Universidad de las Ciencias Informáticas

Facultad 3

Título: **Extensión visual de AcmeLib para la modelación y análisis de la arquitectura de Cedrux.**

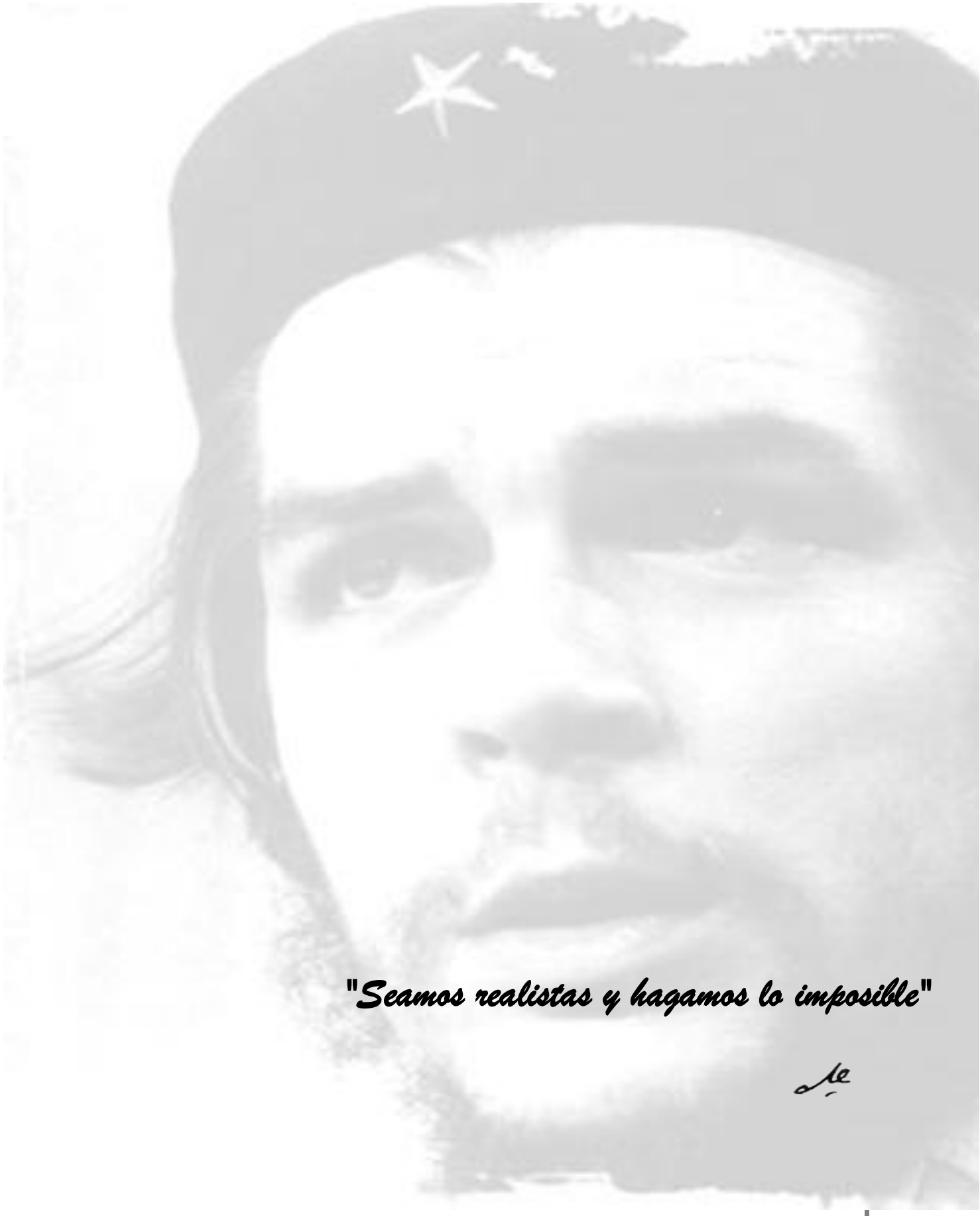
Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autores: Lizander Ocaña González.
Osvaldo Pérez Rojas.
Tutor: Ing. Yusnier Matos Arias.

