Universidad de las Ciencias Informáticas.

Facultad 3

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

Título: Pyramide: plugin del Eclipse para la evaluación de la estabilidad en los subsistemas de Cedrux.

Autor: Adrian Agustín Bahy Pupo

Tutores: Ing. Dinia Zayas Romero

Ing. Leandro Pérez-Borroto Vivero

La Habana, Cuba.

Junio 2013

Declaración de Autoría

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____

Adrian Agustín Bahy Pupo

Autor

_____ _____

Ing. Dinia Zayas Romero

Tutor

Ing. Leandro Pérez-Borroto Vivero

Tutor

"Es posible considerar que la medición es un desvío. Un desvío necesario, porque la mayoría de los seres humanos no son capaces de tomar decisiones claras y objetivas sin apoyo cuantitativo."

Horst Zuse

Agradecimientos

A las personas que me dieron el nacer, me hicieron reír, me hicieron crecer y me guiaron, para hoy estar donde estoy, a todos los que me enseñaron cuanto se y a ser quien soy, a los que siempre me recordaron de dónde vengo y a dónde voy, a todas esas personas, mil gracias hoy yo les doy.

A mi abuelita Sílfida; agradecido infinita y eternamente, vives en mí. A mis padres por el cariño, la confianza y el apoyo durante todos mis años. A mi hermanita, por su ternura y su constante preocupación por mí. A mi abuelo Baltazar, mi gran rey mago, por sus constantes muestras de aprecio. A mis tíos y primos por hacerme participe de sus vidas. A mi novia y a su familia, por el amor que me ofrecen.

A Rafa, mi hermanito en estos 5 años, por su apoyo en los momentos difíles y por estar siempre ahí, a solo una persiana. A Idel, por ayudarme siempre que lo necesito. A Pedro, mi iniciador en el mundo de la ISW y compañero de maratones, por sus consejos, su confianza y su amistad. A Ariadna, mi fiel amiga en estos dos últimos cursos, por su amistad y aprecio.

A mis tutores, por su apoyo y total confianza, especialmente a Dinia, por su amistad y constante preocupación. A Matos, por estar siempre ahí, para las dudas y para ofrecerme sus buenos consejos. A mis compañeros de aula durante estos cinco años. A las personas que he conocido en este último curso y me han convertido en parte de sus vidas.

Dedicatoria

A mi abuelita querida, la persona más especial que he conocido en la vida, quien en este momento se sentiría muy feliz y orgullosa. Gracias por tu tiempo.

Tu hijo más chiquito

RESUMEN

La volatilidad de los requisitos de software hace que los sistemas integrales de gestión deban estar preparados constantemente para la realización de cambios. Estos tienen su impacto en la arquitectura de software definida, por lo que resulta de gran relevancia el conocimiento de cómo va a reaccionar el sistema luego de las modificaciones realizadas. Sobre este aspecto se define la Estabilidad, cuyo objetivo es dotar al sistema definido de la capacidad para enfrentar el proceso de evolución minimizando la ocurrencia de efectos negativos.

El presente trabajo tiene como objetivo desarrollar una herramienta que provea información referente al grado de estabilidad de los subsistemas[1] de Cedrux para facilitar la toma de decisiones arquitectónicas. Esta herramienta es desarrollada como plugin[2] para el Entorno de Desarrollo Integrado Eclipse e integrada con la herramienta Acme Studio, la cual es una herramienta gráfica de modelado del Lenguaje de Descripción Arquitectónica Acme. Hace uso de métricas para medir la estabilidad y recibe como entradas, modelos arquitectónicos confeccionados en Acme Studio. Los resultados del trabajo se validan mediante la realización de pruebas, y la aplicación de la técnica Criterio de expertos para la evaluación de las variables de investigación.

PALABRAS CLAVE

AcmeStudio, arquitectura de software, estabilidad, métricas, plugin

[1] Describe una parte del sistema que encapsula comportamiento, expone un conjunto de interfaces y empaqueta otros elementos de modelo

[2] Módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande.

Tabla de Contenido

Tabla de contenidos

DECLARACIÓN DE AUTORÍA I
RESUMEN IV
INTRODUCCIÓN 1
CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA 6
1.1 INTRODUCCIÓN 6
1.2 ARQUITECTURA DE SOFTWARE 6
1.3 ATRIBUTOS DE CALIDAD 8
1.4 ESTABILIDAD 8
1.4.1 Consideraciones generales de los conceptos relacionados con la estabilidad 9
1.5 EVALUACIÓN DE ARQUITECTURA DE SOFTWARE 10
1.5.1 Especificaciones de la arquitectura que se desea evaluar 10
1.6 TÉCNICAS DE EVALUACIÓN DE ARQUITECTURAS DE SOFTWARE 12
Evaluación basada en métricas 12
1.7 CORRIENTES PARA LA EVALUACIÓN DE LA ESTABILIDAD 13
1.7.1 Métricas para la evaluación de la Estabilidad 14
Consideraciones generales de las métricas estudiadas 16
1.9 METODOLOGÍA DE DESARROLLO DE SOFTWARE 17
Extreme Programming 18
SCRUM 19
OpenUP 19
Consideraciones generales de las metodologías estudiadas 21
1.10 LENGUAJES Y HERRAMIENTAS 22
LENGUAJES DE DESCRIPCIÓN ARQUITECTÓNICA 22
Acme 24
LENGUAJE DE MODELADO UNIFICADO 24
LENGUAJES DE PROGRAMACIÓN 25
ACME STUDIO 25
VISUAL PARADIGM 26
ENTORNO DE DESARROLLO INTEGRADO 26
ECLIPSE 26

Tabla de Contenido

PLUGINS Y PUNTOS DE EXTENSIÓN 27
1.11 CONCLUSIONES 28
CAPÍTULO 2: DISEÑO E IMPLEMENTACIÓN 29
2.1 Introducción 29
2.2. Propuesta del sistema 29
2.3. Productos de trabajo propuestos por OpenUP 29
2.3.1. Visión Pyramide 29
2.3.2. Descripción de los requisitos del software 30
2.3.3. Requisitos Funcionales 31
2.3.4. Requisitos no Funcionales 34
2.3.5. Diagrama de casos de uso del sistema y diagrama de paquetes 35
2.3.6. Diagrama de clases del diseño 38
2.3.7. Descripción de las clases del diseño 39
2.4. Patrones de diseño utilizados 40
2.5. Métricas de validación del diseño 41
2.6. Descripción de implementación por funcionalidades 47
2.7. Conclusiones del capítulo 49
CAPÍTULO 3: VALIDACIÓN Y PRUEBAS 50
3.1. Introducción 50
3.2. Método Delphi 50
3.3. Pruebas de Software 56
3.3.1. Pruebas de caja blanca 56
3.3.2. Pruebas de caja negra 60
3.4. Despliegue de la solución 65
3.5. Aplicación de Pyramide en dos subsistemas de Cedrux 65
3.6. Decisiones arquitectónicas 67
3.7. Conclusiones 68
CONCLUSIONES GENERALES 69
RECOMENDACIONES 70
BIBLIOGRAFÍA 71

INTRODUCCIÓN

Las últimas décadas han sido matizadas por el creciente y notorio avance de la Industria del Software a nivel mundial. La competencia se hace inminente, dejando en total evidencia la necesidad de desarrollar productos donde, a través de la arquitectura de software definida, pueda controlarse el proceso de evolución y los desafíos que este representa. Tal reto requiere de un gran esfuerzo por parte del equipo de desarrollo debido a que el proceso de evolución o mantenimiento del software exige enormes costos. En alusión a esto varios autores del ámbito de la ingeniería de software aseveran que las empresas gastan más recursos en el proceso de mantenimiento que en el desarrollo inicial (Jazayeri, 2002).

En correspondencia con el aumento de la complejidad del software a realizar, estará también su proceso de desarrollo, en tal sentido y para lograr la calidad de dicho proceso, se considera necesario brindarle un lugar importante a la definición y evaluación de la arquitectura de software, debido a que este elemento permite a un nivel de abstracción significativo analizar la estructura y composición del sistema (Rick Kazman, 2007).

La evaluación arquitectónica incide directamente en la calidad del producto final, la realización de una valoración del comportamiento de los atributos de calidad requeridos desde el diseño arquitectónico y la identificación de posibles riesgos inherentes a la selección de una arquitectura determinada así lo ameritan (Rick Kazman, 2007). Los atributos de calidad constituyen elementos de alta relevancia a ser considerados en la arquitectura, debido a que estos tienen un gran impacto en el desarrollo y mantenimiento del sistema influyendo notoriamente en la calidad del mismo (Vera, 2006).

Aparejado al proceso de evolución de todo producto de software e incidiendo de forma directa en la arquitectura del mismo se encuentra el atributo de calidad **mantenibilidad,** definido por la norma ISO/IEC 9126 como: ‘’*la capacidad del producto de software de ser modificado, donde las modificaciones pueden incluir las correcciones, mejoras o adaptaciones del software a cambios en el ambiente, así como en los requisitos y las especificaciones funcionales*’’(Normalización, 2005). Este atributo de calidad engloba cinco subatributos que contribuyen a un correcto proceso de modificaciones, expresando en cada caso capacidades específicas que debe poseer el producto. Esta investigación solo se centra en el subatributo **estabilidad**, la cual es definida por la norma anterior como: ‘’*la capacidad del producto de software para minimizar efectos inesperados cuando se realizan*

modificaciones" (Normalización, 2005). Esta definición sitúa a la **estabilidad** como una subcaracterística importante en el ámbito de la **mantenibilidad** y como primer elemento a tener en cuenta para realizar modificaciones al software.

Haciendo alusión a lo planteado con anterioridad, puede decirse que todo proceso de evolución de software requiere de una arquitectura estable que soporte los cambios evolutivos y sea capaz de permanecer intacta la estabilidad del sistema (Jazayeri, 2002). Estudios realizados por el propio Jazayeri, Bahsoon y Emmerich revelan la importancia de la obtención de datos precisos que muestren el grado de estabilidad que posee la arquitectura definida y contemplan el hecho de que contar con una arquitectura estable es el pilar fundamental para realizar el proceso de evolución del software (Jazayeri, 2002) (Rami Bahsoon, 2006).

El Centro para la Informatización y Gestión de Entidades (CEIGE) perteneciente a la Universidad de las Ciencias Informáticas (UCI) desarrolla varias soluciones informáticas para distintos sectores de la economía cubana. Una de ellas es el Sistema Integral de Gestión Cedrux, concebido por iniciativa del Ministerio de Finanzas y Precios (MFP) en aras de alcanzar el fortalecimiento de la gestión de entidades. Cedrux constituye un paquete de soluciones integrales de gestión para las entidades presupuestadas y empresariales, el cual se desarrolla bajo los principios de independencia tecnológica y su correspondencia con las características y procesos de la economía cubana. Este sistema está constituido para ser integrado, modular y adaptable. La característica de integrado está asociada a permitir controlar los diferentes procesos de la organización y sus relaciones, por lo que los datos que se ingresan solo una vez en uno de los módulos deben poder ser empleados por los restantes. La característica modular está dada por la capacidad que tenga el sistema de desacoplarse para instalar o no los módulos dependiendo de las solicitudes de los clientes, y la característica adaptable está dada por la capacidad que tenga el sistema de adecuarse a los requisitos específicos de la empresa. (Suárez, 2008). Esta última característica según Codorniú y Suárez es el valor añadido fundamental con el que deben contar los sistemas integrales de gestión (Codorniú, 2010).

En el contexto de Cedrux, la adaptabilidad se evidencia cuando los clientes se interesan en partes específicas del sistema, es decir, por subconjuntos de funcionalidades que este ofrece. Incluso pueden estar interesados en contar con la totalidad de las funcionalidades que ofrece el software. Sin embargo, en este hay procesos implementados que no responden completamente al modo en que se desarrollan en sus entornos o empresas. En estos casos, es necesario realizar adaptaciones que por

lo general conllevan a modificaciones en la estructura de componentes, lo cual implica la realización de los siguientes cambios:

- ✓ Adición de funcionalidades para abarcar nuevos elementos relacionados con aspectos operativos o productivos de las entidades presupuestadas y empresariales.
- ✓ Adición de módulos por solicitudes específicas de adaptación a requerimientos particulares del cliente.
- ✓ Modificación de módulos existentes para adaptarse a las particularidades de las entidades presupuestadas y empresariales.
- ✓ Adición, modificación y/o eliminación de funcionalidades durante las pruebas a las que es sometido el sistema como parte de su ciclo de vida.

Estos cambios pueden partir además de las solicitudes de mejoras o identificación de errores durante el soporte y mantenimiento de versiones desplegadas. Sin embargo, su realización genera afectaciones en la estructura de componentes durante el proceso de verticalización[3], debido a los pocos elementos con los que se cuenta para determinar el impacto que estos producen en la arquitectura. Unido a esto, el equipo de arquitectura de Cedrux no tiene definidas en su marco de evaluación, técnicas dirigidas a evaluar la estabilidad de sus subsistemas y no se realiza una correcta explotación de las potencialidades del lenguaje de descripción arquitectónica Acme, seleccionado por el centro CEIGE para modelar la arquitectura. Como consecuencia de esto, los arquitectos se encuentran privados de obtener resultados cuantitativos que reflejen el grado de estabilidad que poseen los subsistemas de Cedrux, atributo que tiene gran peso en la implementación de las verticalizaciones y las decisiones arquitectónicas que en este sentido puedan tomarse.

La problemática planteada anteriormente se ve reflejada en el siguiente **problema** a resolver: ¿Cómo obtener datos precisos sobre la estabilidad de los subsistemas de Cedrux para apoyar la toma de decisiones durante el proceso de verticalización?

El **objeto de estudio** se enmarca en la evaluación de arquitecturas de software y el **campo de acción** está dirigido a la evaluación cuantitativa de la estabilidad de la arquitectura en los subsistemas de Cedrux. Con vista a solucionar el problema anterior se define como **objetivo general,** desarrollar un

[3] Proceso de adaptar un software a la capacidad de sus usuarios.

plugin que aporte datos precisos acerca de la estabilidad para apoyar la toma de decisiones arquitectónicas durante el proceso de verticalización.

Con el propósito de desglosar el objetivo general en metas concretas, se trazaron los siguientes **objetivos específicos:**

- ✓ Establecer un marco teórico relativo al proceso de evaluación de la arquitectura de software para seleccionar una técnica de evaluación que posibilite evaluar la estabilidad en la arquitectura de los subsistemas de Cedrux.
- ✓ Diseñar un plugin que permita aplicar la técnica de evaluación seleccionada a la arquitectura de los subsistemas de Cedrux.
- ✓ Implementar el diseño del plugin para obtener el grado de estabilidad de la arquitectura en los subsistemas de Cedrux.
- ✓ Validar la solución mediante pruebas de caja blanca y caja negra, además del criterio de expertos.

Se plantea como **idea a defender:**

Si se desarrolla un plugin que aporte datos precisos sobre la estabilidad de los subsistemas de Cedrux se podrá apoyar la toma de decisiones durante el proceso de verticalización.

Métodos científicos

Con el fin de apoyar la presente investigación se utilizan métodos teóricos y empíricos según el paradigma establecido en (Hernández León, 2002). Como parte de los métodos teóricos se utiliza el método Histórico-Lógico para conocer los antecedentes del proceso de evaluación de la estabilidad arquitectónica y el Analítico-Sintético para sintetizar lo estudiado mediante los análisis correspondientes.

Como métodos empíricos se utilizó la Entrevista en pos de conocer las particularidades de la arquitectura de los subsistemas de Cedrux a través de preguntas realizadas a los arquitectos de los mismos y el método Encuesta para obtener criterios acerca de la propuesta de solución.

Estructura del Trabajo

Este documento se encuentra estructurado en 3 capítulos tal y como se describe a continuación:

En el CAPÍTULO 1 se abordan los principales conceptos relacionados con la evaluación de la arquitectura de software, la estabilidad y el proceso de evolución o mantenimiento del software. Se realiza un estudio del estado del arte enfocado a la evaluación cuantitativa de la estabilidad.

Por su parte en el CAPÍTULO 2 se definen elementos importantes para lograr el entendimiento de la propuesta y se procede al diseño e implementación del plugin.

Finalmente en el CAPÍTULO 3 se realiza la validación del diseño propuesto utilizando las técnicas correspondientes, se validan las métricas seleccionadas y la obtención de datos precisos acerca de la estabilidad. Se realizan pruebas de caja blanca y caja negra para evaluar la efectividad del código y la puesta en marcha de las funcionalidades, respectivamente.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se detallan los elementos fundamentales que soportan la propuesta de solución en aras de establecer un dominio teórico. Se relacionan los principales conceptos asociados a la arquitectura, la evaluación de esta y las extrapolaciones que del concepto de estabilidad para el mundo de la ingeniería de software realizan varios autores; así como las corrientes que rigen el proceso de evaluación de esta última.

1.2 Arquitectura de software

Si de construcción se trata, una buena base o cimiento que soporte el resto de la infraestructura resulta de gran importancia. La construcción de software no está exenta de este planteamiento y posee como base principal, capaz de soportar el resto de la construcción, a la Arquitectura de Software, (en lo adelante AS). Esta importante subdisciplina de la Ingeniería de Software puede generalizarse como la división prudente de un todo en partes, estableciendo relaciones específicas entre dichas partes para facilitar el entendimiento de éstas por parte de todos los involucrados en el proceso de desarrollo.

Destacados autores del ámbito de la Ingeniería de Software establecen sus definiciones acerca de la arquitectura, tal es el caso de (Ivar Jacobson, 1999) donde se plantea : *una arquitectura es el conjunto de decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y las interfaces por las que se compone el sistema, junto con su comportamiento tal como se especifica en las colaboraciones entre dichos elementos, la composición de estas estructuras y elementos de comportamiento en subsistemas progresivamente grandes, y el estilo arquitectónico que guía esta organización, los elementos, sus interfaces, sus colaboraciones y su composición.*

La IEEE en su estándar 1471-2000 (IEEE, 2000) define la AS como: *la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.*

Por su parte Kazman y sus colegas en (Rick Kazman, 2003) expresan , *la arquitectura del software de un programa o sistema de cómputo es la estructura o las estructuras del sistema que incluyen los componentes del software, las propiedades visibles de esos componentes y las relaciones entre ellos.*

A decir de Pressman en (Pressman, 2006) la arquitectura no es el software operativo. En cambio, es una representación que permite al ingeniero de software, analizar la efectividad del diseño para cumplir con los requisitos establecidos, considerar opciones arquitectónicas en una etapa en que aún resulta relativamente fácil hacer cambios al diseño y de esa forma reducir los riesgos asociados con la construcción del software.

