adecuada calidad en Cedrux, es mediante la revisión de diferentes artefactos de forma manual, revisando diferentes tablas que contienen información de la misma, la documentación de toda la arquitectura está registrada en Excel, lo cual no permite un fácil análisis de la información, no existe una métrica hoy en día que se utilice para desarrollar la evaluación del atributo de calidad interoperabilidad, por todo lo antes mencionado conlleva que la toma de decisiones con respecto al rediseño de la arquitectura y los posibles cambios que puedan ocurrir en ella se demore, por lo cual puede incumplir en el tiempo pactado con el cliente, así como el rediseño de la misma.

La problemática planteada permite definir el **problema a resolver** ¿Cómo reducir el tiempo en que se logra evaluar cuantitativamente la interoperabilidad para agilizar el

proceso de toma de decisiones relacionadas con la arquitectura del subsistema Contabilidad de Cedrux?

Para la solución del problema expuesto se define como **objetivo general:** Desarrollar un plugin para medir cuantitativamente la interoperabilidad del subsistema Contabilidad de Cedrux, a partir de la modelación de su arquitectura, de manera que se reduzca el tiempo en que se lleva a cabo esta actividad y se agilice la toma de decisiones relacionadas con la arquitectura del subsistema de contabilidad de Cedrux.

Propuesta de solución

La obtención de un plugin que permita evaluar la arquitectura del subsistema de Contabilidad que forma parte del sistema de Planificación de Recursos Empresariales ERP, 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 Contabilidad en un lenguaje ADL.
- La selección de una métrica capaz de medir la interoperabilidad de la arquitectura de dicho subsistema.
- El desarrollo de un plugin que implementa la métrica seleccionada facilitando su uso en los distintos momentos que se necesite ponerla en práctica.

Las métricas para software, como otras métricas, no son perfectas, sin embargo el diseño sin medición es una alternativa inaceptable. A continuación se mostrará y explicará una de las métricas de interoperabilidad que proporciona al diseñador una mejor visión interna y así el diseño evolucionará a un mejor nivel de calidad.

Nombre de la métrica: Intercambiabilidad de datos, en base a su formato

X=A/B.

Donde A es la cantidad de componentes de la arquitectura que son interoperables y B es la cantidad de componentes que tiene la arquitectura. Para la arquitectura mostrada en la ilustración 5.

Una vez modelada la arquitectura se realiza una evaluación necesaria para determinar las medidas claves para el cálculo de la métrica. Se deben comprobar los siguientes valores para calcular la métrica.

- Formato de datos de intercambio.
- Cantidad de datos recibidos.
- Componente sin conexión con otro componente.

A se hallará sumando los componentes que cumplan con el formato de datos de intercambio, mientras que los demás indicadores se utilizarán para emitir el reporte ayudando al arquitecto a encontrar algunos de los problemas en su arquitectura.

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

Si el valor de 0<=X<=0.6 implica que la arquitectura no es interoperable.

Si el valor de 0<=0.7<=1 implica que la arquitectura es interoperable.

Para un mejor entendimiento del atributo a evaluar cabe destacar que es la interoperabilidad, para un mejor conocimiento en el tema.

Según los criterios referenciados en cuanto al atributo de calidad interoperabilidad, se puede decir que es la capacidad que dos o más sistemas de intercambiar y compartir datos sin la intervención de un tercer sistema.

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

No.	Preguntas	BA	A	NA
1	¿Cómo considera usted la ecuación **X=A/B** para el cálculo de la interoperabilidad?			
2	Partiendo de la propuesta del presente trabajo cómo considera usted la realización del plugin para reducir el tiempo de evaluación de la interoperabilidad de la arquitectura.			
3	¿Qué grado de utilidad usted le concede a la realización del plugin para la toma de decisiones con respecto a la arquitectura desde el punto de vista de la interoperabilidad durante el desarrollo de los subsistemas que se desarrollan de Cedrux?			

4	¿Cómo valora usted el valor científico de la propuesta?			

Anexo 12

Lista de los expertos.

Msc. Pedro M Nogales Cobas.

Ing. Yusnier Matos Áreas.

Msc. Osmar Leyet Fernández.

Ing. Leandro Pérez-Borroto Vivero.

Ing. Omar Antonio Díaz Peña.

Ing. William González Obregón.

Glosario de términos

AcmeStudio: Herramienta de diseño arquitectónico.

ADL: Lenguaje de Descripción Arquitectónica.

Arquitectura de Software: Considerada a grandes rasgos como la descripción de los componentes de un sistema informático y las relaciones entre ellos. Además constituye un amplio marco que describe su estructura, componentes, su forma y la manera que estos interactúan. Es la base fundamental para lograr el éxito de un sistema.

Calidad del software: La calidad desde el punto de vista del software es la concordancia de este, producido con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo documentados explícitamente y con las características implícitas que se espera de todo software desarrollado profesionalmente.

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

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

Interoperabilidad: Es la capacidad que dos o más sistemas de intercambiar y compartir datos sin la intervención de un tercer sistema.

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

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

RF: Requerimientos Funcionales.

RNF: Requerimientos No Funcionales.

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