Ciudad de La Habana

Junio del 2012

"Año 54 de la Revolución"



"Seamos realistas y hagamos lo imposible"

Le

DECLARACIÓN DE AUTORÍA

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

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

Lizander Ocaña González	Osvaldo Pérez Rojas	Ing. Yusnier Matos Arias
Firma del Autor	**Firma del Autor**	**Firma del Tutor**
_____	_____	_____

DATOS DE CONTACTO

Tutor: Ing. Yusnier Matos Arias

Edad: 26 años.

Ciudadanía: cubano.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

Categoría docente: Profesor instructor.

E-mail: ymarias@uci.cu.

Ingeniero en Ciencias Informáticas, graduado en 2008 en la Universidad de las Ciencias Informáticas. Instructor. Tres años de experiencia en el desarrollo de software. Tres años de graduado.

AGRADECIMIENTOS

Primero que todo quisiera agradecer a las dos personas que más he querido en la vida, a mis difuntos abuelos Luisa y Pablo, los cuales son los máximos responsables de que esto suceda, gracias, a ellos que me dieron la mejor educación que se le pueda dar a una persona…y donde quiera que estén quiero que sepan que nunca los voy a olvidar.

A mis padres por brindarme todo el amor del mundo, en espacial a mi mamá que es la que siempre ha estado a mi lado en las buenas y en las malas, sé que aunque sea todo un hombre siempre seré su niño...eso es lo que siempre me dice. A mi padre que no se me ponga bravo que aunque él no lo crea, también lo quiero del tamaño del cielo.

A mi familia en general que siempre han estado muy atentos a los pasos que he dado en mi vida…a mis tíos…a mis primos, no quisiera dejar de mencionar a mi primo Yunielkys que vive en Miami el cual me ha ayudado muchísimo…"a todos", gracias por tratarme como el niño de la familia.

Al mejor profesor que he tenido Roberto Argüelles, el cual me enseñó que todo lo que uno se propone lo puede lograr.

A Maikel y Alkaid que son como hermanos para mí, ojalá y el destino quiera que una vez que nos separemos nos volvamos a ver.

A Yailet por la gran amiga que es, por ocuparse de mi cada vez que me enfermé ella siempre estuvo ahí y por tener la paciencia para aguantar a Maikel todos estos años, de no ser así tendría yo que cargar con él.

A Leisniel, Alexei, Ismel por brindarme su amistad cuando apenas comenzaba a conocer la UCI.

A mi compañero de tesis "El Lisor" por el apoyo brindado durante el desarrollo del trabajo.

A todos mis compañeros, con los cuales he compartido apartamento durante este año.

A todo el que de una forma u otra ha tenido que ver con la llegada de este momento.

Osvaldo

DEDICATORIA

Dedico el presente Trabajo de Diploma a mis padres y a mi profesor del pre-universitario Roberto Argüelles.

Osvaldo

RESUMEN

La Universidad de las Ciencias Informáticas (UCI) tiene entre otras misiones la producción de software y proyectos productivos, los cuales hacen un gran aporte a la economía del país, así como al desarrollo de la industria de software.

Uno de los software desarrollados en la Universidad de las Ciencias Informáticas es el Sistema Integral de Gestión Cedrux, el cual por su alcance nacional es considerado de gran importancia ya que representa un gran beneficio para el país. Este proyecto se encuentra actualmente en desarrollo, se le han identificado una serie de problemas para llevar a cabo el proceso de verticalización, pues no cuenta con elementos suficientes para determinar cuan factible es una u otra variación arquitectónica respecto de otras.

A lo largo de este trabajo se realiza un estudio de cómo se lleva a cabo el proceso de verticalización en el sistema Cedrux. Teniendo como objetivo principal extender la herramienta AcmeStudio con funcionalidades que permitan realizar comparaciones de factibilidad entre distintas variantes de un diseño arquitectónico. Con el fin de reducir el tiempo en que se realiza el proceso de verticalización.