En aseveración del autor de este trabajo, las definiciones vistas anteriormente demuestran que existe una tendencia a considerar la AS, como la base organizacional de un sistema que viene determinada principalmente por los componentes como elementos básicos de la funcionalidad del sistema, caracterizados por una interfaz segmentada en puertos y conectores quienes describen la interacción entre los diferentes componentes para permitir un bajo acoplamiento entre ellos. En esta descripción, se emplea también una serie de conexiones que definen la composición de todos ellos formando una estructura única denominada configuración.

Basado en el estudio bibliográfico realizado y en los principios arquitectónicos a asumir en la presente investigación, se decide adoptar la definición de AS propuesta por el estándar 1471-2000 de la IEEE como base conceptual de cada uno de los elementos arquitectónicos que se presentan en la investigación.

Varios autores han realizado valoraciones acerca de la relevancia de la arquitectura. Kazman y sus colegas en su libro *Software Architecture in Practice*, *second edition*, (Rick Kazman, 2003) identifican tres razones claves que revelan la importancia de la arquitectura, estas se enuncian a continuación:

- Las representaciones de la arquitectura del software permiten la comunicación entre todas las partes interesadas en el desarrollo de un sistema informático.
- La arquitectura destaca las decisiones iniciales relacionadas con el diseño que tendrán un impacto profundo en todo el trabajo de la ingeniería del software que le sigue, y lo que también resulta importante, en el éxito final del sistema.

- La arquitectura constituye un modelo relativamente pequeño e intelectualmente comprensible de cómo está estructurado el sistema y cómo trabajan juntos sus componentes.

Tales razones evidencian que decidir la arquitectura ideal para desarrollar un producto de software cualquiera, resulta imprescindible. El establecimiento de un diseño arquitectónico que responda fielmente a los requisitos funcionales y no funcionales, y que además realice el correcto uso de los recursos existentes y los componentes que puedan ser reutilizables, incide directamente en la calidad del producto final.

Sin embargo, la arquitectura por sí sola, no es capaz de lograr cualidades, sino que sienta las bases para el logro de la calidad, pero este propósito será en vano si no se presta atención a los detalles proporcionados por los atributos de calidad. El logro de estos debe ser considerado en todo el diseño, la implementación y el despliegue. Ningún atributo de calidad es totalmente dependiente del diseño, ni es totalmente dependiente de la aplicación o implementación. Los resultados satisfactorios son una cuestión de decidir una arquitectura adecuada, que responda a las exigencias de estos atributos en el producto (Rick Kazman, 2003). En el epígrafe 1.3 se aborda acerca de estos atributos y su influencia en la calidad del producto final.

1.3 Atributos de calidad

Algunos autores definen el término calidad de software como *el grado en el cual el software posee una combinación deseada de atributos* (Mario Barbacci, 2005). *Tales atributos son requerimientos adicionales del sistema* (Rick Kazman, 2007) que hacen referencia a características que éste debe satisfacer, diferentes a los requerimientos funcionales. Estas características o atributos se conocen con el nombre de atributos de calidad, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios (Mario Barbacci, 2005). Estos atributos o requisitos implícitos que debe cumplir un sistema constituyen también criterios de medida de la calidad del software.

1.4 Estabilidad

Según el Diccionario de la Real Academia Española en su XXI edición, estabilidad es *permanencia, duración en el tiempo; firmeza, seguridad en el espacio, y estable es constante, firme, permanente*. A

continuación se presentan las extrapolaciones que de esta definición realizan varios autores en el ámbito de la ingeniería de software.

Martin expresa que *la estabilidad se puede definir como algo que no es fácilmente cambiable. Claramente los módulos que son más difíciles de cambiar, serán los menos volátiles, por lo tanto, cuanto más difícil de cambiar sea un módulo, más estable es y menos volátil será* (Martin, 1997).

Por su parte Jazayeri define la estabilidad como *la capacidad que posee la arquitectura de soportar los cambios evolutivos y permanecer con su estabilidad intacta, por tal motivo la denomina estabilidad arquitectónica* (Jazayeri, 2002).

La norma ISO/IEC 9126 define la estabilidad como *un subatributo de la mantenibilidad que expresa: la capacidad del producto de software para minimizar los efectos inesperados que puedan ocurrir cuando se realizan modificaciones* (ISO/IEC, 2003).

Bahsoon y Emmerich afirman que *la estabilidad tiene como objetivo expresar el grado en que la arquitectura de un sistema dado es capaz de evolucionar, mientras esta y sus decisiones de diseño no cambien* (Rami Bahsoon, 2006).

1.4.1 Consideraciones generales de los conceptos relacionados con la estabilidad

Resulta apreciable al analizar los conceptos emitidos a priori que todos se encuentran fuertemente relacionados con la arquitectura definida para el software, pues de ella depende todo el proceso de desarrollo. Tal y como se afirma en la norma ISO/IEC 9126 y como indican los demás conceptos enunciados, la estabilidad es una capacidad que posee el software de mantenerse firme ante el necesario proceso de evolución. Martin, Jazayeri y Bahsoon adoptan la posición de que la arquitectura adquiere la mayor responsabilidad debido a que según sea el grado de flexibilidad de esta para soportar las modificaciones sin que ocurran efectos inesperados, se puede garantizar una buena evolución para el software. El nivel de aparejamiento que existe entre la estabilidad y la arquitectura, revela la importancia de la evaluación de esta última para conocer en qué medida satisface la estabilidad, acentuando en este sentido la utilización de una forma de evaluación que muestre resultados precisos acerca del nivel de estabilidad que posee la arquitectura del sistema. Teniendo en cuenta la similitud de las definiciones

existentes a cerca de la estabilidad y el prestigio que posee la norma ISO/IEC 9126, se toma la definición de estabilidad emitida por esta, como concepto guía de la presente investigación.

1.5 Evaluación de Arquitectura de Software

Todo producto de software se realiza con la intención de satisfacer las necesidades y expectativas de los clientes, por lo que velar por la calidad de dicho producto durante todo el ciclo de vida es una actividad primordial. Tales razones engendran la entrada de una serie de procesos para que el producto final cumpla con las normativas de calidad adoptadas por la entidad. El proceso de evaluación de arquitectura influye de manera especial en la obtención de una correcta funcionalidad del producto, la evaluación es realizada teniendo en cuenta los requisitos del cliente, que incluyen la funcionalidad del sistema y sus atributos de calidad.

Al evaluar una arquitectura se busca medir propiedades del sistema sobre la base de especificaciones abstractas, para favorecer los requisitos, restricciones y atributos de calidad en busca de la seguridad de su cumplimiento por parte del sistema. Otro aspecto importante de la evaluación se centra en la necesidad de analizar e identificar los riesgos que puedan presentarse en la estructura diseñada. Evaluar una arquitectura permite:

- ✓ Analizar y evaluar la calidad exigida por los usuarios.
- ✓ Tomar decisiones de diseño.
- ✓ Identificar restricciones de implementación.
- ✓ Fijar la estructura organizacional del desarrollo, construcción y ejecución del sistema.
- ✓ Satisfacer los atributos de calidad.
- ✓ Realizar estimaciones más certeras.

1.5.1 Especificaciones de la arquitectura que se desea evaluar

Resulta necesario mostrar una descripción del contexto en el que se aplicará la solución especificando los elementos que se tienen en cuenta dentro de este. En este caso, se exponen de manera breve algunos aspectos de la arquitectura de Cedrux con el fin de esclarecer el significado que tienen para el presente trabajo.

La arquitectura de Cedrux define siete vistas arquitectónicas. El presente trabajo solo se enfoca en las vistas de integración y la de componentes. Estas brindan elementos como:

La vista de integración, como se expone en (Fernández, 2011), se encarga de los procesos de integración a nivel de sistema. Esta integración puede ser interna (entre componentes de un mismo subsistema) y externa (entre distintos subsistemas).

La vista de componentes, que a decir de (Fernández, 2011), se encarga de las definiciones de la taxonomía de componentes. Es decir, se conceptualizan los tipos de componentes, se especifican sus características, además de la composición interna de cada uno de ellos. Estas especificaciones acerca de los tipos de componentes definidos en Cedrux se pueden encontrar en (Arias, 2013).

La interacción entre dichos componentes se realiza mediante un conector. El conector definido se conoce como IoC. Este se utiliza de dos formas (Martínez, 2010):

1. IoC-externo: Este se usa para la interacción entre componentes que no están incluidos en su nivel jerárquico dentro de un mismo componente.

2. IoC-interno: Este se usa para la interacción entre componentes que descienden de un mismo componente en algún nivel de la jerarquía a la que pertenecen.

La arquitectura que se estudia, está determinada por el tipo de sistema, en este caso un Sistema Integral de Gestión, que incorpora el estilo arquitectónico Llamada y Retorno, combinando dos patrones arquitectónicos pertenecientes a este, los cuales son: Basado en Componentes y Modelo Vista Controlador. Los mayores beneficios de este tipo de sistemas se encuentran en la integración de todos los procesos importantes de una empresa. Tales beneficios evidencian la necesidad de que el sistema se desarrolle con integración continua, pues de no ser así una fase de integración podría demorarse demasiado tiempo. Por otra parte, no se podría asegurar que en una fase de integración, por el elevado número de componentes y la alta relación entre ellos se puedan detectar y resolver todos los errores que aparezcan (Martínez, 2010).

Para la integración entre subsistemas se usa como mediador un fichero XML nombrado IoC (Inversión de Control), donde cada subsistema publica sus servicios. En el caso de la integración entre los componentes internos de un proyecto se usa algo similar solo que en ese caso el mediador es un XML nombrado IoC interno, cada subsistema posee uno para que sus componentes publiquen los servicios que le brindan al

resto de los componentes del proyecto. En el caso de la integración con otros sistemas se implementa una interfaz donde se ubican todos los servicios que brinda Cedrux (Martínez, 2010).

1.6 Técnicas de evaluación de arquitecturas de software

La evaluación de arquitectura constituye un proceso relevante y de estricto cumplimiento, el nivel de control y de detalle que este puede introducir, arrojará aportes significativos en la calidad del producto. Con tales fines existen numerosos métodos y técnicas para llevar a cabo una evaluación de arquitectura. Por la necesidad de obtener resultados cuantitativos acerca de una subcaracterística en específico esta investigación solo se centra en las técnicas para la evaluación de la arquitectura y dentro de estas, se hace énfasis en las técnicas cuantitativas. A continuación se presenta en la figura 2, la clasificación de técnicas de evaluación de arquitectura dada por (Alvarez, 2012).

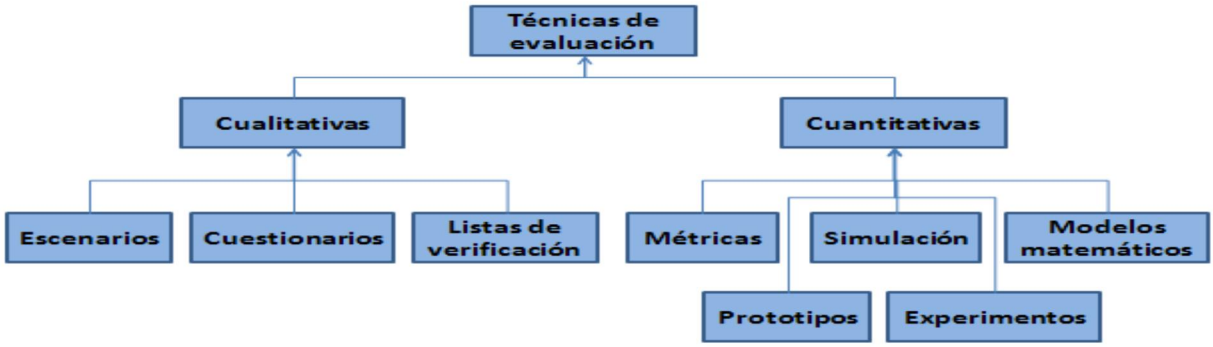

Figura 1: Técnicas de evaluación de arquitectura (Alvarez, 2012)

Las técnicas cuantitativas tienden a tener más aceptación en cuanto a la obtención de valores, sin embargo tienen en su contra la necesidad de herramientas que las soporten, porque en tiempo de producción, se hace muy engorroso ponerlas en práctica manualmente (Alvarez, 2012).

Evaluación basada en métricas

Este tipo de evaluación se enfoca en obtener resultados a partir de fórmulas que establecen medidas para distintos aspectos observables en el proceso de evaluación. Las métricas suelen definirse 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, permitiendo descubrir y corregir problemas potenciales antes de que se conviertan en efectos catastróficos. (Pressman, 2006)

La posibilidad que ofrecen estas de adaptarse al sistema en cuestión y de emitir un resultado cuantitativo que exprese el estado de lo que se ha medido y que ayude a la toma de decisiones, es precisamente la razón principal que las sitúa como una de las técnicas de evaluación cuantitativas más usadas en la actualidad.

1.7 Corrientes para la evaluación de la estabilidad

Aunque muy similares y con los mismos objetivos, los diferentes conceptos conocidos acerca de la estabilidad han generado varias formas para su evaluación. Según Jazayeri existen dos enfoques para la evaluación de la estabilidad: Predictivo y Retrospectivo (Jazayeri, 2002).

Ambos enfoques parten del supuesto de que el principal objetivo de la arquitectura de software es guiar la evolución del sistema. La evaluación retrospectiva analiza sucesivas versiones del sistema para analizar la suavidad con la que la evolución tuvo lugar, mientras que la evaluación predictiva proporciona información detallada sobre la evolución del sistema de software basado en el examen de un conjunto de cambios probables y el grado en que la arquitectura puede soportarlos.

En este sentido se proponen diferentes corrientes para medir el grado de estabilidad de una arquitectura definida. Una de ellas consiste en realizar medidas simples tales como: el tamaño de los módulos, la cantidad de módulos que fueron añadidos y la cantidad que fueron borrados en diferentes versiones. Otra de las corrientes se enfoca en medir el grado de acoplamiento existente entre los módulos del sistema (Jazayeri, 2002).

Es objetivo de esta investigación utilizar estas corrientes en busca de formalizar cómo se van a inferir los elementos que ellas proponen desde un modelo estático donde se encuentre representada la arquitectura, utilizando para ello, los enfoques Retrospectivo y Predictivo antes mencionados.

1.7.1 Métricas para la evaluación de la Estabilidad

El estudio de las distintas métricas concebidas para evaluar la estabilidad se enfoca en la búsqueda de elementos comunes aplicados a cada una de estas durante el proceso de medición. Con esta premisa se examinaron las métricas propuestas en los estudios analizados con anterioridad en el epígrafe dedicado a la **estabilidad**.

La ISO/IEC 9126 define la métrica **Impacto del Cambio,** la cual consiste en contar el número de impactos adversos detectados tras la modificación y compararlos con la cantidad de modificaciones realizadas. Para su medición se establece la fórmula (X=1-A/B) donde A se define como el número de impactos adversos detectados tras la modificación y B representa el número de modificaciones realizadas. (ISO/IEC, 2003)

Esta norma también propone medir la **Localización del impacto de la modificación**, donde se cuenta el número de variables afectadas por la modificación y se comparan con el total de variables del producto. Encaminado a su evaluación se define la fórmula (X=A/B), donde A representa el número de variables afectadas tras la modificación y B el total de variables del producto.

Robert Martin en (Martin, 1997) también hace un aporte dirigido al uso de las métricas para medir la estabilidad, el cual consiste en evaluar la estabilidad posicional de un paquete o componente de software a través de la utilización de los siguientes conceptos:

Acoplamientos Aferentes (Ca): Se define como el número de clases fuera del paquete que dependen de las clases contenidas en el paquete.

Acoplamientos Eferentes (Ce): Se define como el número de clases dentro del paquete que dependen de clases externas al paquete.

Inestabilidad (I): Esta métrica está comprendida en el intervalo [0, 1], de forma que I=0 indica un paquete completamente estable, y por el contrario I=1 indica un paquete completamente inestable. La inestabilidad se define mediante la siguiente relación:

$$I = \frac{Ce}{Ca + Ce}$$

El estándar IEEE 982.1-1988 sugiere un **Índice de Madurez del Software** (IMS) que proporciona una indicación de la estabilidad de un producto de software basada en los cambios que ocurren con cada versión del producto, (Pressman, 2006). Se determina la siguiente información:

Mr = número de módulos en la versión actual

Fc = número de módulos en la versión actual que se han cambiado

Fa = número de módulos en la versión actual que se han añadido

Fd = número de módulos de la versión anterior que se han borrado en la versión actual

El índice de madurez del software se calcula de la siguiente manera:

$$IMS = \frac{[Mr - (Fa + Fc + Fd)]}{Mr}$$

Tabla 1: Comparación entre las métricas de estabilidad según los indicadores estudiados

Métricas Vs Indicadores	Impacto del Cambio	Localización del impacto del cambio	Inestabilidad	Índice de madurez del software
Cantidad de módulos	-	-	-	✓
Cantidad de módulos cambiados	-	-	-	✓
Cantidad de				

módulos borrados	-	-	-	✓
Acoplamiento entre módulos	-	-	✓	-

Consideraciones generales de las métricas estudiadas

Resulta apreciable el hecho de que las métricas propuestas por la ISO/IEC 9126 se encuentran enfocadas a productos y resulta más difícil aplicarlas cuando se trata de un modelo estático de la arquitectura que contiene elementos y relaciones como las expuestas en los indicadores de la comparación. Por su parte, la métrica Inestabilidad propuesta por Martin, utiliza las métricas acoplamientos aferentes y eferentes, elementos relacionados con el acoplamiento y las dependencias entre los módulos, aspecto significativo a la hora de realizar cambios en la arquitectura debido a que brindan información acerca de las dependencias que existen entre las diferentes partes del modelo arquitectónico. Finalmente, la métrica Índice de Madurez del Software propuesta por la IEEE y empleada por Pressman en su libro Un enfoque Práctico, permite realizar una inferencia directa de los elementos del modelo para satisfacer tres de los indicadores propuestos y conocer a través de estos la compatibilidad existente entre los modelos arquitectónicos definidos en diferentes versiones del producto. Por tal motivo y siguiendo el principio de las dos corrientes expuestas por Jazayeri, se decide adaptar la métrica Inestabilidad al contexto de Cedrux, aprovechando que pueden inferirse a través de ella varios de los elementos presentes en un modelo arquitectónico. Además se decide utilizar la métrica Índice de Madurez del Software para obtener un coeficiente que exprese la suavidad con la que el cambio ha tenido lugar. Cabe destacar dentro de que enfoque entra cada métrica según sus características. En el caso de la métrica Índice de Madurez del Software, que realiza la evaluación tomando versiones sucesivas, se está en presencia del enfoque Retrospectivo. Entre tanto, la métrica Inestabilidad brinda la posibilidad de analizar el impacto de cambios referente al nivel de dependencia existente entre los elementos del modelo arquitectónico.

Con el propósito de adaptar la métrica Inestabilidad definida por Martin, al contexto de Cedrux, se decide tomar como acoplamientos aferentes a la cantidad de servicios que ofrece el componente y como acoplamientos eferentes a la cantidad de servicios que consume el componente. De esta forma se mantendría igual la fórmula de Inestabilidad (I=Ce/ (Ca+Ce)), comprendida en el intervalo [0, 1], de forma

que I=0 indica un componente completamente estable, y por el contrario I=1 indica un componente completamente inestable. La correspondencia de acoplamientos aferentes y eferentes con los servicios de los componentes será validado por el criterio de expertos.

1.9 Metodología de Desarrollo de software

Todo proceso de desarrollo de software necesita de un hilo conductor que guíe al equipo de desarrollo durante todo el ciclo de vida hacia la obtención de resultados satisfactorios. Con este fin y ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental en el momento de desarrollar un producto de software, surgen las metodologías de desarrollo (Pérez, 2008). La fuente mencionada clasifica las metodologías en:

Metodologías robustas: centradas especialmente en el control del proceso, establecen rigurosamente las actividades involucradas, los artefactos que se deben producir, las herramientas y notaciones que se usarán.

Metodologías ágiles: orientadas a la interacción con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Son efectivas en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

Teniendo en cuenta la clasificación de las metodologías expuesta con anterioridad, las características del sistema que se desea desarrollar y el corto tiempo para su implementación, se decide utilizar como guía para el proceso de desarrollo una metodología ágil.

Actualmente existen varias metodologías ágiles capaces de adaptarse a las características del sistema que se desea desarrollar, entre estas se destacan eXtreme Programming (XP), SCRUM Y Proceso Unificado Ágil (OpenUP), por su fácil y rápida implementación. Debido a que cualquiera de estas tres metodologías puede ser utilizada en el desarrollo de la solución propuesta, se examinan cada una de estas con el propósito de escoger la que cuente con los aspectos más significativos

Extreme Programming

Comúnmente, un proyecto desarrollado con la metodología XP considera entre 10 y 15 ciclos o iteraciones aplicando los principios de simplicidad, comunicación e interacción permanente con el cliente en virtud de comprobar constantemente los requisitos, así como la aplicación de la técnica de programación por parejas, en la cual uno de los programadores escribe código y el otro lo prueba, invirtiendo los papeles con posterioridad (Regalado, 2009).

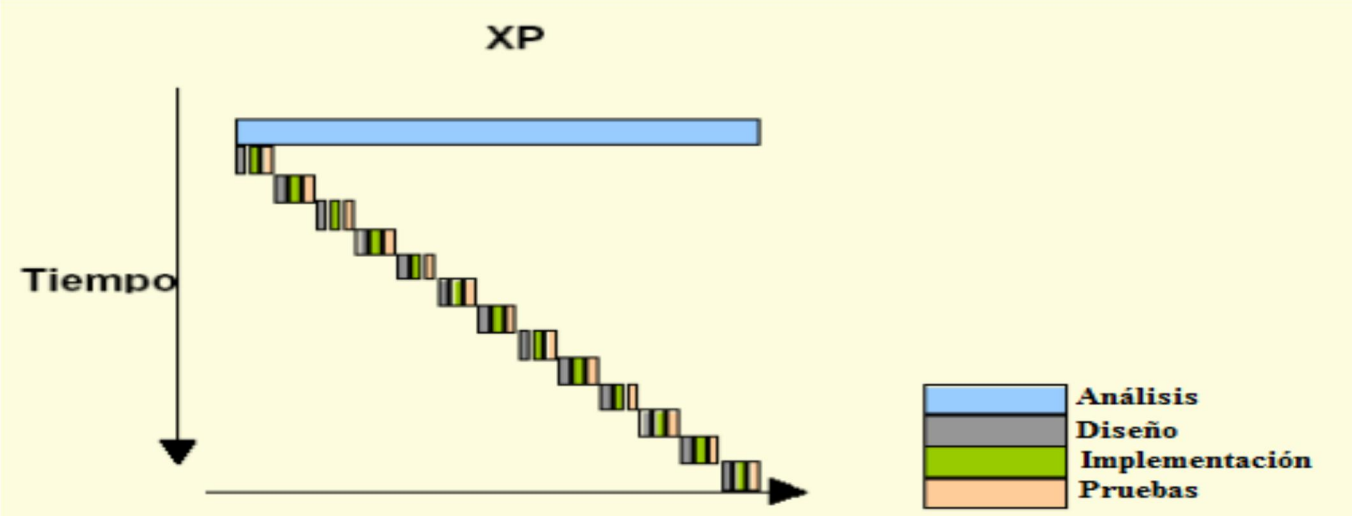

Figura 2: Metodología XP

Esta metodología se compone de una serie de valores, los cuales deben ser cumplidos por el equipo de desarrollo para alcanzar el producto deseado por el cliente (Pérez, 2012). Según la fuente citada estos valores son: simplicidad, comunicación, retroalimentación y coraje. Estos valores son aplicados durante todo el ciclo de vida del producto a desarrollar, es decir durante la ejecución de cada una de sus cuatro fases, lo cual permite:

- La introducción de nuevos requisitos o cambiar los anteriores ágilmente.
- Adecuación a proyectos pequeños y medianos.
- Adecuación a proyectos con alto riesgo.
- Ciclo de vida iterativo e incremental.
- La duración de cada iteración estará entre una y tres semanas.

SCRUM

En Scrum un proyecto se ejecuta en bloques temporales (iteraciones o sprints) de un mes natural (pueden ser de dos o tres semanas, si así se necesita). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto que sea susceptible de ser entregado con el mínimo esfuerzo cuando el cliente lo solicite (Pérez, 2012).

Figura 3: Metodología SCRUM

El Sprint es el ritmo de los ciclos de Scrum. Está delimitado por la reunión de planificación del sprint y la reunión retrospectiva. Una vez que se fija la duración del sprint es inamovible. La mayoría de los equipos eligen dos, tres o cuatro semanas de duración. Diariamente durante el sprint, el equipo realiza una reunión de seguimiento muy breve. Al final del sprint se entrega el producto al cliente en el que se incluye un incremento de la funcionalidad que tenía al inicio del sprint (Pérez, 2012).

El proceso parte de la lista de requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente ha priorizado los requisitos balanceando el valor que le aportan respecto a su costo y han sido divididos en iteraciones y entregas, comprendiendo la realización de siete actividades centradas en la planificación, las iteraciones diarias y la retroalimentación (Pérez, 2012).

OpenUP

OpenUP constituye una versión optimizada de RUP y al igual que esta posee una aproximación iterativa incremental dentro de un ciclo de vida estructurado. Cuenta además con una filosofía ágil enfocada en la naturaleza colaborativa del desarrollo de software. Está concebida para proyectos pequeños, se

caracteriza por ser una metodología basada en casos de usos, centrada en la arquitectura, iterativo e incremental. Contiene fundamentalmente un conjunto de simplificaciones de roles, actividades, artefactos y guías (Eclipse, 2012).

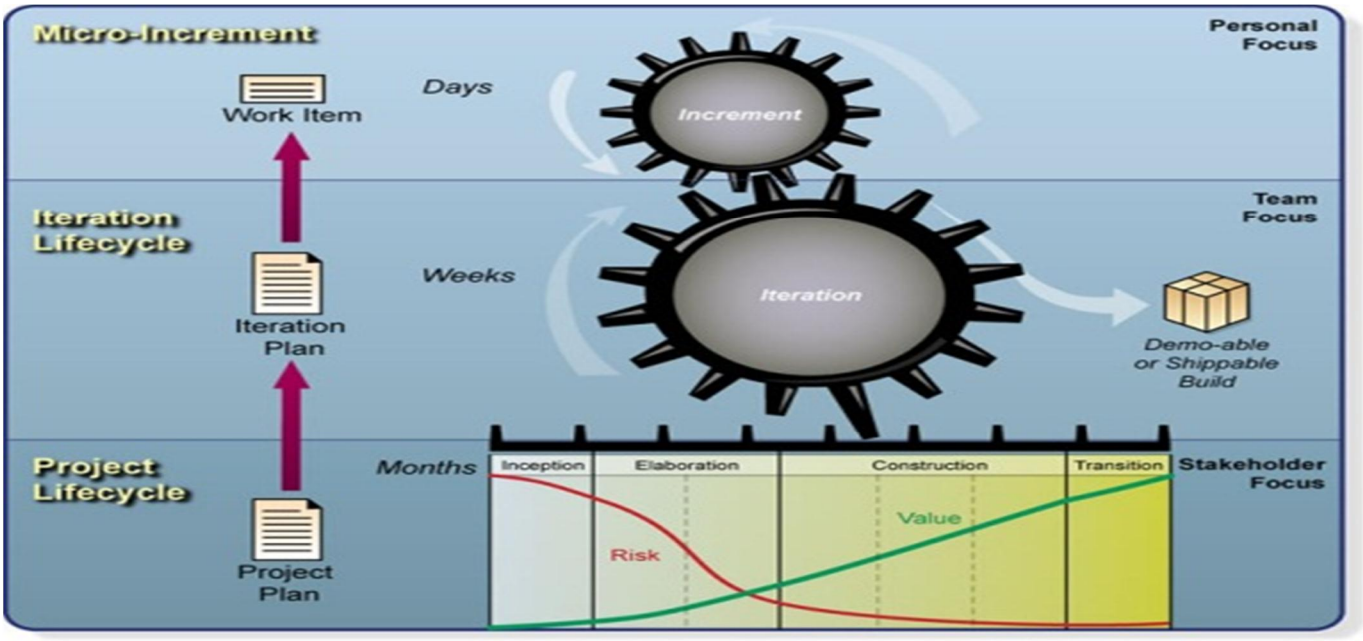

Figura 4: Metodología OpenUP

Esta metodología divide el proyecto en iteraciones: planeadas en intervalos de tiempos fijos usualmente medido en semanas. Las iteraciones concentran al equipo en entregas incrementales de valor para los involucrados en una manera predecible. Esto se realiza definiendo bien engranadas y finas tareas de una lista de elementos de trabajo (Eclipse, 2012).

A continuación se muestran algunas de sus características.

- Permite detectar errores tempranos a través de un ciclo iterativo.
- Es apropiado para proyectos pequeños y de bajos recursos.
- Permite disminuir las probabilidades de fracaso en los proyectos pequeños e incrementar las probabilidades de éxito.
- Evita la elaboración de documentación, diagramas e iteraciones innecesarios requeridos en la metodología RUP.

Consideraciones generales de las metodologías estudiadas

Una vez conocidos los aspectos significativos de las tres metodologías en cuestión, se procede a una comparación formal de cada una de estas para seleccionar la que guiará el proceso de desarrollo del plugin.

Tabla 2. Comparación de las metodologías estudiadas

Indicadores	XP	SCRUM	OpenUP
Principios	Retroalimentación Simplicidad Cambios incrementales Aceptar el cambio Aceptar la responsabilidad	Reconocimiento Entregas rápidas Aproximación pragmática	Simplicidad Agilidad Centra actividades de alto valor Adaptación del producto para satisfacer las necesidades
Fases	Exploración Planificación Iteraciones por entregas Producción Mantenimiento Muerte	Planificación del Sprint Seguimiento del Sprint Sprint Revisión de sprint	Inicio Elaboración Construcción Transición
Actividades	Codificar Hacer pruebas Escuchar Diseñar	Planificación de proyecto Selección de requisitos Planificación de la iteración	Modelo Implementación Despliegue Prueba

		Ejecución de la iteración Demostración y Retrospectiva	
Artefactos	Historia de usuarios Diagrama de clases	Lista de requisitos priorizada Lista de tareas de la iteración Incremento	Especificación de requisitos Diagrama de casos de uso Diagrama de diseño Diseño de casos de prueba

Luego de analizar estas metodologías ágiles se decide utilizar la metodología OpenUP en el desarrollo de la solución propuesta, los siguientes elementos apoyan tal decisión.

- Define exactamente los principios y fases por las que debe regirse el proceso de desarrollo.
- Cumple con todas las actividades necesarias a realizar por parte del equipo de desarrollo.
- Genera los artefactos que permiten obtener la documentación necesaria para una mejor comprensión del plugin a desarrollar.

1.10 Lenguajes y Herramientas

Tras definir la propuesta de solución y de escoger a la metodología de desarrollo ágil OpenUP para guiar el proceso de desarrollo, se presentan a continuación los lenguajes y herramientas que se utilizarán en la solución.

Lenguajes de descripción arquitectónica

Los lenguajes de descripción de arquitectura, en lo adelante ADL, remontan su surgimiento a la década del 90 del siglo pasado, poco tiempo después de haberse institucionalizado la arquitectura de software como disciplina profesional. Un ADL en su definición más simple es una entidad consistente en cuatro “Cs”: componentes, conectores, configuraciones y restricciones, del inglés (constraints) (Wolf, 1997).

Otra definición importante de ADL es la que lo exhibe como un lenguaje descriptivo de modelado que se focaliza en la estructura de alto nivel de la aplicación antes que en los detalles de implementación de sus módulos concretos (Vestal, 1993).

De manera general los ADL suministran construcciones para especificar abstracciones arquitectónicas y mecanismos para descomponer un sistema en componentes y conectores, especificando de qué manera estos elementos se combinan para formar configuraciones y definiendo familias de arquitecturas o estilos. Contando con un ADL, un arquitecto puede razonar sobre las propiedades del sistema con precisión, pero a un nivel de abstracción convenientemente genérico. Algunas de esas propiedades podrían ser, por ejemplo, protocolos de interacción, anchos de banda y latencia, localización del almacenamiento, conformidad con estándares arquitectónicos y previsiones de evolución ulterior del sistema (Reynoso, 2004).

En opinión del autor del presente trabajo, la mayoría de las definiciones tienen como sentido común la representación de las cuatro “Cs”, aunque algunos autores agregan otros elementos como son las propiedades, los estilos y los paradigmas. A continuación se describen los conceptos principales que forman parte de un ADL.

Los componentes representan los elementos computacionales primarios de un sistema. Intuitivamente, corresponden a las cajas de las descripciones de caja-y-línea de las arquitecturas de software. Ejemplos típicos serían clientes, servidores, filtros, objetos, pizarras y bases de datos. En la mayoría de los ADL los componentes pueden exponer varias interfaces, las cuales definen puntos de interacción entre un componente y su entorno.

Por su parte los conectores representan interacciones entre componentes. Corresponden a las líneas de las descripciones de caja-y-línea. Ejemplos típicos podrían ser tuberías (pipes), llamadas a procedimientos, broadcast de eventos, protocolos cliente-servidor, o conexiones entre una aplicación y un servidor de base de datos. Los conectores también tienen una especie de interfaz que define los roles entre los componentes participantes en la interacción (Reynoso, 2004).

Las configuraciones se constituyen como grafos de componentes y conectores. En los ADL más avanzados la topología del sistema se define independientemente de los componentes y conectores que

lo conforman. Los sistemas también pueden ser jerárquicos: componentes y conectores pueden subsumir la representación de lo que en realidad son complejos subsistemas (Reynoso, 2004).

Finalmente las restricciones representan condiciones de diseño que deben acatarse incluso en el caso que el sistema evolucione en el tiempo. Restricciones típicas serían restricciones en los valores posibles de propiedades o en las configuraciones topológicas admisibles. Por ejemplo, el número de clientes que se puede conectar simultáneamente a un servicio (Reynoso, 2004).

Acme

Acme es uno de los ADL más usados en la actualidad, este se define como un lenguaje capaz de soportar el mapeo de especificaciones arquitectónicas entre diferentes ADL, o en otras palabras, como un lenguaje de intercambio de arquitectura. Además define siete elementos básicos que son: Componentes, conectores, sistemas, puertos, roles, representaciones y mapas de representación (Reynoso, 2004).
Acme soporta una variedad de front-ends de carácter gráfico, el más difundido de ellos es Acme Studio que es un entorno gráfico basado en Windows y Linux, susceptible de ser configurado para soportar visualizaciones específicas de estilos e invocación de herramientas auxiliares.
Tales ventajas han condicionado su utilización en el centro CEIGE y hacen de este lenguaje el candidato principal para llevar a cabo el modelado de la arquitectura.

Lenguaje de Modelado Unificado

El Lenguaje Unificado de Modelado (UML) es un lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos de un sistema de software intensivo. El UML le da una manera estándar de escribir los planos de un sistema, que abarca aspectos conceptuales, como las empresas procesos y funciones del sistema, así como las cosas concretas, como las clases escritas en un determinado lenguaje de programación, esquemas de bases de datos y componentes de software reutilizables. El lenguaje UML es aplicable a cualquier persona involucrada en la producción, la implementación y el mantenimiento de software (Grady Booch, 2004).

Lenguajes de programación

Un lenguaje de programación es un lenguaje artificial que intenta parecerse al lenguaje humano, utilizado para definir una secuencia de instrucciones capaces de ser interpretadas y ejecutadas por una computadora. Este lenguaje establece un grupo de reglas sintácticas y semánticas, las cuales rigen la estructura del programa de computación que se escribe.