Palabras claves: AcmeStudio, factible, variación arquitectónica, verticalización.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Hoy en día la información se genera cada segundo en todas las organizaciones y en cada uno de sus niveles. En el ámbito empresarial, tener a la mano la información necesaria puede significar una ganancia o una pérdida monetaria. A través de las últimas décadas, han aparecido y evolucionado los sistemas de planeación de recursos empresariales para ayudar en este sector, más conocidos como ERP[1] (del inglés *Enterprise Resource Planning*).

Uno de estos ERP es el Sistema Integral de Gestión Cedrux, el cual se desarrolla en el Centro de Informatización de la Gestión de Entidades (CEIGE), de la Universidad de las Ciencias Informáticas (UCI), con el objetivo de automatizar las distintas áreas de negocio en las entidades empresariales y presupuestadas del país. Este ERP está concebido para ser verticalizado, es decir, debe estar preparado para que se puedan adicionar o eliminar funcionalidades, o simplemente modificarlas con el menor impacto posible para adaptarse a las necesidades o particularidades de clientes específicos.

Para la implementación de las verticalizaciones, es necesario realizar modificaciones que conllevan a realizar cambios estructurales de la arquitectura de Cedrux. Por el hecho de esta última estar basada en componentes, dichas modificaciones se pueden traducir en:

a) Adición o eliminación de estos.

b) Eliminación o modificación de las relaciones entre dichos componentes.

Hasta este momento se puede tratar componente de acuerdo a la definición de la IEEE, la cual plantea que: “Componente es cualquier elemento estructural abstracto, visible, externo, de alto nivel, analizable, que pueda constituir una funcionalidad de la solución del sistema”. Más adelante se concretará el término más enfocado al contexto de este trabajo.

Durante el proceso descrito anteriormente los arquitectos han identificado determinadas afectaciones. Estas están dadas fundamentalmente, por los pocos elementos con que se cuenta para determinar cuán factible es una u otra configuración arquitectónica (respecto de otras) a partir de las variantes que pueden surgir como consecuencia de las modificaciones a la estructura de componentes[2]. Entiéndase por

[1] ERP: Planeación de Recursos Empresariales.
[2] Estructura de componentes: organización estructural dada por los componentes y sus interconexiones en una arquitectura. (es equivalente a una configuración arquitectónica o a un sistema)

configuración arquitectónica, la topología u organización estructural dada por los componentes y sus interconexiones en una arquitectura.

En este sentido se han realizado acciones como la modelación de las verticalizaciones en un Lenguaje de Descripción Arquitectónica (ADL, del inglés *Architectural Description Languages*).

Para la selección de este ADL se realizó una investigación donde se optó por utilizar el ADL Acme para modelar la arquitectura de Cedrux en general y de las verticalizaciones de éste en particular.(MORALES 2009). A partir de lo anterior, la arquitectura puede ser modelada haciendo uso de las facilidades que provee la herramienta AcmeStudio, que es una de las interfaces visuales de Acme. Con esta herramienta se pueden realizar determinados análisis para identificar deficiencias en los modelos realizados.

Actualmente se están realizando extensiones a la misma con el fin de realizar otros análisis sobre la arquitectura. Sin embargo, algunos problemas se mantienen en el proceso de verticalizaciones:

- La herramienta no permite revisar otras variantes de modelos con los que se trabaja, sin perder la información que se tiene del actual.
- Durante la modelación, con la herramienta no es posible realizar comparaciones entre variantes de un modelo arquitectónico, con el fin de determinar mejores valores de factibilidad[3] entre los mismos.

Las deficiencias anteriores retrasan el tiempo en que se implementan dichas verticalizaciones. Estos atrasos están dados fundamentalmente por rediseños y reimplementación de las mismas.

A partir de la problemática anterior, se formula el siguiente **problema a resolver:** Los pocos elementos con que se cuenta para realizar comparaciones de factibilidad entre distintas variantes de la estructura arquitectónica provocan atrasos en la implementación de las verticalizaciones de Cedrux.

Dado el problema anterior se propone como **objeto de estudio**: Las herramientas de modelado arquitectónico[4] basadas en Lenguajes de Descripción de Arquitecturas.