Java

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

Acme Studio

Acme Studio se define como una herramienta capaz de soportar el mapeo de especificaciones arquitectónicas entre diferentes ADL, o en otras palabras, como un lenguaje de intercambio de arquitectura. Soporta la definición de cuatro tipos de arquitectura: la estructura (organización de un sistema en sus partes constituyentes); las propiedades de interés (información que permite razonar sobre el comportamiento local o global, tanto funcional como no funcional); las restricciones (lineamientos sobre la posibilidad del cambio en el tiempo); los tipos y estilos. La estructura se define utilizando siete tipos de entidades: componentes, conectores, sistemas, puertos, roles, representaciones y rep-mapas (mapas de representación) (Reynoso, 2004).

La disponibilidad de la herramienta gráfica AcmeStudio para el modelado tiene suficiente documentación y páginas web. Hay documentación y herramientas disponibles libremente en Internet. La herramienta ha sido desarrollada en la plataforma Eclipse en el lenguaje de programación Java, facilitando la extensión mediante plugins.

Visual Paradigm

En la actualidad existen varias herramientas CASE capaces de soportar todo el ciclo de desarrollo del plugin del presente trabajo. Sin embargo, se decide utilizar la herramienta Visual Paradigm, por ser una herramienta multiplataforma. Además, a través de sus diagramas ayuda a una rápida construcción de aplicaciones de calidad con un menor costo. Permite realizar ingeniería directa e inversa, es decir, obtener el código a partir de diagramas, así como diagramas a partir de un código previamente escrito. Los diagramas a realizar con la ayuda del Visual Paradigm serán los de: diagramas de casos de uso del sistema y diagrama de clases del diseño.

Entorno de Desarrollo Integrado

Un Entorno de Desarrollo Integrado (IDE) es una aplicación informática compuesta por un conjunto de herramientas de programación, encargadas de facilitar el trabajo de los desarrolladores de software a la hora de llevar a cabo una aplicación. Permite dedicarse a un único lenguaje de programación o hacer uso de varios de ellos.

Eclipse

Entre los diferentes IDE existentes en la actualidad se decide utilizar Eclipse, por la premisa de que AcmeStudio es un front-end perteneciente a este IDE. Además de contar con sus grandes ventajas para la creación de plugins de manera sencilla. Permite escribir, compilar, depurar y ejecutar programas escritos en Java, aunque puede servir para otros lenguajes de programación. Es un proyecto de código abierto y gratuito, con una gran base de usuarios y documentación disponible. A continuación se enuncian algunas de sus ventajas:

- Es un IDE con muchas distribuciones y soporta una gran cantidad de lenguajes.
- Es una herramienta de código abierto. Corre en una gran cantidad de sistemas operativos incluyendo Windows y Linux.
- Provee la utilidad de comenzar el programa con los plugins especificados, permitiendo acceder a distintas aplicaciones sin necesidad de levantar todas a la vez al momento de ejecutarlo.

- La plataforma Eclipse está construida en base a plugins. Este mecanismo permite desarrollar, integrar y correr nuevos plugins.

Plugins y Puntos de Extensión

Un plugin es un módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande. Puede verse además como la unidad más atómica y extensible de Eclipse que puede ser distribuida de manera separada. La plataforma Eclipse consiste aproximadamente, en 100 plugins trabajando juntos. A los límites entre ellos, que permiten conectar un plugin a otro, se le denomina: puntos de extensión. Los puntos de extensión son el mecanismo por el cual un plugin puede adicionarle funcionalidades a otro. Estos consisten en código Java empaquetado en un archivo de Java (JAR), con algunos archivos de solo lectura, y otros recursos como imágenes, catálogo de mensajes, entre otros (Clayberg, 2008).

Cada plugin posee un manifiesto del plugin, en el cual se describe su interconexión con otros plugins, o sea, se declara un número determinado de puntos de extensión y a su vez, se exponen las extensiones que se han realizado de otros puntos de extensión definidos por otros plugins. El manifiesto del plugin está representado por un par de ficheros: el MANIFEST.MF y el plugin.xml (Clayberg, 2008).

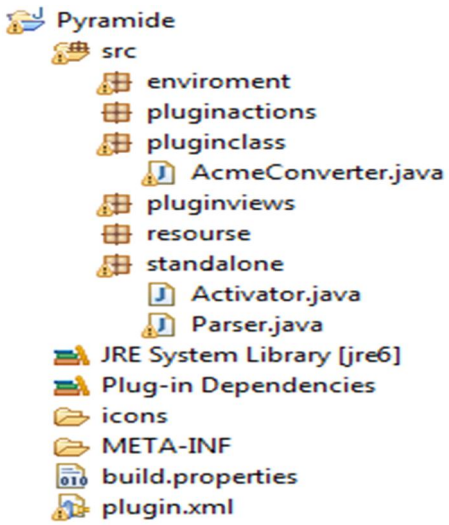

Figura 5: Estructura simple de un plugin

1.11 Conclusiones

El desarrollo de este capítulo condujo a las siguientes conclusiones:

- El estudio relacionado con la arquitectura de software y el proceso de evolución demostró la estrecha relación existente entre ambos y la necesidad de que la arquitectura responda de forma estable a la realización de cambios.
- La Estabilidad es un atributo importante para el desarrollo del proceso de evolución en un Sistema Integral de Gestión y un elemento significativo durante la realización de la integración de componentes.
- Predecir el grado en que una arquitectura cumple con determinado atributo, es muy factible y brinda resultados precisos si se utiliza la técnica de evaluación basada en métricas.
- La herramienta Acme Studio ofrece la posibilidad de extender funcionalidades, su importancia radica en que pueden implementarse funciones para evaluar los atributos de calidad desde la perspectiva de un modelo arquitectónico estático.
- OpenUP es una metodología ágil que ofrece un hilo conductor durante todo el ciclo de vida del software. Genera artefactos que proveen un correcto desarrollo sin obviar elementos significativos y muestran con creces el estado del proceso de desarrollo. Tales aspectos la sitúan como la metodología de desarrollo ágil idónea para desarrollar la solución.

CAPÍTULO 2: Diseño e Implementación

2.1 Introducción

El presente capítulo expone el diseño del plugin basado en los fundamentos abordados previamente. El plugin se denomina Pyramide debido a que este tipo de construcción es una de las más estables. El presente capítulo expone la estructura y características del plugin Pyramide, así como su conjunto de funcionalidades. En aras de lograr mayor documentación y comprensión del sistema se procede a la utilización de la metodología ágil OpenUP y los artefactos que esta genera.

2.2. Propuesta del sistema

El plugin Pyramide tiene como objetivo informatizar el proceso de evaluación de la Estabilidad en los subsistemas de Cedrux, haciendo uso de las facilidades que para este tipo de evaluación ofrece el ADL Acme. El sistema tiene como entrada el modelo arquitectónico de uno o varios subsistemas generados en la herramienta AcmeStudio, para luego aplicar las métricas definidas con anterioridad y emitir un resultado cuantitativo que apoye la toma de decisiones.

2.3. Productos de trabajo propuestos por OpenUP

La metodología OpenUP orienta los pasos a seguir durante el proceso de desarrollo y propone la generación de una serie de productos de trabajo durante el ciclo de vida de un software, a continuación se exponen los más relevantes en aras de lograr una documentación que respalde al proceso de desarrollo.

2.3.1. Visión Pyramide

Dentro de los artefactos que plantea generar OpenUP se encuentra el documento visión. Este artefacto contiene la definición de la mirada de los involucrados del producto a desarrollar, especificado en términos de las necesidades y características claves de los involucrados. Proporciona una visión completa del sistema de software en desarrollo y los soportes del contrato entre clientes y desarrolladores.

Tabla 3. Visión Pyramide y declaración del problema

El problema de	¿Cómo obtener datos precisos sobre la estabilidad de los subsistemas de Cedrux para apoyar la toma de decisiones durante el proceso de verticalización?
Afecta a	Arquitectos de las líneas de Cedrux.
Cuyo impacto es	Carencia de datos precisos para realizar una mejor toma de decisiones.
Una solución exitosa sería	Confeccionar un plugin para eclipse que realice el proceso de evaluación de la estabilidad usando como entrada el modelo arquitectónico de uno o varios subsistemas confeccionados en la herramienta Acme Studio para establecer intervalos acerca del grado de estabilidad que estos poseen y definir un catálogo de decisiones arquitectónicas con respecto a dichos intervalos.

2.3.2. Descripción de los requisitos del software

La ingeniería de requisitos proporciona un mecanismo apropiado para entender lo que el cliente quiere, analizar necesidades, evaluar factibilidad, negociar una solución razonable, especificar la solución sin ambigüedades, validar la especificación y administrar los requisitos conforme estos se transforman en un sistema operacional (Pressman, 2006).

La captura de requisitos tiene como objetivo obtener una descripción detallada y precisa de lo que el sistema debe realizar para satisfacer a los clientes. Al definir los requisitos se hace alusión al alcance que tendrá el sistema en términos de qué y cómo, o capacidades y cualidades. A la hora de definirlos se clasifican en dos grupos: funcionales y no funcionales (Guide, 2007).

Existen varias técnicas para la elicitación de requisitos; las entrevistas, el desarrollo conjunto de aplicaciones, la tormenta de ideas, la utilización de escenarios y la revisión documental. Para capturar los requisitos que debe cumplir Pyramide, se utilizaron las técnicas: **entrevistas**, **tormenta de ideas** con arquitectos y líderes de algunas líneas de Cedrux y se realizó una **revisión documental**. Esta última se realizó con el objetivo de revisar las métricas existentes para calcular la estabilidad. De este estudio se

derivó la posibilidad de adaptar la métrica Inestabilidad, para utilizar como acoplamientos aferentes y eferentes a los servicios que ofrece y consume determinado componente.

2.3.3. Requisitos Funcionales

Los requerimientos funcionales son los que definen las funciones que el sistema será capaz de realizar, describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. Es importante que se describa el ¿Qué? y no el ¿Cómo? se deben hacer esas transformaciones. Estos requerimientos al tiempo que avanza el proyecto de software se convierten en los algoritmos, la lógica y gran parte del código del sistema (Chávez, 2006).

Los requisitos funcionales están referidos a las funciones que debe cumplir el sistema y pueden verse como capacidades o condiciones que el sistema debe poseer. Acto seguido se enumeran los requisitos funcionales para la confección del plugin Pyramide.

Tabla 4. Requisitos Funcionales de Pyramide

RF1. Seleccionar modelo	RF3. Aplicar métrica IMS
RF2. Aplicar métrica Inestabilidad	RF4. Emitir reporte detallado

Descripción de los requisitos funcionales

Seleccionar modelo: El sistema permitirá la selección de los modelos a evaluar.

Aplicar métrica inestabilidad: El sistema devuelve el grado de estabilidad del modelo seleccionado.

Aplicar métrica IMS: El sistema devuelve el grado de estabilidad de los modelos seleccionados, mostrando datos del proceso de comparación.

Emitir reporte detallado: El sistema permitirá guardar en formato PDF los resultados de las métricas aplicadas de forma detallada y ofrecerá como valor añadido un catálogo de decisiones arquitectónicas.

Validación de requisitos

Durante la actividad de elicitación de requisitos puede ocurrir que algunos de estos no estén lo suficientemente claros o que no se esté muy seguro de haber entendido correctamente los requisitos obtenidos hasta el momento, tal situación puede llevar a un desarrollo que no cumpla con el objetivo final del sistema, por lo que resulta imprescindible validar los requisitos capturados. La validación es la etapa final de la Ingeniería de Requisitos. Su objetivo es, ratificar los requisitos, es decir, verificar todos los requisitos que aparecen en el documento especificado para asegurarse que representan una descripción, por lo menos, aceptable del sistema que se debe implementar. Esto implica verificar que los requerimientos sean consistentes y que estén completos (Chávez, 2006). Para validar los requisitos definidos en el presente trabajo se construyeron prototipos de interfaz y se produjo una revisión técnica formal de los mismos.

Los prototipos son simulaciones del posible producto, que luego son utilizados por el usuario final, ofreciendo una importante retroalimentación en cuanto a si el sistema diseñado con base a los requisitos recolectados le permite al usuario realizar su trabajo de manera eficiente (Chávez, 2006).

El desarrollo del prototipo comienza con la captura de requisitos. Desarrolladores y clientes se reúnen y definen los objetivos globales del software, identifican todos los requisitos que son conocidos, y señalan áreas en las que será necesaria la profundización en las definiciones. Luego de esto, tiene lugar un “diseño rápido”. El diseño rápido se centra en una representación de aquellos aspectos del software que serán visibles al usuario (por ejemplo, entradas y formatos de las salidas). El diseño rápido lleva a la construcción de un prototipo (Chávez, 2006).

A continuación se presentan los prototipos de interfaz referentes a cada requisito definido:

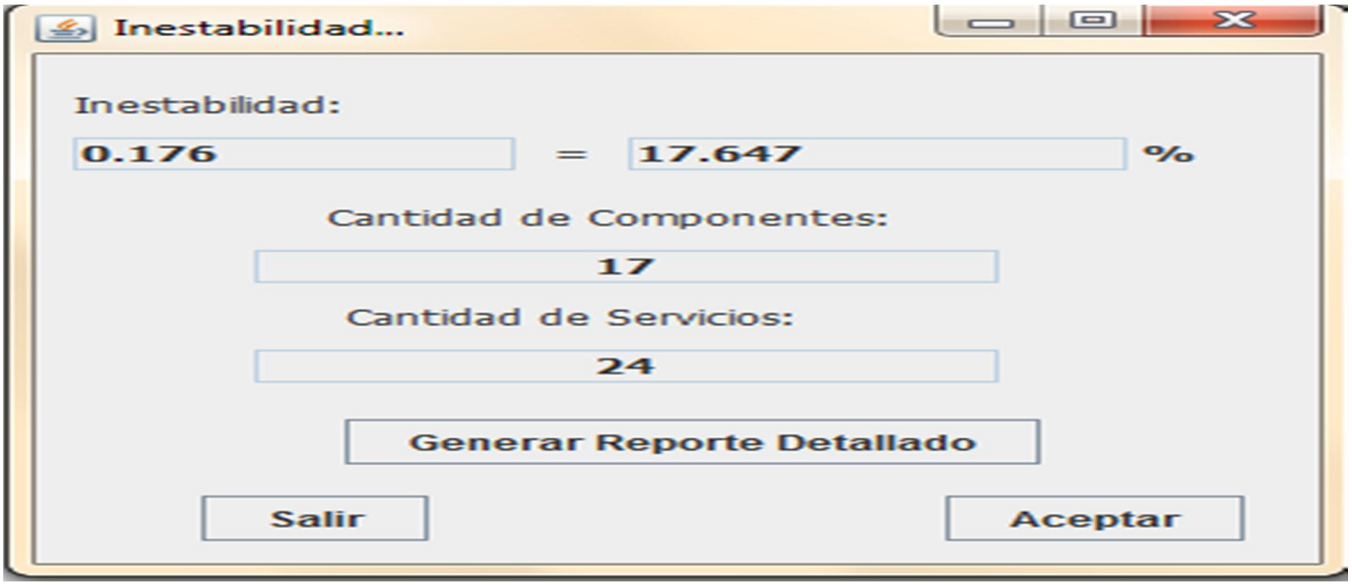

Figura 6: Prototipo de interfaz del requisito Aplicar Métrica Inestabilidad

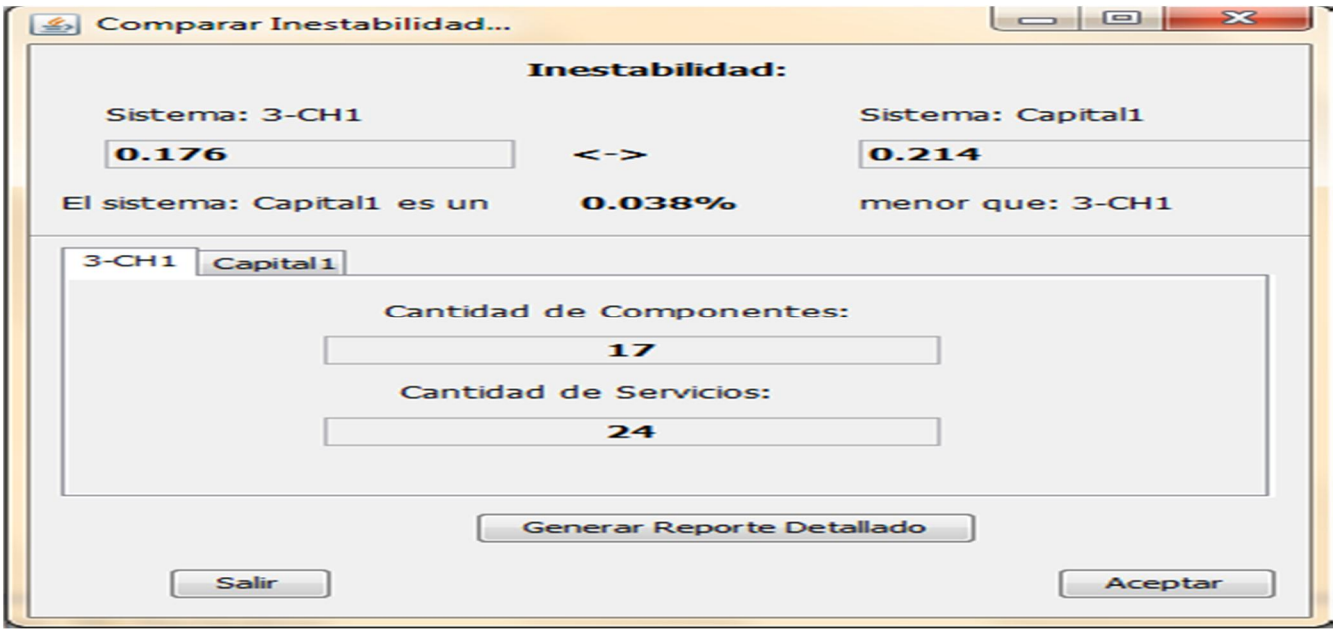

Figura 7: Prototipo de interfaz del requisito Aplicar Métrica Inestabilidad (comparación)