El **objetivo general** de la investigación es desarrollar un plugin para el IDE Eclipse que extienda la herramienta AcmeStudio con funcionalidades que permitan realizar comparaciones de factibilidad entre distintas variantes de un diseño arquitectónico con el fin de reducir el tiempo en que se implementan las verticalizaciones del sistema Cedrux.

[3] Factibilidad: valor que se obtiene mediante una fórmula predefinida y que se da en unidades.
[4] Modelado arquitectónico: simplificación de la realidad creado para entender mejor un sistema

Para cumplimentar el objetivo general de la investigación, se desglosa este en objetivos específicos que contribuirán a una mejor evaluación y análisis de los hitos alcanzados durante el desarrollo del trabajo. Los **objetivos específicos** son los siguientes:

1. Establecer un marco teórico referente a los diferentes ADLs que presentan herramientas gráficas, analizarlos y tomar partido sobre los resultados de dicho análisis con el fin de sentar las bases para el desarrollo de la solución.
2. Desarrollar un *plugin* basado en la arquitectura del IDE Eclipse, que pueda ser integrado a la herramienta AcmeStudio de manera que desde el entorno de esta, se puedan realizar funcionalidades que tributen al cumplimiento del objetivo del trabajo.
3. Realizar pruebas de caja negra y aplicar la herramienta en un caso de estudio con el fin de analizar los resultados y así validar el presente trabajo.

El **campo de acción** se enmarca en la modelación de variaciones arquitectónicas.

Para dar cumplimiento a los objetivos específicos y general se plantean las siguientes **tareas** de investigación:

- Revisión de documentación relacionada con el objeto de estudio del trabajo.
- Análisis de la información a partir de la revisión bibliográfica realizada.
- Descripción del contexto sobre el que se enfoca la solución que se propone.
- Levantamiento de requisitos.
- Propuesta de diseño.
- Validación del diseño propuesto.
- Implementación de la herramienta.
- Diseño de pruebas.
- Aplicación de pruebas de caja negra.
- Descripción del caso de estudio.
- Análisis de los resultados a partir de la aplicación en el caso de estudio.

- Validación del trabajo.

Idea a defender: Si se desarrolla un plugin basado en la arquitectura de Eclipse, que permita realizar comparaciones de factibilidad entre distintas variantes de un diseño arquitectónico, se integra este con la herramienta AcmeStudio y se aplica en el proceso de verticalizaciones de Cedrux, se reducirá el tiempo en que se implementan las verticalizaciones de dicho sistema.

Métodos de investigación:

- **Analítico-sintético:** La utilización de este método teórico posibilitará el análisis de los elementos más importantes que se relacionan con desarrollo de plugin para el IDE Eclipse. Dará la facilidad de obtener y procesar la información de lo más importante y significativo relacionado con este tema.
- **Histórico-lógico:** Mediante la utilización de este método se puede estudiar la evolución que ha tenido hasta la actualidad todo lo referente a plugins para el IDE Eclipse.
- **Modelación:** Este método permitirá la realización de diagramas y modelos que permitan estudiar nuevas relaciones y cualidades del objeto de estudio y así estructurar teóricamente el sistema.

En el capítulo I, Fundamentación Teórica, se tratan los conceptos y definiciones necesarias para llevar a cabo el desarrollo de un plugin de Eclipse. También se realiza un estudio de los Lenguajes de Descripción de Arquitectura que cuentan con herramientas gráficas, así como las herramientas y tecnologías a utilizar para desarrollar un plugin.

En el capítulo II, Características del Sistema, se describen las características del plugin que se desarrollará, se identifican las funcionalidades que debe tener, su diseño, las principales clases que posee y su implementación.

En el capítulo III, Validación y Pruebas, se valida el diseño y la solución del trabajo aplicando la herramienta en un caso de estudio, así como las pruebas cuyo objetivo es detectar y corregir errores, antes de entregar el producto.