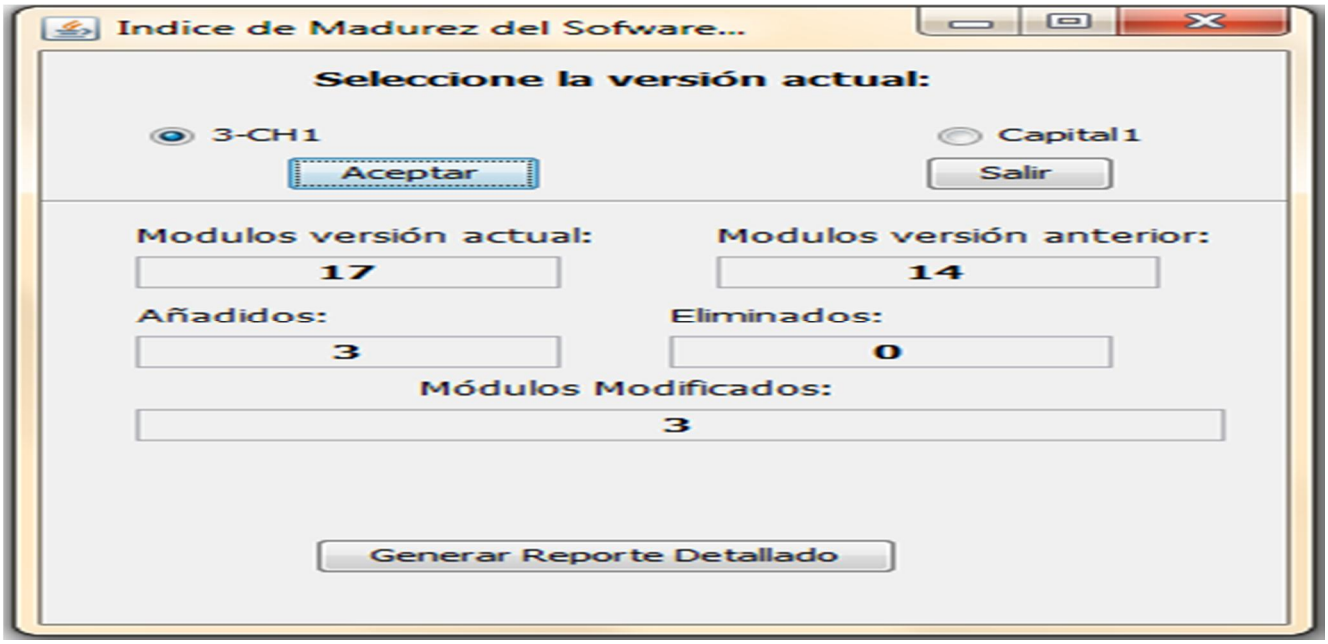

Figura 8: Prototipo de interfaz del requisito Aplicar Métrica IMS

En las revisiones técnicas formales el equipo de desarrollo debe conducir al cliente a través de los requisitos del sistema, explicándole las implicaciones de cada requisito. El equipo de revisión debe verificar la consistencia y completitud de cada requisito (Sommerville, 2005) . En el caso específico de los requisitos descritos para la construcción del plugin Pyramide se realizó una revisión técnica formal con el cliente, donde se le explicó todo lo referente a cada requisito, comprobándose la consistencia y completitud de estos. Para validar el requisito Aplicar Métrica Inestabilidad se realizó una encuesta, donde los expertos en el tema emitieron sus criterios.

2.3.4. Requisitos no Funcionales

Los requisitos no funcionales tienen que ver con características que de una u otra forma puedan limitar el sistema, como por ejemplo, el rendimiento (en tiempo y espacio), interfaces de usuario, fiabilidad (robustez del sistema, disponibilidad de equipo), mantenimiento, seguridad, portabilidad, estándares, etc.

Los requisitos no funcionales están enfocados a elementos que el sistema debe tener en cuenta a la hora de su confección, expresan cualidades o propiedades necesarias para lograr un mejor funcionamiento y

aceptación por parte del cliente. A continuación se describen los requisitos no funcionales a tener en cuenta en el desarrollo del plugin Pyramide.

Requisitos de Funcionalidad

- El sistema permitirá generar reportes en formato PDF.

Requisitos de Confiabilidad

- El sistema no permitirá la entrada de datos incorrectos.

Requisitos de Usabilidad

- El idioma de todas las interfaces de la aplicación será el español.
- El sistema solo permitirá habilitar las opciones de acuerdo al número de modelos seleccionados.

Requisitos de rendimiento

- El sistema no excederá los 3 segundos de tiempo de respuesta al cargar un proyecto.
- El sistema no excederá los 3 segundos de tiempo de respuesta a la hora de obtener el grado de estabilidad de cada componente y del sistema en general.

Requisitos de Software

- JDK 1.5 o superior.
- Eclipse IDE 3.3 o superior.
- Acme Studio 3.5.3 o superior.

2.3.5. Diagrama de casos de uso del sistema y diagrama de paquetes

Los casos de uso representan procesos o funcionalidades que se realizan en el sistema, constituyen una descripción de cómo actúa el sistema. En la metodología OpenUP este artefacto presenta una visión general del comportamiento previsto del sistema. Es la base para un acuerdo entre las partes interesadas

y el equipo del proyecto en lo que respecta a la funcionalidad prevista del sistema. También guía a las distintas tareas del ciclo de vida de desarrollo de software.

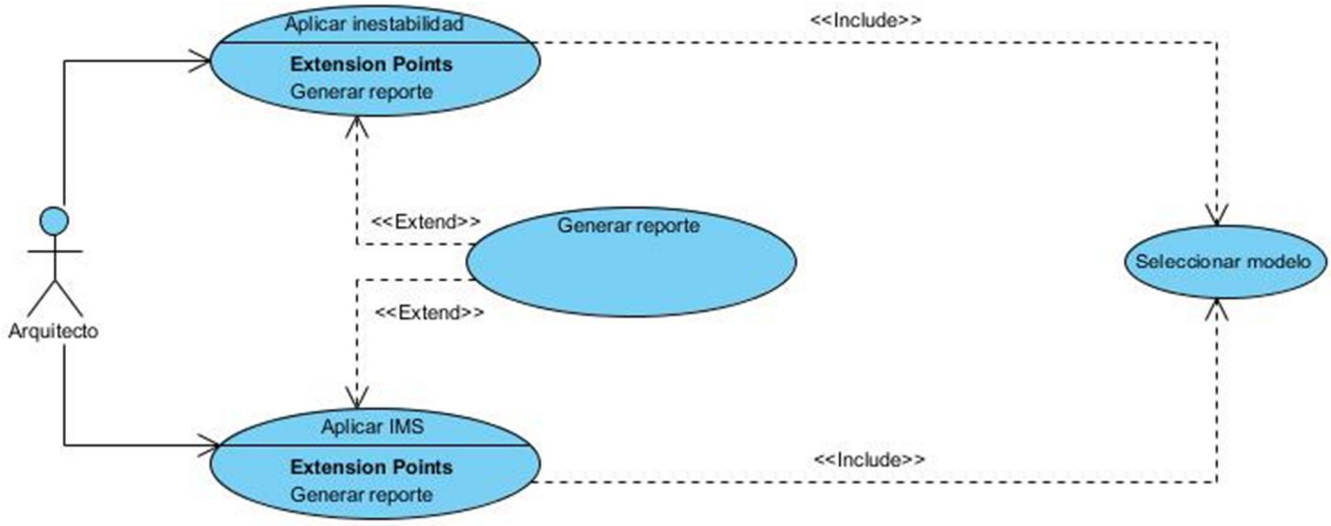

Figura 9: Diagrama de casos de uso del sistema

Actores:

- **Arquitecto:** Es el encargado de realizar la evaluación de la estabilidad y generar los reportes.

Casos de uso:

- **Seleccionar modelo:** Describe la acción de seleccionar el o los modelos que desean ser evaluados o comparados.
- **Aplicar inestabilidad:** Describe el proceso de aplicación de la métrica inestabilidad.
- **Aplicar IMS:** Describe el proceso de aplicación de la métrica índice de madurez del software.
- **Generar reporte:** Describe la acción de generar y buscar el directorio donde será guardado el reporte.

Diagrama de paquetes

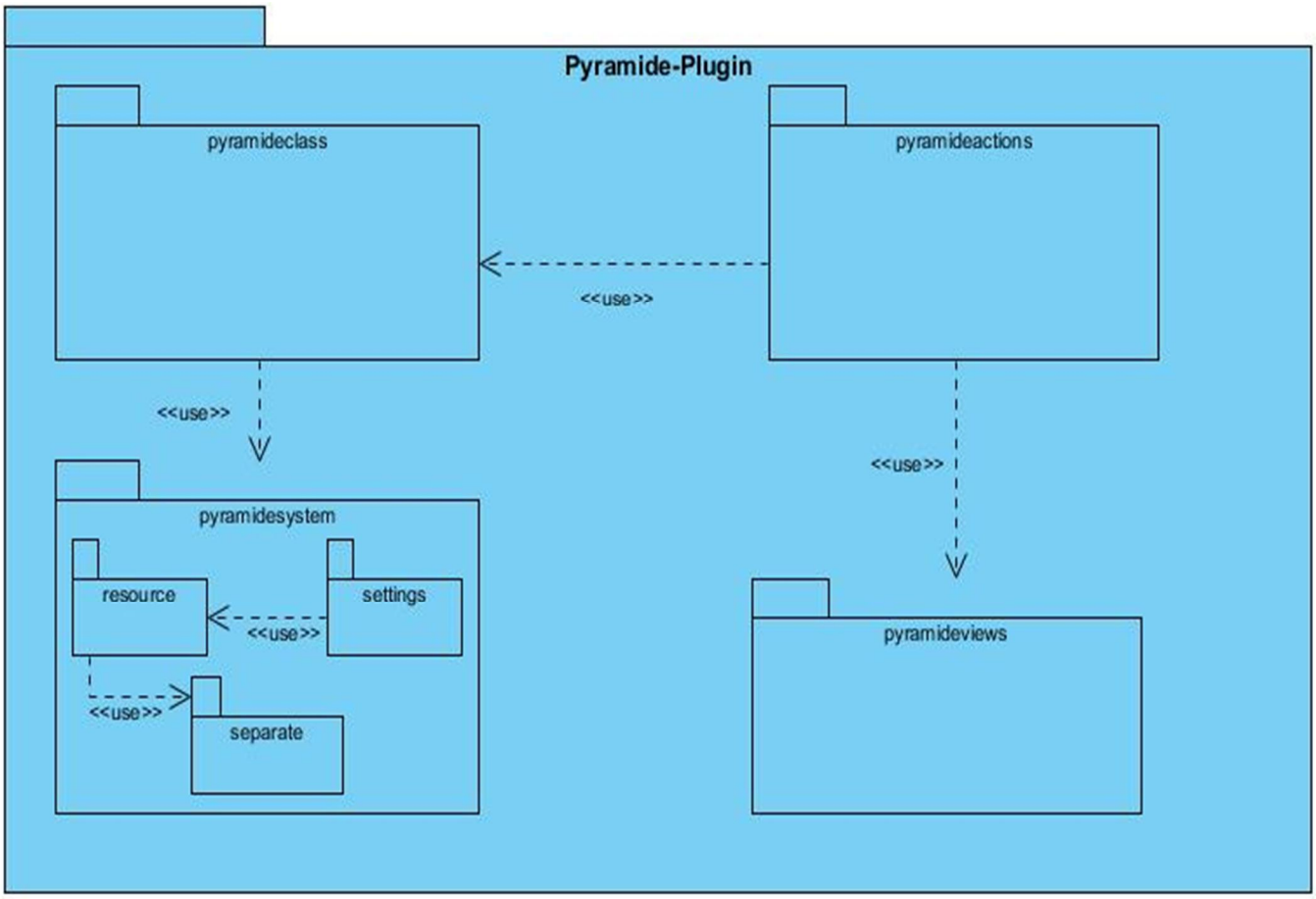

Figura 10: Diagrama de paquetes

Descripción de los paquetes

- **Pyramideclass:** Contiene las clases StabilityClass, IMSClass y CompareClass, que son las encargadas de implementar los métodos correspondientes para evaluar a través de las métricas Inestabilidad e IMS.
- **Pyramidesystem:** Agrupa 3 paquetes internos, estos son: pyramidesystem.resource, que contiene los recursos necesarios para manejar el código obtenido; pyramidesystem.separate que contiene las clases Activator y Parser, las cuales facilitan el trabajo con los objetos acme. El tercer paquete se denomina pyramidesystem.settings y contiene las clases que permiten manejar errores.

- **Pyramideactions:** Contiene las clases que fungen como controladoras, estas son: StabilityAction, IMSAction y CompareAction.
- **Pyramideviews:** Contiene las clases que fungen como interfaces del plugin.

2.3.6. Diagrama de clases del diseño

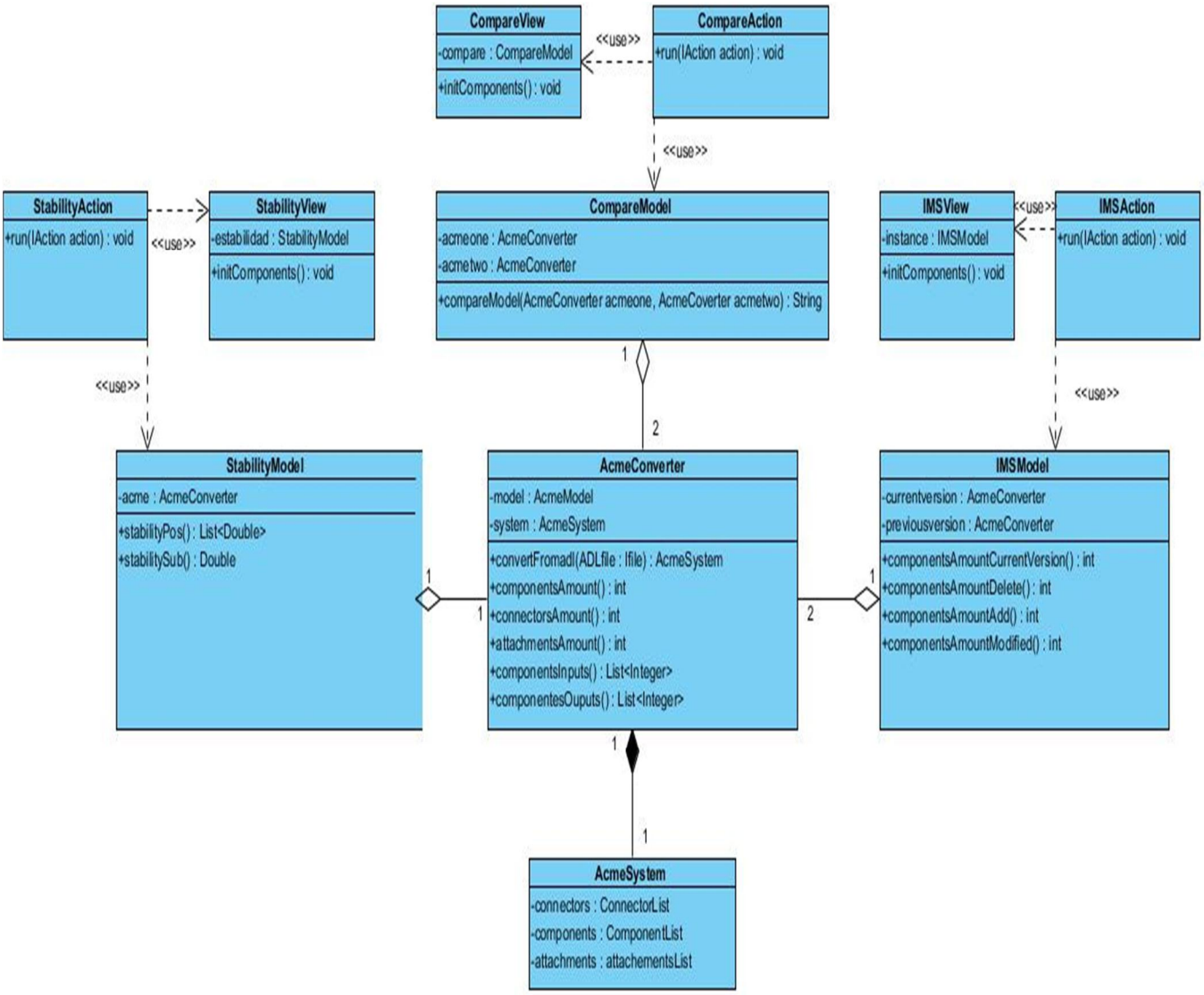

Figura 11: Diagrama de clases del diseño

2.3.7. Descripción de las clases del diseño

La siguiente tabla muestra una breve descripción de cada una de las clases diseñadas.

Tabla 5.Descripción de las clases del diseño

Elementos representados	Descripción
CompareModel	Posee la funcionalidad encargada de realizar la comparación entre modelos utilizando la métrica inestabilidad.
StabilityModel	Clase encargada de aplicar la métrica inestabilidad para un solo modelo.
AcmeConverter	Clase contenedora de todos los elementos Acme.
IMSModel	Es la clase encargada de aplicar las funcionalidades referentes al cálculo de la métrica IMS.
AcmeSystem	Clase compuesta por todos los elementos Acme. Es la clase que se genera a partir de lo que se ha modelado.
StabilityView	Contiene y maneja todos los elementos a mostrar en la interfaz.
CompareView	Contiene y maneja todos los elementos a mostrar en la interfaz.
IMSView	Contiene y maneja todos los elementos a mostrar en la interfaz.
StabilityAction	Funge como clase controladora de las clases StabilityModel y StabilityView.
CompareAction	Funge como clase controladora de las clases CompareModel y CompareView.

IMSAction	Funge como clase controladora de las clases IMSModel e IMSView.

2.4. Patrones de diseño utilizados

Patrones de asignación de responsabilidades

Este tipo de patrones se enfoca en la asignación de responsabilidades a los objetos en sentido general, encargándose de conocer atributos y relaciones con otros objetos. A continuación se evidencia cómo fueron utilizados algunos de estos patrones en el diseño de la solución.

Creador: Guía la asignación de responsabilidades relacionadas con la creación de objetos posibilitando la asignación al responsable de la creación de una nueva instancia de alguna clase. En el diseño mostrado previamente, este patrón es utilizado en la clase *AcmeConverter* que es la responsable de manipular a través de la clase *AcmeSystem* cada uno de los elementos Acme necesarios para la solución.