CAPITULO 1 FUNDAMENTACIÓN TEÓRICA

INTRODUCCIÓN

En el presente capítulo se hace un estudio de diferentes Lenguajes de Descripción Arquitectónica que presentan herramientas gráficas con el objetivo de tomar ideas de su modo de modelar la arquitectura. La selección se realiza teniendo en cuenta el trabajo con elementos visuales mediante los que pueden ser representadas variantes de un mismo modelo. En este sentido, lo fundamental es el estudio de ADLs que presentan herramientas gráficas, dado que estos por definición, están concebidos para modelar la arquitectura. Se muestran elementos básicos para el desarrollo de plugins para Eclipse, así como la arquitectura de este IDE.

A medida que se desarrolla el capítulo, y en aras de lograr un mejor entendimiento del mismo, se exponen conceptos que a consideración de los autores son de interés en este documento. Finalmente, se presentan las tecnologías y herramientas a utilizar para el desarrollo de la herramienta.

1.1. Arquitectura de software

Arquitectura de Software: se refiere a un grupo de abstracciones y patrones que nos brindan un esquema de referencia útil para guiarnos en el desarrollo de software dentro de un sistema informático.(GUGLIELMETTI 2010) La arquitectura de software, tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad, y disponibilidad.

1.1.1. 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 ADLs los

componentes pueden exponer varias interfaces, las cuales definen puntos de interacción entre un componente y su entorno. (KICILLOF Universidad de Buenos Aires 2004)

1.1.2. Conectores

Representan interacciones entre componentes. Corresponden a las líneas de las descripciones de caja y línea. Ejemplos típicos podrían ser tuberías, 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. (KICILLOF Universidad de Buenos Aires 2004) Resumiendo podemos decir que la función de los conectores es conectar a los componentes y llevar a cabo la comunicación entre ellos.

1.1.3. Configuraciones o Sistemas

Se constituyen como grafos de componentes y conectores. En los ADLs 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. (KICILLOF Universidad de Buenos Aires 2004)

1.2. Lenguajes de Descripción de Arquitectura.

1.2.1. Conceptualización.

Con el desarrollo de la arquitectura de software, podría pensarse que hay abundancia de herramientas de modelado que facilitan la especificación de desarrollos basados en principios arquitectónicos, que dichas herramientas han sido consensuadas y estandarizadas hace tiempo y que son de propósito general, adaptables a soluciones de cualquier mercado vertical y a cualquier estilo arquitectónico. La creencia generalizada sostendría que modelar arquitectónicamente un sistema se asemeja al trabajo de articular un modelo en ambientes ricos en prestaciones gráficas, como es el caso del modelado de tipo CASE o UML, y que el arquitecto puede analizar visualmente el sistema sin sufrir el aprendizaje de una sintaxis especializada. (KICILLOF Universidad de Buenos Aires 2004)

Los ADL se remontan a los lenguajes de interconexión de módulos (MIL) de la década de 1970, pero se han comenzado a desarrollar con su denominación actual a partir de la década de 1990, poco después de fundada la propia arquitectura de software como especialidad profesional. Estos lenguajes surgen por la necesidad de satisfacer los requisitos descriptivos de alto nivel de abstracción que las herramientas basadas en objeto en general y UML en particular no cumplen satisfactoriamente.(DAVID GARLAN Cambridge University Press, pp. 2000)

Los ADL permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento. No existe hasta hoy una definición consensuada y unívoca de ADL, pero comúnmente se acepta que un ADL debe proporcionar un modelo explícito de componentes, conectores y sus respectivas configuraciones. La definición de ADL que habrá de aplicarse en lo sucesivo es la de 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 Informe técnico. Honeywel Tecnology Centerl 1993) . Se estima deseable, además, que un ADL suministre soporte de herramientas para el desarrollo de soluciones basadas en arquitectura y su posterior evolución.(KICILLOF Universidad de Buenos Aires 2004)

Los lenguajes de descripción de arquitecturas, ocupan una parte importante del trabajo arquitectónico desde la fundación de la Arquitectura de Software (AS). Ya que contando con un ADL, un arquitecto puede razonar sobre las propiedades del sistema con precisión, pero a un nivel de abstracción convenientemente genérico.

Un ADL debe modelar o soportar los siguientes conceptos: (KICILLOF Universidad de Buenos Aires 2004)

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

- Composición automática de código explicativo.