Alta cohesión: Existe afinidad entre cada clase y los métodos que implementan, estas poseen responsabilidades vinculadas acordes a la información que controlan. Su utilización mejora la claridad y facilidad con que se entiende el diseño, simplifica el mantenimiento y las mejoras de funcionalidad, generan bajo acoplamiento, soporta mayor capacidad de reutilización. Este patrón se pone de manifiesto en la mayoría de las clases porque cada una es capaz de realizar sus responsabilidades sin la utilización de las demás.

Bajo acoplamiento: Consiste en tener un diseño de clases lo menos ligadas posibles. De tal forma que en caso de producirse una modificación en alguna de ellas, tenga la mínima repercusión en el resto de las clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases. Este patrón se evidencia en todo el diagrama debido a la escasa relación que existe entre las clases diseñadas.

Experto: Se puso en práctica en las clases del paquete *pyramideclass*, con el uso de clases que poseen responsabilidades específicas a cumplir de acuerdo con la información que manejan, el paquete cuenta con clases que poseen funciones concretas de acuerdo con la información que gestiona.

Controlador: Se utilizó en las clases del paquete *pyramideactions* con el objetivo de controlar los eventos correspondientes a las interfaces de cada una de las clases del paquete *pyramideclass*.

2.5. Métricas de validación del diseño

El IEEE Standard Glossary of Software Engineering Terms define métrica como una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado (Pressman, 2006).

Las métricas del software permiten medir de forma cuantitativa la calidad de los atributos internos del producto, esto permite al ingeniero evaluar la calidad durante el desarrollo del sistema. Son varios los puntos de vista relacionados con la calidad del software. Las métricas de diseño a nivel de componentes se concentran en las características internas de los componentes del software con medidas que pueden ayudar al desarrollador a juzgar la calidad de un diseño a nivel de componente (Pressman, 2006).

Un aspecto importante a tener en cuenta en la evaluación del diseño, es la creación de métricas básicas inspiradas en el estudio de la calidad del diseño orientado a objetos. Los atributos de calidad que se tienen en cuenta son: (Pressman, 2006).

- **Responsabilidad:** Es la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- **Complejidad de implementación:** Es la complejidad de implementación que posee una estructura de diseño de clases.
- **Reutilización:** Es el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento:** Es el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.
- **Complejidad del mantenimiento:** Es el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
- **Cantidad de pruebas:** Es el número o el grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (componente, módulo, clase, conjunto de clases, etc.) diseñado.

Las métricas seleccionadas como instrumento para evaluar la calidad del diseño del plugin son las siguientes:

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

Tabla 6. Atributos de calidad evaluados por la métrica TOC

Atributo de calidad	Modo en que lo afecta
Responsabilidad	Aumento del TOC provoca aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Aumento del TOC provoca aumento de la complejidad de implementación de la clase.
Reutilización	Aumento del TOC provoca disminución del grado de reutilización de la clase.

Se definieron los siguientes criterios y categorías de evaluación para los atributos de calidad anteriores:

Tabla 7. Criterios de evaluación para la métrica TOC

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

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

Tabla 8. Atributos de calidad evaluados por la métrica TOC

Atributo de calidad	Modo en que lo afecta

Acoplamiento	Aumento del RC provoca aumento del Acoplamiento de la clase.
Complejidad de mantenimiento	Aumento del RC provoca aumento de la complejidad del mantenimiento de la clase.
Reutilización	Aumento del RC provoca disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Aumento del RC provoca aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 9. Atributos de calidad evaluados por la métrica TOC

Atributo	Categoría	Criterio
Acoplamiento	Baja	<= Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Complejidad de mantenimiento	Baja	<= Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Cantidad de pruebas	Baja	<= Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Reutilización	Baja	<= Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio

En el presente epígrafe se exponen los procedimientos empleados para la validación del diseño propuesto. Se presentan las métricas Tamaño operacional de clase (TOC) y Relaciones entre clases (RC), para evaluar la calidad del diseño propuesto.

Resultados obtenidos de la aplicación de la métrica TOC

Al aplicar la métrica TOC se concluyó que Pyramide cuenta con 11 clases y un total de 19 procedimientos, con un promedio de 1,72 procedimientos por clase. El resultado demuestra que la mayoría de las clases presentan un bajo nivel de responsabilidad, complejidad y una alta reutilización siendo este diseño lo más simple posible, permitiendo una sencilla implementación y una amplia realización de pruebas con mayor facilidad. Las figuras 12,13 y 14 muestran los resultados arrojados por la medición de los atributos de calidad propuestos por esta métrica.

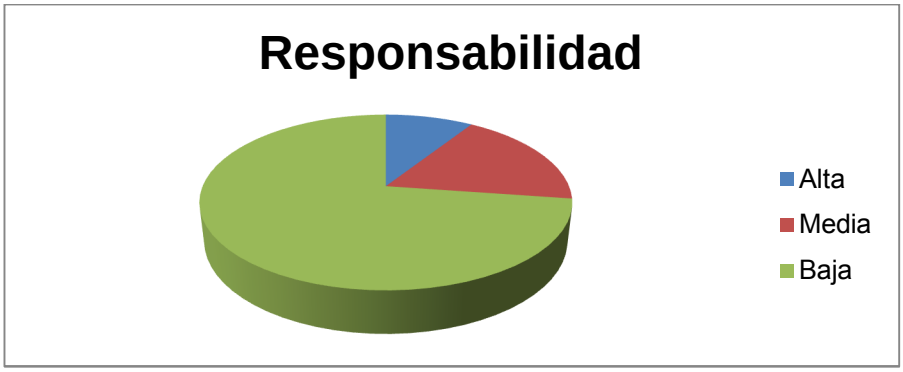

Figura 12: Representación en % de la incidencia de los resultados obtenidos en el atributo Responsabilidad

Figura 13: Representación en % de la incidencia de los resultados obtenidos en el atributo Complejidad de Implementación

Figura 14: Representación en % de la incidencia de los resultados obtenidos en el atributo Reutilización

Resultados de la Métrica RC: Relaciones entre clases

Para aplicar la métrica RC se determinó que la aplicación cuenta con 11 clases y un total de 10 dependencias entre ellas, con un promedio de 0,9 dependencias por clase. Luego de aplicarse dicha métrica de diseño y obtenidos los resultados de la evaluación del instrumento de medición de la misma, se puede concluir que el diseño propuesto tiene una calidad aceptable teniendo en cuenta que el 63,63% de las clases empleadas poseen menos de dos dependencias de otras clases, lo que lleva a evaluaciones positivas del atributo de calidad: acoplamiento. Los atributos de calidad (complejidad de mantenimiento y cantidad de pruebas) arrojan valores positivos, con un 36,36% de baja complejidad de mantenimiento y cantidad de pruebas. Para el atributo de calidad reutilización se concluye que de manera general el diseño cuenta con un 36,36% de alta reutilización. Estos resultados favorecen la reutilización de las clases así como la modificación e implantación del diseño. Como resultado de la evaluación de esta métrica se obtuvieron los resultados que muestran las figuras 15, 16, 17 y 18.

Figura 15: Representación en % de la incidencia de los resultados obtenidos en el atributo Acoplamiento

Figura 16: Representación en % de la incidencia de los resultados obtenidos en el atributo Complejidad de Mantenimiento

Figura 17: Representación en % de la incidencia de los resultados obtenidos en el atributo Reutilización.

Figura 18: Representación en % de la incidencia de los resultados obtenidos en el atributo Cantidad de pruebas

Al aplicar las métricas a las clases del diseño definidas se cumple que las métricas de Tamaño Operacional de la clase y Relaciones entre clases cumplen las restricciones para la aceptación de las clases diseñadas.

2.6. Descripción de implementación por funcionalidades

Para el desarrollo del plugin se utilizó Eclipse que provee un asistente para crear el esqueleto de lo que va a ser el plugin. Este IDE provee un plugin asociado denominado PDE (Plugin Development Enviroment) que no es más que el ambiente de desarrollo de plugin que propone la Fundación de Eclipse y constituye uno de los principales subproyectos de la misma. A continuación se detallan las funcionalidades que ofrece Pyramide.

Seleccionar modelo: Esta funcionalidad es la encargada de la selección de uno o más modelos para la evaluación, dicha selección está dada por la métrica que se desea aplicar o bien para establecer una comparación entre modelos.

Aplicar métrica IMS: Es la funcionalidad encargada de medir cada uno de los indicadores propuestos por la métrica Índice de Madurez del Software, para obtener como resultado el grado de estabilidad de cada modelo.

Aplicar métrica Inestabilidad: Se encarga de medir los indicadores propuestos por la métrica Inestabilidad para obtener el coeficiente de estabilidad del subsistema.

Comparar modelos: Es la funcionalidad encargada de realizar una comparación entre modelos utilizando la métrica Inestabilidad para decidir cuál de estos es más estable.

Emitir reporte detallado de Inestabilidad: Consiste en la emisión de un reporte que contenga toda la información correspondiente a la métrica que se esté aplicando. Se emite en formato PDF y dentro contiene la fecha de creación, el coeficiente de estabilidad del subsistema, la información referente a la comparación entre estos y finalmente según el coeficiente de estabilidad que tenga el o los modelos, se emite un catálogo de decisiones arquitectónicas comprobadas, que pueden ser tomadas según el grado de estabilidad que posea el subsistema.

Emitir reporte detallado de Índice de Madurez del Software: Consiste en la emisión de un reporte que contenga toda la información correspondiente a la métrica que se está aplicando. Se emite en formato PDF y dentro contiene la fecha de creación, el coeficiente de estabilidad de cada subsistema, así como la información referente a la cantidad de módulos que contiene cada uno, la cantidad que fueron añadidos, modificados y eliminados de una versión hacia otra.

La figura 19 muestra el flujo correspondiente a cada una de las funcionalidades comenzando por la selección del modelo, en dependencia de cuantos modelos sean seleccionados (1 ó 2), entonces se activarán las funcionalidades. Para el caso particular donde se seleccione un solo modelo, se activará la funcionalidad Calcular Inestabilidad y se brindará la posibilidad de generar un reporte detallado con información acerca de la evaluación. Si se seleccionan dos modelos entonces se podrán aplicar las funcionalidades IMS y realizar una comparación entre modelos aplicando la métrica Inestabilidad.

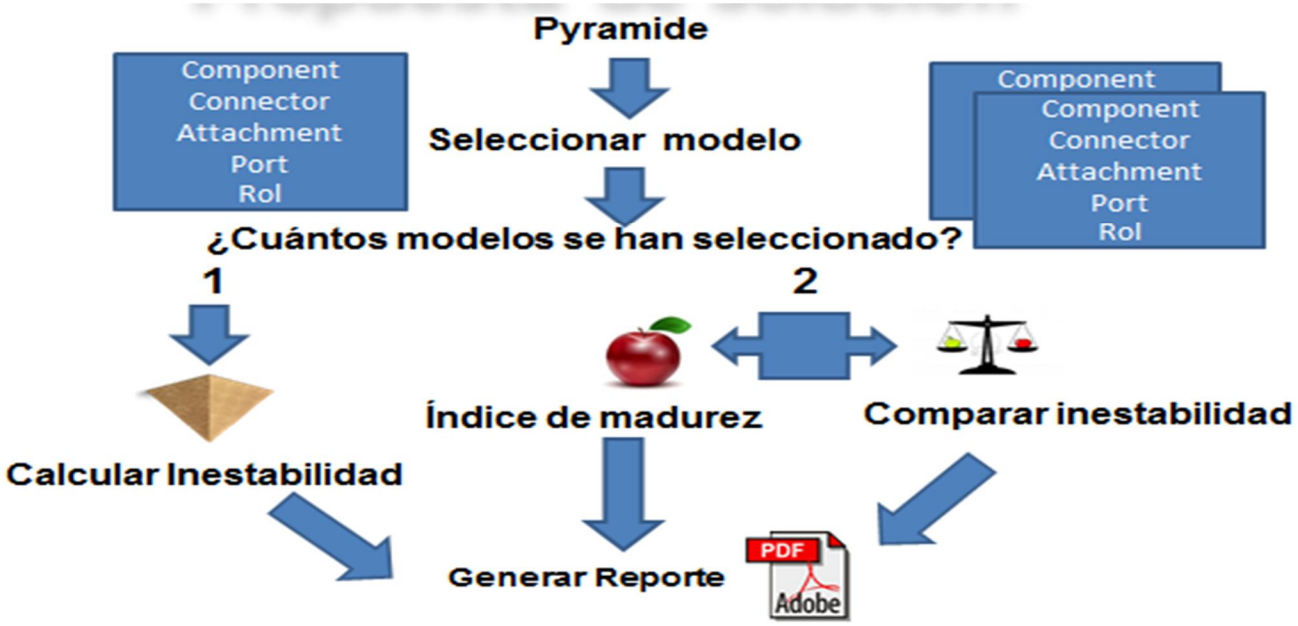

Figura 19: Flujo de funcionalidades ofrecidas por Pyramide

Bibliotecas usadas

Se utilizaron varias bibliotecas, a continuación se detalla el uso de estas:

- **Com.itextpdf:** Se empleó para crear y escribir en un documento PDF.
- **Java.io.File:** Se utilizó en el trabajo con ficheros.
- **Javax.swing.filechooser:** Creación de ventanas para selección de ficheros.
- **Org.acmestudio:** Utilizada en el trabajo con Acme Studio.
- **Javax.swing.UIManager:** Se empleó para alcanzar una mejor apariencia de las interfaces.

2.7. Conclusiones del capítulo

El desarrollo de este capítulo arrojó las siguientes conclusiones:

- Se utilizaron varios de los productos de trabajo propuestos por la metodología OpenUP, con el propósito de documentar y guiar el proceso de desarrollo de Pyramide, lo cual condujo a un mejor entendimiento de lo que se debía implementar.
- Se realizó el diseño de clases y la validación de este a través de las métricas TOC y RC, ratificándose el uso de los patrones de diseño empleados en la solución y evidenciando la calidad que el mismo posee.
- Se expusieron elementos de la implementación, tales como: descripción de funcionalidades, las bibliotecas utilizadas y la explicación del flujo de trabajo de Pyramide, aspectos que promueven el entendimiento por parte del cliente final.

CAPÍTULO 3: Validación y Pruebas

3.1. Introducción

El presente capítulo se centra en la validación de las métricas seleccionadas y la utilización de Pyramide para comprobar que resuelve el problema planteado, a través del método Delphi. Se realizan pruebas de caja blanca y de caja negra para evaluar la efectividad del código y la puesta en marcha de las funcionalidades, respectivamente. Se aplica el plugin a la arquitectura de dos subsistemas de Cedrux para obtener su coeficiente de estabilidad y con los resultados arrojados apoyar el proceso de toma de decisiones arquitectónicas.

3.2. Método Delphi

El método Delphi es un procedimiento eficaz y sistemático que tiene como objetivo la recopilación de opiniones de expertos sobre un tema en particular, en este caso la evaluación de arquitectura. Se aplica con la idea de obtener mediante respuestas a cuestionarios, los criterios que emitan los expertos acerca de la propuesta de solución.

Este método se considera conveniente para realizar la validación de la solución propuesta debido a que su fuente de información está compuesta por un grupo de personas que poseen un conocimiento elevado de la materia a evaluar.

La aplicación de este método está dada por dos cuestiones fundamentales:

- ✓ Elaboración del cuestionario.
- ✓ Selección de los expertos.

En el presente trabajo estas cuestiones fueron aplicadas de la forma que se explica a continuación.

Selección del grupo de expertos

En aras de seleccionar el grupo de expertos se determinaron las áreas del conocimiento que estos deben dominar, determinándose que estas serían: arquitectura de software y su proceso de evaluación, modelado de arquitectura y lenguajes de descripción arquitectónica.

Determinación del coeficiente de conocimiento de los expertos

Las características de los expertos influyen decisivamente en la confiabilidad de los resultados obtenidos y expresan calificación técnica, capacidad de emitir una decisión al respecto, conocimientos específicos sobre el tema a evaluar, disposición a participar, entre otros.

Para verificar el nivel que tienen los expertos sobre los temas antes expuestos se determinó el Coeficiente de competencia (K) a partir de su conocimiento o información sobre el tema (Kc) y el Coeficiente de argumentación o valoración (Ka) mediante la siguiente fórmula: K= 1⁄2 (kc + ka). La interpretación de los coeficientes de competencias es la siguiente:

• Si 0,8 < k < 1 Coeficiente de competencia alto.

• Si 0,5 < k < 0,8 Coeficiente de competencia medio.

• Si k < 0,5 Coeficiente de competencia bajo.

Para determinar el coeficiente de conocimiento o información (Kc) el experto marcará en la casilla enumerada, que se encuentra en la encuesta de autovaloración, según su criterio acerca de la capacidad que él tiene sobre los temas que han sido sometidos a su consideración, en una escala del 0 a 10 y que después para ajustarla a la teoría de las probabilidades se multiplicará por 0,1.

Para determinar el coeficiente de argumentación o valoración (Ka) se ofrece una tabla con cierta información (encuesta de autovaloración). El experto debe marcar, según su criterio, los elementos que le permiten argumentar su evaluación del nivel de conocimiento seleccionado anteriormente. Las marcas de los expertos se traducen a puntos teniendo en cuenta la escala que se muestra en la siguiente tabla de escalas:

Tabla 10. Escalas de puntuación

No	Fuentes	Alto (A)	Medio(M)	Bajo (B)
P1	Análisis teóricos realizados por usted	0.2	0.15	0.10
P2	Su experiencia	0.2	0.15	0.10
P3	Trabajos de autores nacionales	0.2	0.15	0.10
P4	Trabajos de autores extranjeros	0.15	0.15	0.10
P5	Su intuición	0.2	0.15	0.05

Tabla 11. Coeficiente de competencia de los expertos

Expertos	P1	P2	P3	P4	P5	ka	Kc	K	Competencia
E1	0.2	0.2	0.15	0.15	0.2	0.9	0.9	0.9	Alto
E2	0.15	0.2	0.15	0.15	0.15	0.8	0.7	0.75	Medio
E3	0.15	0.15	0.10	0.15	0.15	0.7	0.7	0.7	Medio
E4	0.15	0.10	0.2	0.15	0.05	0.65	0.6	0.62	Medio
E5	0.10	0.2	0.15	0.10	0.05	0.6	0.5	0,55	Medio

Lanzamiento de la encuesta

Habiendo seleccionado los expertos, se procede a la elaboración del cuestionario que será enjuiciado por estos en aras de emitir su criterio acerca de la solución propuesta. La encuesta aplicada con tal objetivo consta de 7 preguntas encaminadas a obtener una evaluación crítica de dicha solución.