En la práctica, se ha hecho muy difícil lograr un ADL que cumpla con todas estas características. En primer lugar porque estas están definidas originalmente en entornos académicos, donde priman los estudios teóricos sobre el tema. En los entornos de producción, la realidad es mucho más rica. Influyen muchas variables que en la teoría no siempre se tienen en cuenta. Estas ideas han sido defendidas por (Medvidovic y Taylor, en:"*A component- and message-based architectural style for GUI software*") es una de las causas por las cuales los ADLs están asociados a dominios de específicos, definidos por

- particularidades de los tipos de aplicaciones que se desarrollan.
- b) tipos de arquitecturas.
- c) tipos de análisis que realizan sobre las arquitecturas.

Lo anterior es importante que se tenga en cuenta para entender por qué existen varios ADLs, y no todos son aplicables al mismo contexto. Incluso, en muchas ocasiones, son incompatibles.

En el siguiente acápite se presentan estudios realizados de un subconjunto de estos ADLs, teniendo en cuenta como indicador principal el soporte que tienen de herramientas gráficas. La selección se hizo a partir de un estudio previo a este trabajo, donde el objetivo era seleccionar un ADL para modelar la arquitectura de Cedrux (MORALES 2009). En dicho estudio aparece un conjunto más amplio de varios ADL y se concluyó en el mismo usar ACME, con AcmeStudio para la modelación de Cedrux.

1.2.2. Lenguajes de descripción de arquitecturas.

En este epígrafe se analizarán algunos ADL que poseen herramientas gráficas, enfocando el estudio a su forma de modelar la arquitectura y las herramientas específicas que usan para esto, sirviendo así de ayuda para el desarrollo de este trabajo.

- ***Wright***

Wright fue desarrollado por David Garlan en la *Universidad Carnegie Mellon* (CMU) como parte del proyecto mayor de ALBE. Es un ADL de propósito general con énfasis en la comunicación y además una

herramienta de formalización de conexiones arquitectónicas. En Wright la interfaz de los componentes se representa a través de una serie de puertos y la de los conectores a través de una serie de roles, donde los puertos describen el comportamiento de los componentes y los roles el comportamiento que deben tener los conectores una vez que se conecten los componentes. Los puertos y roles que se relacionen entre sí deben ser compatibles aunque no existen tipos predefinidos para los componentes y conectores, aquí las interfaces indican cual es el protocolo que rige el comportamiento del componente o conector que se está especificando a través del álgebra de procesos CSP[5], de esta forma se logra describir la interacción que tiene lugar entre el elemento y su entorno, mediante la sincronización de eventos que permiten modelar tanto la invocación de operaciones, como el intercambio de mensajes. En Wright existe la posibilidad de usar cualquier herramienta de análisis o técnica que pueda usarse para CSP por ejemplo el FDR[6], el cual proporciona a Wright la garantía de que un conector esté libre de bloqueo en todos los escenarios posibles. (KICILLOF Universidad de Buenos Aires 2004)

- ***C2***

C2 o Chiron-2 es un estilo de arquitectura de software que se ha impuesto como estándar en el modelado de sistemas que requieren intensivamente pasaje de mensajes y que suelen poseer una interfaz gráfica dominante.(KICILLOF Universidad de Buenos Aires 2004)

El mismo define que un sistema de software está compuesto por componentes ordenados en capas o niveles que se coordinan y comunican mediante peticiones de servicios y notificaciones de eventos), y por conectores los cuales tienen como única función conectar a los componentes y llevar a cabo la comunicación entre ellos, en cada nivel o capa hay un solo conector el cual actúa de delimitador para su nivel correspondiente. En el caso de los componentes solo se pueden vincular con un conector, mientras un conector se puede vincular con más de un componente y más de un conector.(KICILLOF Universidad de Buenos Aires 2004)

En C2 no existe limitación para los lenguajes de la implementación, aunque normalmente soporta desarrollos en C++, Ada, C# y Java. Hasta el momento existen extensiones de Microsoft Visio para C2, así como puede usarse una herramienta llamada SAAGE la cual requiere Visual J++, COM o ya sea

[5] CSP: Communicating Sequential Process. En español: Comunicación de procesos secuencial.