Validación de la propuesta

Para validar el plugin desarrollado se aplicó la encuesta mostrada en el **Anexo 3**, la cual se encuentra asociada a la validación de la métrica adaptada y a la utilización de la aplicación.

Indicadores de medición

Para el análisis de los resultados de la puesta en práctica de la presente investigación, se toma como variable demostrativa el apoyo a la toma de decisiones.

El objetivo de dicha encuesta es:

- Determinar la utilidad de Pyramide, para resolver el problema planteado en la introducción del presente trabajo. Para esto se formularon las preguntas 5, 6 y 7 de la encuesta.
- Demostrar que se da cumplimiento a la variable establecida.

Utilidad de Pyramide

Para valorar este aspecto se tuvieron en cuenta las respuestas de los expertos en la encuesta mostrada en el anexo 3, enmarcada en demostrar la necesidad de obtener un resultado cuantitativo acerca de la estabilidad de la arquitectura para tomar decisiones sobre esta durante el proceso de verticalización de Cedrux, permitiendo ajustarse a las características de cada entidad del país. El objetivo principal de aplicar la encuesta es el de demostrar la efectividad del plugin desarrollado.

La pregunta 1 de la encuesta estaba centrada en la validación de la adaptación a servicios, propuesta para la métrica Inestabilidad. Las respuestas podían ser positivas o negativas; estas se muestran en la figura 20.

Figura 20: Respuestas a la pregunta 1 de la encuesta

Con el propósito de obtener un intervalo para especificar las decisiones arquitectónicas que puedan tomarse una vez conocido el grado de estabilidad de la arquitectura definida, se realizó la pregunta número 2 de la encuesta. Los resultados se muestran a continuación.

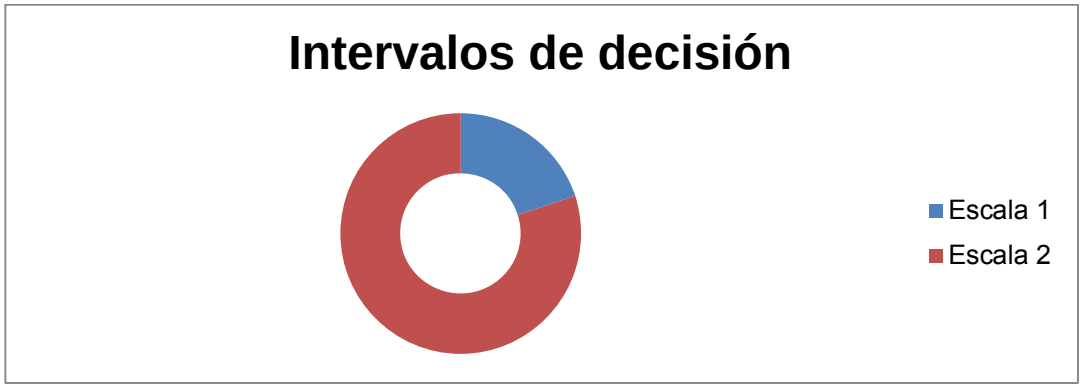

Figura 21: Escala de estabilidad escogida por los expertos

Las respuestas a las preguntas 5, 6 y 7 podían ser positivas o negativas. El resultado arrojado por estas se muestra a continuación.

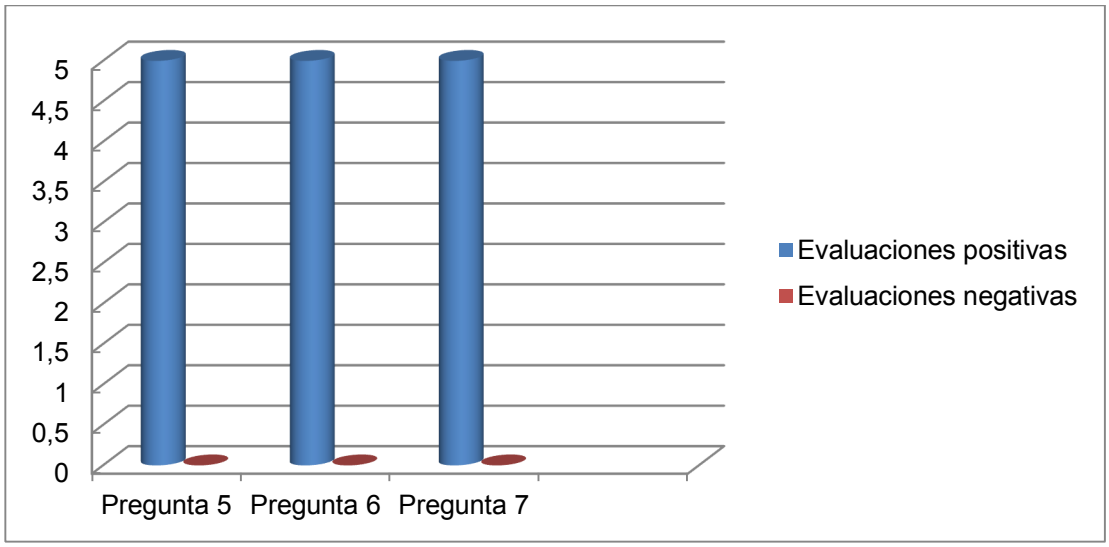

Figura 22: respuestas a las preguntas 5, 6 y 7 de la encuesta

Se propuso la pregunta 3 de la encuesta para validar las decisiones arquitectónicas propuestas y establecer en qué intervalo podrían aplicarse de forma óptima.

- Decisión 1: Fusionar componentes en función de minimizar las dependencias.
- Decisión 2: Retirar servicios.
- Decisión 3: Incorporar servicios.
- Decisión 4: Cambiar los parámetros entrada y salida de los servicios.
- Decisión 5: La modificación interna de una funcionalidad o componente del sistema no afectará a las funcionalidades o componentes que dependan de esta.
- Decisión 6: Dividir un componente en varios componentes para balancear las funciones y aumentar la reutilización.

Los resultados arrojados se muestran a continuación:

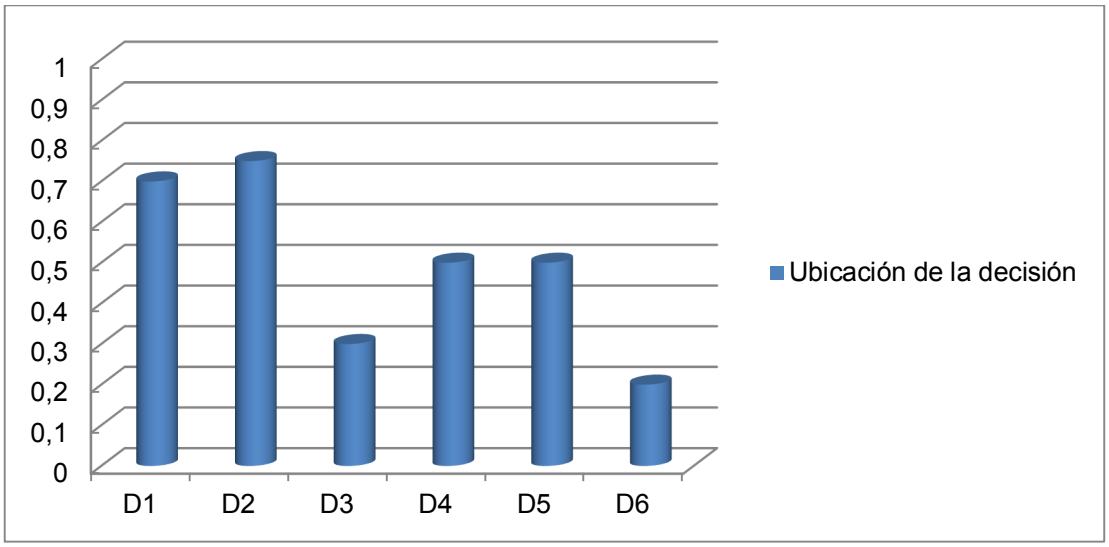

Figura 23: Intervalo para cada decisión arquitectónica

Los resultados mostrados en las figuras 19, 20, 21 y 22 muestran el grado de aceptación de Pyramide por parte de los arquitectos de sistema de las distintas líneas de Cedrux. Las decisiones validadas por los expertos se adjuntan al reporte detallado como sugerencias, según el coeficiente de estabilidad que posea el subsistema.

3.3. Pruebas de Software

La prueba es un proceso de ejecución de un programa con la intención de descubrir errores. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces. Una prueba tiene éxito si descubre un error no detectado hasta entonces (Guide, 2007).

3.3.1. Pruebas de caja blanca

Con la realización de **pruebas de caja blanca** se comprueban los caminos lógicos del software y se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado, este método de pruebas se realiza sobre el código, por lo que requiere del conocimiento de la estructura interna del programa y debe garantizar como mínimo que: (Guide, 2007)

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

A continuación se presenta la realización de esta prueba utilizando la técnica del camino básico, la cual consta de varios pasos: (Guide, 2007)

- ✓ 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.
- ✓ Derivación de casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Para llevar a cabo la prueba de camino básico se tomó un trozo de código correspondiente a uno de los métodos de la clase StabilityModel como se evidencia en la siguiente figura.

```
public List<Double> stabilityPos(){
      double stab=0;  1
      List<Double> allinestab=new ArrayList<Double>();  2
      for (int i = 0; i < acme.componentsInputs().size(); i++) {  3 y 4
            if((acme.componentsInputs().get(i)+acme.componentsOutputs
().get(i))==0){  5
                  allinestab.add(0.0); 6
            }else{  7
            stab=acme.componentsOutputs().get(i)/(acme.componentsInputs
().get(i)+acme.componentsOutputs().get(i));  8
            allinestab.add(stab);  9
            }
      }
      return allinestab;      10
}
```

Figura 24: Código fuente del método stabilityPos de la clase StabilityModel

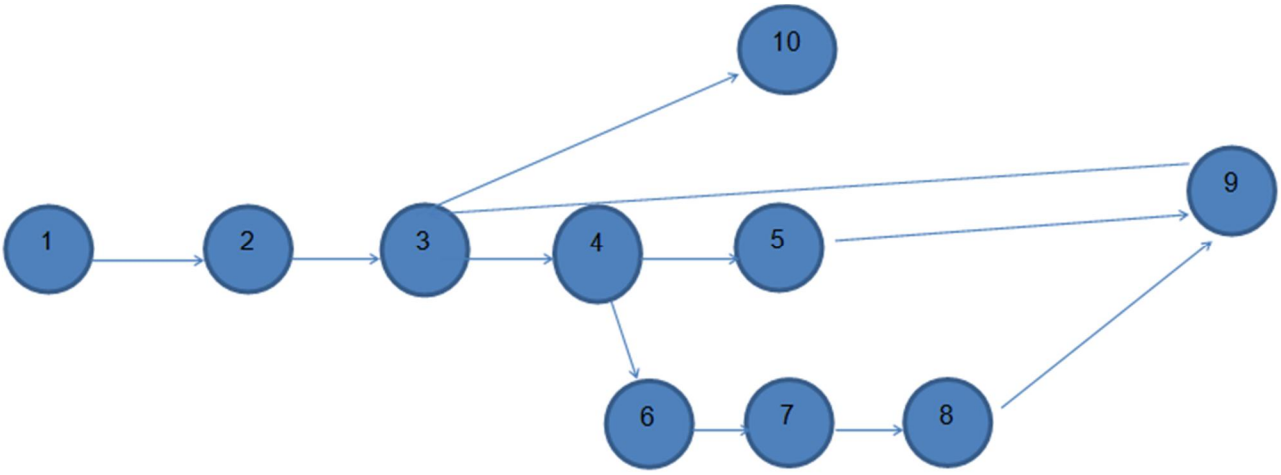

Figura 25: Grafo de flujo asociado al método stabilityPos de la clase StabilityModel

Una vez dibujado el grafo se procede al cálculo de la complejidad ciclomática de este, dando como resultado 3 al aplicar las fórmulas fundamentales para este cálculo.

V (G)= Número de regiones = 3

V (G)= *A - N + 2,* donde: A es el número de aristas del grafo y N es el número de nodos, por tanto:

V (G)= 11-10+2= 3

V (G)= *P* + 1, donde: P es el número de nodos predicado (nodo del cual salen varias aristas) contenidos en el grafo G, por tanto:

V (G)= 2+1=3

Calculada la complejidad ciclomática se procede a establecer un conjunto básico de caminos independientes, como sigue:

Camino 1 (1-2-3-10)

Camino 2 (1-2-3-4-5-9-3-10)

Camino 3 (1-2-3-4-6-7-8-9-10)

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

Camino 1 (componentsInputs ().size<0)

Camino 2 (componentsInputs () + componentsOuputs ()=0)

Camino 3 (componentsInputs () + componentsOuputs () distinto de 0)

De esta forma queda aplicada la técnica de camino básico al código representado en la figura.

3.3.2. Pruebas de caja negra

El método de **pruebas de caja negra** se lleva a cabo sobre la interfaz del software. El objetivo es demostrar que las funciones del software son operativas, que las entradas se aceptan de forma adecuada, produciendo un resultado correcto, y que la integridad de la información externa se mantiene, en la realización de estas pruebas no se ve el código. (Guide, 2007)

Estas pruebas se centran principalmente en los requisitos funcionales del software, permitiendo encontrar: (Guide, 2007)

- ✓ Funciones incorrectas o ausentes.
- ✓ Errores de interfaz.
- ✓ Errores en estructuras de datos o en accesos a las Bases de Datos externas.
- ✓ Errores de rendimiento.
- ✓ Errores de inicialización y terminación.

Se realizaron dos iteraciones de este tipo de pruebas, resultando una no conformidad referente al escenario: Selección de dos modelos, donde se estaba cargando vacío uno de los archivos en la primera iteración de estas. A continuación se presentan las figuras, donde se muestran los casos de prueba

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
[1: Seleccionar Modelo]	El sistema permitirá la selección de los modelos a evaluar.	EP 1.1: Selección de un modelo.	– En la barra de menú de la herramienta Acme Studio, seleccionar el ítem File, se escoge la opción Import y en su interfaz se escoge la opción file system. Acto seguido se busca el directorio del proyecto y se da clic en finalizar. Una vez disponible el proyecto en el árbol de navegación de Acme Studio se selecciona dentro de este, a través de un clic el fichero. Acme del modelo a evaluar. Activándose la función Inestabilidad, representada por una Pirámide invertida.
		EP 1.2: Selección de dos modelos.	– En la barra de menú de la herramienta Acme Studio, seleccionar el ítem File, se escoge la opción Import y en su interfaz se escoge la opción file system. Acto seguido se busca el directorio del proyecto y se da clic en finalizar. Una vez disponible el proyecto en el árbol de navegación de Acme Studio se selecciona dentro de este, a través de un clic el fichero. Acme del modelo a evaluar.

Figura 26: Escenarios del requisito Seleccionar modelo.

Habiendo visto los escenarios que pueden ocurrir como parte del requisito, se procede a la descripción de la variable que será probada, en este caso, la variable **modelo**, para especificar los elementos a considerar en su ejecución. La figura 27 muestra dicha descripción.

1.1.1 Descripción de variable

[Para cada variable de realiza una descripción de modo que se conozcan los principales elementos de las mismas]
Considerar la siguiente guía:

Menús de pestaña

- seleccione un elemento en cada turno

Subir archivos

- en blanco
- archivo de 0 byte
- archivos grandes
- archivo con nombre corto
- archivo con nombre grande
- nombre de archivo sintácticamente incorrecto, si es posible (por ejemplo, "Nombre Con Espacios.tar.gz")]

Figura 27: Descripción de la variable

Una vez conocida la descripción de la variable, se muestran los posibles estados que esta puede obtener en el escenario 1.1.

No	Nombre de campo	Tipo	Válido	Inválido	Inválido	Inválido
[1]	[Archivo con extensión .Acme]	[Archivo]	Archivo de extensión .Acme	Selección de un archivo que no sea de la extensión .Acme	Selección de un archivo vacío con extensión .Acme	Selección de un archivo con extensión .Acme donde los elementos que lo componen no estén en la taxonomía definida por el lenguaje Acme.

Figura 28: Posibles estados de la variable modelo

Id del escenario	Escenario	Archivo.acme	Respuesta del sistema	Resultado de la prueba
EP 1	Selección de un modelo	Capital Humano. Acme	Se activa la funcionalidad representada por la Pirámide invertida, indicando que puede aplicarse la métrica Inestabilidad.	Se activa la funcionalidad representada por la Pirámide invertida, indicando que puede aplicarse la métrica Inestabilidad.
EP 1	Selección de un modelo	Capital Humano.jpg	Se mantiene deshabilitada la funcionalidad representada por la Pirámide invertida, indicando que no puede aplicarse la métrica Inestabilidad para el modelo seleccionado.	Se mantiene deshabilitada la funcionalidad representada por la Pirámide invertida, indicando que no puede aplicarse la métrica Inestabilidad para el modelo seleccionado.
EP 1	Selección de un modelo	Capital Humano. Acme (elementos modificados).	El sistema emite un mensaje indicando que el archivo seleccionado puede contener datos elementos erróneos.	El sistema emite un mensaje indicando que el archivo seleccionado puede contener datos elementos erróneos.
EP 1	Selección de un modelo	Selección de un archivo vacío con extensión .Acme	El sistema muestra un mensaje indicando que el archivo seleccionado se encuentra vacío.	El sistema muestra un mensaje indicando que el archivo seleccionado se encuentra vacío.