[6] FDR : Verificador de modelos comercial para CSP.

Rational Rose o DRADEL como *front-ends*[7]. Entre las herramientas independientes de plataforma que dispone C2 está el entorno gráfico ArchStudio implementado en estilo C2. (ERIC DASHOFY Department of Informatics, University of California, Irvine. 2007)

- ***Rapide***

Es un ADL de propósito general que permite modelar interfaces de componentes y la conducta de los mismos. Se centra en obtener prototipos ejecutables en VHDL, C, C++, Ada y Rapide[8] a partir de las especificaciones arquitectónicas donde las deben contener explícitamente reflejadas las propiedades de flujo de información, sincronización, concurrencia y temporización de la arquitectura que representan. El modelo de ejecución adoptado por este ADL está basado en los posets[9] los cuales son identificados a través de patrones de eventos. El objetivo de estos posets es describir las secuencias válidas de eventos que pueden ocurrir en un determinado sistema, los cuales se corresponden con la invocación de las operaciones entre los componentes, estos eventos se dividen en acciones que modelan comunicaciones asíncronas y funciones que modelan comunicaciones síncronas; de esta forma es posible simular las especificaciones realizadas con este ADL, así como verificar si las trazas de eventos generados cumplen determinadas restricciones(MORALES 2009).

Es importante señalar que carece de la definición de conector como parte del lenguaje, y lo que se hace es modelar los sistemas de software como un conjunto de componentes conectados a través del envío de mensajes o eventos, así que la descripción de la arquitectura se lleva a cabo por medio de una lista de componentes interconectados entre sí a través de las operaciones importadas y exportadas en las interfaces de dichos componentes. Una característica significativa es que este presenta un mecanismo de herencia para la reutilización de especificaciones y su adaptación a cambios de los requisitos. La disponibilidad de herramientas de soporte así como su documentación se puede obtener de forma gratuita, por otra parte hasta el momento el conjunto de herramientas se encuentra disponible para Solaris 2.5, SunOS 4.1.3 y Linux [10], dichas herramientas son: **RPDC [11]** y **POV [12].**(MORALES 2009)

[7] Front-Ends :Parte del software que interactúa con los usuarios

[8] (VHDL, C. C++, Ada y Rapide): Todos excepto Rapide (ADL) son lenguajes de programación.

[9] Posets: Conjunto de eventos parcialmente ordenados.

[10] Solaris 2.5, SunOS 4.1.3. y Linux: Todos son Sistemas Operativos de licencia GPL.

- ***Jacal***

Fue desarrollado por Nicolás Kicillof y un grupo de investigación del Departamento de Computación de la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires. El objetivo principal de Jacal es lo que actualmente se denomina "animación" de arquitecturas, que no es más que poder visualizar una simulación de cómo se comportaría en la práctica un sistema basado en la arquitectura que se ha representado ya que la notación principal de Jacal es gráfica. Este ADL posee un conjunto predefinido y extensible de conectores, cada uno con una representación distinta. Cada componente cuenta con puertos (ports) que constituyen su interfaz y a los que pueden adosarse conectores. Jacal ofrece además del nivel de interfaz un nivel de comportamiento en el cual se describe la relación entre las comunicaciones recibidas y enviadas por un componente, usando diagramas de transición de estados con etiquetas en los ejes que corresponden a nombres de puertos por los que se espera o se envía un mensaje. Las herramientas que actualmente están disponible para editar y animar arquitecturas en Jacal son una aplicación Win32 que no requiere instalación, basta con copiar el archivo ejecutable para comenzar a usarla. (MORALES 2009)

- ***Acme***

Fue desarrollado por Robert Monroe y David Garlan en la CMU, específicamente en la Escuela de Ciencias de la Computación de dicha universidad. Acme se define como una herramienta capaz de soportar el mapeo de especificaciones arquitectónicas entre diferentes ADLs, es decir, es un lenguaje de intercambio de arquitecturas. (KICILLOF Universidad de Buenos Aires 2004)

La estructura de este ADL se define utilizando siete tipos de entidades: componentes, conectores, sistemas, puertos, roles, representaciones y mapas de representación. Acme es un lenguaje genérico utilizado para describir arquitecturas de software y familias de arquitecturas. El mismo proporciona construcciones para describir la arquitectura como sistemas gráficos de componentes interactuando mediante los conectores, cuenta además con un mecanismo de representación de descomposición jerárquica de los componentes y conectores en subsistemas y una manera de describir familias y tipos de

[11] RPDC: Compilador de código fuente *RAPIDE*, el cual reporta errores de sintaxis y semánticos. Genera un ejecutable que al correr produce un archivo "*.log* "que es un registro de los eventos producidos durante la ejecución.