Figura 29: Juego de datos para el escenario 1.1

Id del escenario	Escenario	Archivo1.acme	Archivo2.acme	Respuesta del sistema	Resultado de la prueba
EP 1	Selección de dos modelos	Capital Humano1. Acme	Capital Humano. Acme	Solo se activan las funcionalidades representadas por la Balanza y la Manzana, indicando que pueden aplicarse las métricas Inestabilidad para comparar en caso del primero ye índice de madurez del software en el caso de la segunda.	No se activan ninguna de las funcionalidades correspondientes a la selección de dos modelos.
EP 1	Selección de dos modelos	Capital Humano1. Acme	Capital Humano.jpg	Se mantienen deshabilitadas las funcionalidades representadas por la Balanza y la Manzana, indicando que no pueden aplicarse las métricas Inestabilidad para comparar en caso de la primera e índice de madurez del software en el caso de la segunda.	Se mantienen deshabilitadas las funcionalidades representadas por la Balanza y la Manzana, indicando que no pueden aplicarse las métricas Inestabilidad para comparar en caso de la primera e índice de madurez del software en el caso de la segunda.

Figura 30: Juego de datos del escenario 1.2

3.4. Despliegue de la solución

Una vez realizado el diseño, la implementación y validación del plugin, se procede al despliegue en los proyectos de Cedrux. Para realizar este proceso de forma correcta se requiere que la computadora perteneciente al arquitecto, cuente con los requisitos necesarios para que corra el sistema. La figura 31 muestra los elementos necesarios a tener en cuenta a la hora de ser desplegado.

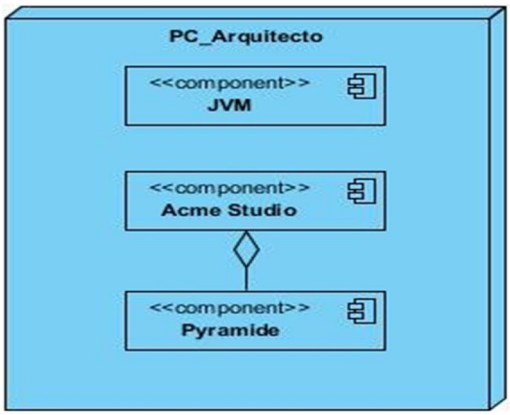

Figura 31: Diagrama de despliegue

3.5. Aplicación de Pyramide en dos subsistemas de Cedrux

En este epígrafe se utiliza Pyramide para predecir el grado de estabilidad que poseen dos de los subsistemas de Cedrux, se realiza una interpretación de dichos resultados y se proponen las decisiones arquitectónicas que deberían tomarse en cada caso. A petición de los arquitectos de dichos subsistemas solo se aplica la funcionalidad referente al cálculo de la Inestabilidad y se realiza una comparación entre los modelos de ambos subsistemas. La figura 32 muestra la leyenda para el modelado arquitectónico de la arquitectura de estos subsistemas siguiendo las pautas establecidas en el estilo arquitectónico de Cedrux definido en (Arias, 2013).

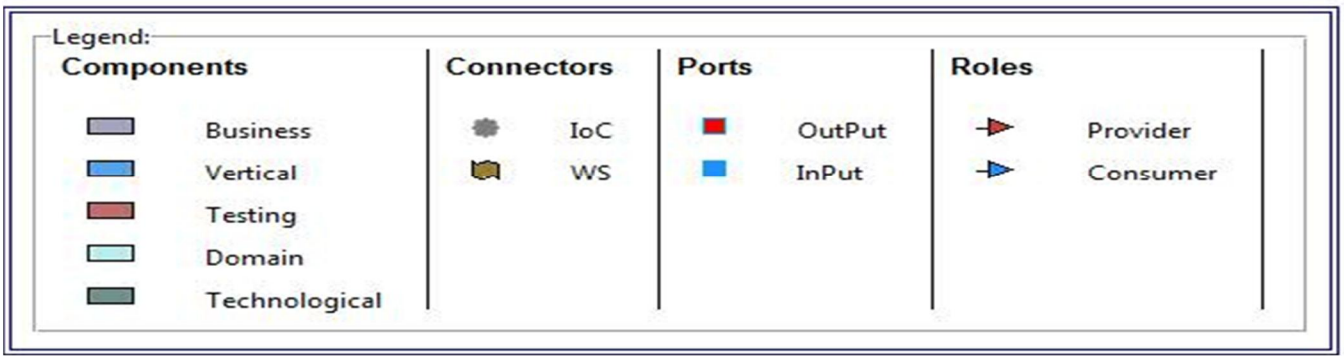

Figura 32: Leyenda del estilo arquitectónico de Cedrux

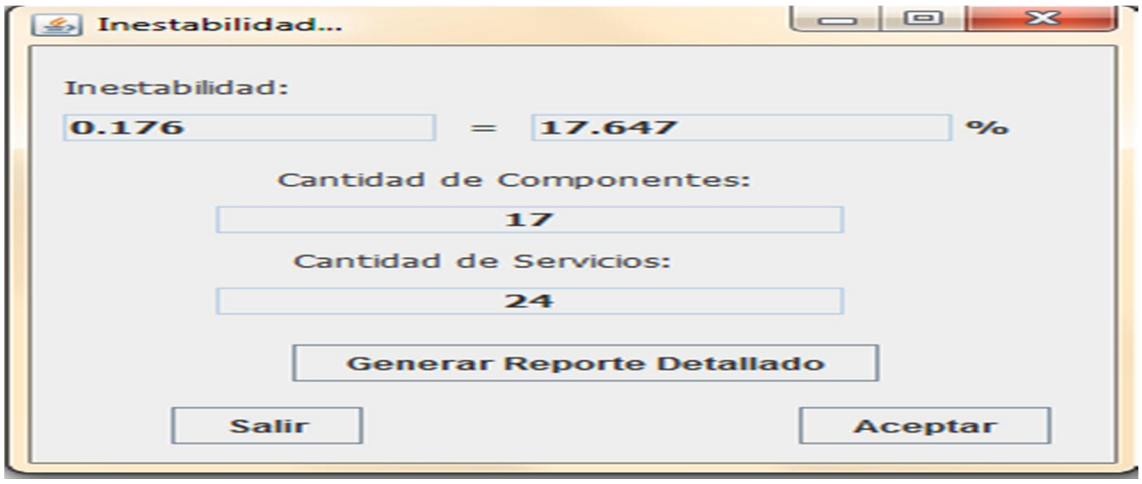

Figura 33: Interfaz de Inestabilidad para el subsistema Capital Humano

La figura 33 muestra el grado de estabilidad que posee el subsistema Capital Humano, esto puede ser traducido como estable, si se tiene en cuenta que la métrica Inestabilidad plantea que mientras más cercano a cero esté el valor de evaluación, más estable será. Dentro de los intervalos definidos para Cedrux, este valor obtenido muestra la alta estabilidad del subsistema, por lo que no sería tan necesario aplicar decisiones para estabilizarlo aún más.

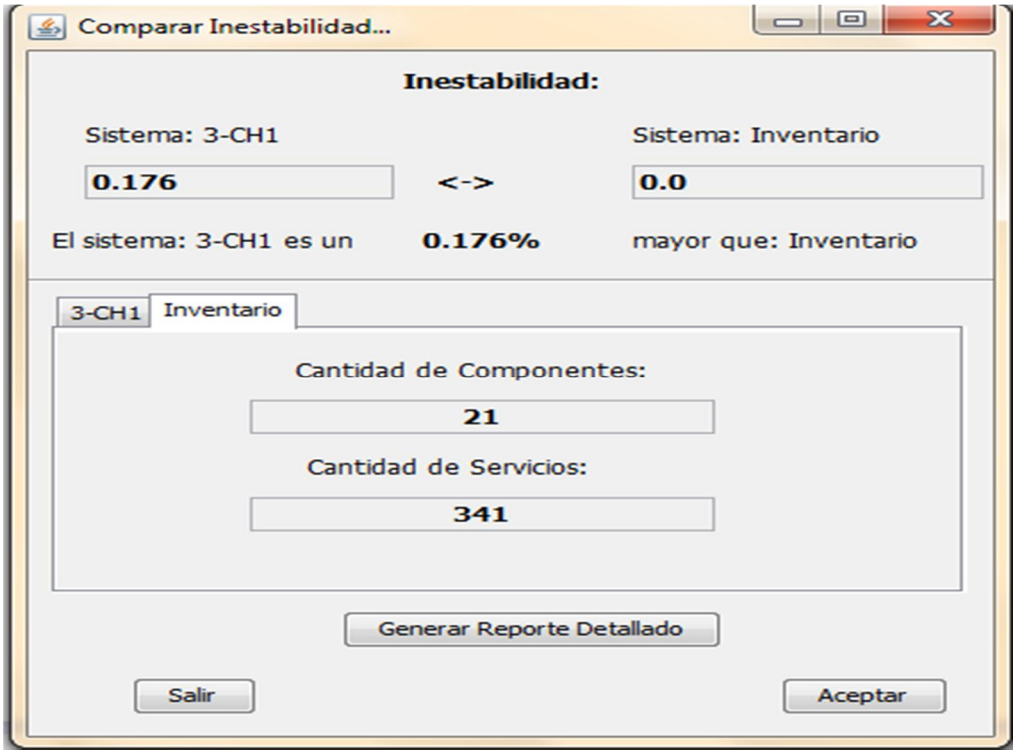

Figura 34: Comparación de Inestabilidad entre los subsistemas Capital Humano e Inventario

La figura 34 muestra una comparación de inestabilidad entre los modelos arquitectónicos de los subsistemas Capital Humano e Inventario. Resulta apreciable el hecho de que ambos modelos poseen una alta estabilidad de acuerdo a la escala definida. En el caso particular de inventario, puede decirse que posee el caso ideal para la evaluación de la estabilidad.

3.6. Decisiones arquitectónicas

A través de la encuesta aplicada se validó el conjunto de decisiones arquitectónicas planteadas por el autor del presente trabajo y se propusieron otras por algunos de los expertos. A continuación se presenta el catálogo de decisiones arquitectónicas, donde cada una de ellas está asociada a un intervalo dentro de la escala 2 expuesta en la pregunta 3 de la encuesta, ver anexo 2**.**

Catálogo de decisiones:

- Fusionar componentes en función de minimizar las dependencias (0.7).

- Retirar servicios (0.75).
- Incorporar servicios (0.3).
- Cambiar los parámetros entrada y salida de los servicios (0.5).
- La modificación interna de una funcionalidad o componente del sistema no afectará a las funcionalidades o componentes que dependan de esta (0.5).
- Dividir un componente en varios componentes para balancear las funciones y aumentar la reutilización (0.2).
- Colocar restricciones adicionales en función de eliminar acoplamientos innecesarios (0.6).
- Integrar servicios para disminuir dependencias (0.7).

3.7. Conclusiones

- El método Delphi es muy eficiente para validar investigaciones contando con el criterio de especialistas del tema en cuestión. Su utilización propició la validación de la solución propuesta.
- La realización de pruebas de caja blanca y de caja negra, proporcionan un mecanismo de detección de errores en el código fuente y en los distintos estados de las variables respectivamente, aspecto fundamental para el proceso de desarrollo de software, en su aplicación se obtuvieron resultados satisfactorios, mostrando así que Pyramide cuenta con un adecuado funcionamiento y cumple las necesidades planteadas por el cliente.
- Pyramide ofrece resultados precisos acerca del grado de estabilidad de los subsistemas de Cedrux, pero su mayor relevancia radica en el catálogo de decisiones arquitectónicas que provee, para disminuir o aumentar la estabilidad de un subsistema, siempre y cuando sea necesario.

CONCLUSIONES GENERALES

El desarrollo de la presente investigación condujo a las siguientes conclusiones generales:

- Se realizó un estudio calificador de las métricas existentes para medir la estabilidad, los enfoques que conllevan a la utilización de estas y el aporte práctico que producen, al arrojar resultados cuantitativos. Se trataron definiciones claves para la investigación.
- Se elaboró el diseño del plugin a implementar, validando el mismo teniendo en cuenta: las métricas TOC y RC, las cuales mostraron resultados satisfactorios en su aplicación.
- Se realizó la implementación del plugin, así como la aplicación de pruebas de software para su validación, dándole solución a las necesidades de los arquitectos del centro CEIGE de obtener el grado de estabilidad de su subsistema durante el proceso de verticalización.
- Se propuso un catálogo de decisiones a tomar, según el grado de estabilidad que posea el subsistema, contribuyendo a la toma de estas, lo que demuestra el cumplimiento del objetivo propuesto.

RECOMENDACIONES

- ✓ Se recomienda el uso de Pyramide en todos los subsistemas de Cedrux para obtener datos precisos y tomar mejores decisiones arquitectónicas a la hora de implementar las verticalizaciones.
- ✓ Incrementar el catálogo de decisiones arquitectónicas que puedan tomarse al evaluar la estabilidad con Pyramide.
- ✓ Aprovechar las ventajas que ofrece la herramienta AcmeStudio para extender funcionalidades implementadas en el lenguaje de programación Java.

BIBLIOGRAFÍA

Alvarez, Larisa González. 2012. *MÉTODO PARA LA EVALUACIÓN ARQUITECTÓNICA DE CEDRUX[Tesis de maestría].* La Habana : Universidad de las Ciencias Informáticas, 2012.

Arias, Yusnier Matos. 2013. *Estilo Arquitectónico para el Sistema Integral de Gestión Cedrux.* La Habana : s.n., 2013.

Bosch, Jan. 2006. *Design & Use of Software Architectures. Segunda Edición.* s.l. : Addison Wesley, 2006.

Camacho, Erika. 2004. *Arquitecturas de software.* 2004.

Chávez, Michel Arias. 2006. *La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software.* Sán José de Costa Rica : Revista Intersedes, Universidad de Costa Rica, 2006.

Clayberg, Erick. 2008. *Eclipse Plug-ins Third Edition.* Massachusetts : s.n., 2008.

Codorniú, César Lage. 2010. *Solución arquitectónica de la Configuración General de Cedrux para la parametrización de negocio del sistema.[Tesis de maestría].* La habana : Universidad de las Ciencias Informáticas, 2010.

Eclipse. 2012. *epf.eclipse.org. Introduction to OpenUp. epf.eclipse.org. Introduction to OpenUp. [En línea] 24 de 1 de 2012.* 2012.

Fernández, Osmar Leyet. 2011. *Propuesta metodológica para la obtención de los componentes de software en los proyectos del sistema Cedrux.[Tesis de Maestría] .* La Habana : Universidad de las Ciencias Informáticas, 2011.

González, Yisét Milán. 2011. *Propuesta de escenarios arquitectónicos y atributos de calidad para la evaluación arquitectónica del sistema Cedrux.* La Habana : Universidad de las Ciencias Informáticas, 2011.

Grady Booch, Jim Rumbaugh, Ivar Jacobson. 2004. *Lenguaje Unificado de Modelado.* 2004.

Guide, Rational. 2007. *Ayuda en linea.* 2007.

Hernández León, Dr.Rolando A , González Coello,Sayda. 2002. *El paradigma cuantitativo de la investigación científicaed. La Habana, Editorial Universitaria.* 2002.

IEEE. 2000. *Std 1471-2000 .* 2000.

ISO/IEC. 2003. *Software Engineering Product Quality.* 2003.

Ivar Jacobson, Grady Booch,James Rumbaugh. 1999. *Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 1999.

Jazayeri. **Evolution, On Architectural Stability and. 2002.** Viena : s.n., 2002.

Mario Barbacci, Mark Klein. 2005. *Quality Atributes.* Pennsylvania : Software Engineering Institute, Carnegie Mellon University, 2005.

Martin, Robert. 1997. ñ*Stability***ò** *C++ Report. February 1997.* 1997.

Martínez, Nemury Silega. 2010. *Guía Metodológica para gestionar la integración de componentes en los proyectos de Cedrux.[Tesis de Maestría].* La Habana : Universidad de las Ciencias Informáticas, 2010.

Normalización, Oficina Cubana de. 2005. *ISO/IEC 9126:2005.* La Habana : s.n., 2005.

Pérez, Isaías Carrillo. 2008. *Metodologías de desarrollo de software.* 2008.

Pérez, María José Pérez. 2012. *Guía Comparativa de Metodologías Ágiles.* Segovia : Universidad de Valladolid, 2012.

Pressman, Roger. 2006. *Ingeniería de Software Un Enfoque Práctico .* 2006.

ð . 2002. *Ingeniería de Software, Un Enfoque Práctico, Quinta Edición.* : McGraw-Hill, 2002.

Rami Bahsoon, Wolfgang Emmerich. 2006. *Architectural Stability and Middleware:An Architecture-Centric Evolution.* Londres : s.n., 2006.

Regalado, Yamila Vigil. 2009. *Metodología de evaluación de arquitectura de software.* La Habana : Universidad de las Ciencias Informáticas, 2009.

Reynoso, Carlos Billy. 2004. *Lenguajes de descripción de arquitectura.* Buenos Aires : s.n., 2004.

Rick Kazman, Paul Clements. 2007. *Evaluating Software Architectures. Methods and case studies 2da Edición.* s.l. : Adison Wesley, 2007.

Rick Kazman, Paul Clements,Len Bass. 2003. *Software Architecture in Practice,Second Edition.* s.l. : Addison Wesley, 2003.

Ríos, Dr. José Carlos del Toro. 2008. *Documento Visión del Programa ERP-Cuba.* La habana : s.n., 2008.

Sommerville, Ian. 2005. *Ingeniería de Software, Séptima Edición.* s.l. : Addison Wesley, 2005.

Suárez, María Cristina Carreón. 2008. *Construcción de un catálogo de patrones de requisitos funcionales para ERP.[Tesis de maestría].* Cataluña : Universidad Politécnica de Cataluña, 2008.

Vega, Anisleydi Céspedes. 2008. *Procedimiento para la Evaluación de Arquitecturas de Software basadas en Componentes.* La Habana : Universidad de las Ciencias Informáticas, 2008.

Vera, Angelo Benvenuto. 2006. *Implementación de Sistemas ERP, su impacto en la gestión de la empresa e integración con otras TIC.* 2006.

Vestal, Steve. 1993. *A cursory overview and comparison of four Architecture Description Languages.* s.l. : Honeywell Technology Center, 1993.

Wolf, Alexander. 1997. *Succeedings of the Second International Software Architecture.* s.l. : ACM SIGSOFT, 1997.