[12] POV: Las siglas significan Partial Order Viewer y permite observar en forma gráfica la traza de eventos producidos por los ejecutables.

elementos en términos de estas construcciones. No se puede dejar de mencionar también que su facilidad de definir diferentes tipos de componentes, conectores y familias de estilos permite que se ajuste a la taxonomía de la arquitectura de Cedrux. (KICILLOF Universidad de Buenos Aires 2004)

Acme soporta una variedad de *front-ends* de carácter gráfico, que se componen primero por el AcmeStudio el cual está basado en el IDE Eclipse, y tiene soporte tanto en Windows como en Linux en su versión 3.5.2 que será la usada en este trabajo. Susceptible de ser configurado para soportar visualizaciones específicas de estilos e invocación de herramientas auxiliares, además cuenta con una biblioteca llamada AcmeLib la cual define un conjunto de clases para manipular representaciones arquitectónicas Acme. Su código se encuentra disponible tanto en C++ como en Java, y puede ser invocada por lo tanto desde cualquier lenguaje la plataforma clásica de Microsoft o desde el framework de .NET. En el caso de java requiere JRE[13] que permite la ejecución de programas java sobre todas las plataformas soportadas. AcmeStudio permite definir todos los conceptos del ADL Acme de forma correcta, es una herramienta fácil de usar y entender ya que proporciona una ayuda muy sustancial y además existe documentación para esta herramienta que se puede descargar de forma gratuita en su página oficial: http://acme.able.cs.cmu.edu/acmeweb/ . Acme también tiene un ambiente diseñado en ISI [14] que usa el editor de PowerPoint para manipulación gráfica acoplado con analizadores que reaccionan a cambios de una representación DCOM[15] de los elementos arquitectónicos y de sus propiedades asociadas. Otro ambiente grafico es el GME (del inglés *Metaprogrammable Graphical Model Editor*), un ambiente de múltiples vistas que, al ser meta-programable[16], se puede configurar para cubrir una rica variedad de formalismos visuales de diferentes dominios.(KICILLOF Universidad de Buenos Aires 2004)

[13] JRE: Java Runtime Environment en español significa Entorno en Tiempo de Ejecución Java.

[14] ISI: Editor de diseño visual.

[15] DCOM: En español Modelo de Objetos de Componentes Distribuidos que es una tecnología COM(Modelo de Objetos de Componentes).

[16] Meta-programable: Programas que escriben o manipulan otros programas (o a sí mismos) como datos, o que hacen en tiempo de compilación parte del trabajo que, de otra forma, se haría en tiempo de ejecución.

Luego de haber realizado el estudio de los diferentes ADLs que presentan herramientas gráficas se concluye que de ellos el que se empleará en el desarrollo de la herramienta será el Acme. Se decidió optar por este ADL por las ventajas que presenta, siendo estas de gran conveniencia para el equipo de desarrollo. Se identifica como característica más importante que está basado en el IDE Eclipse, lo cual posibilita una mejor implementación de la solución. Además de presentar gran facilidad para adaptarse a la taxonomía de Cedrux.

1.3. Verticalizaciones de Cedrux

Como se mencionó en la introducción del trabajo, el sistema Cedrux está concebido para ser verticalizado. Un *software* puede ser verticalizado de varias maneras. El simple hecho de particularizar las funcionalidades de acuerdo con las características y las necesidades de un cliente o un tipo de cliente específico, puede verse como una verticalización. Sin embargo este proceso puede ocurrir a varios niveles de abstracción. Por ejemplo, se pueden parametrizar funcionalidades a nivel de aplicación. Esto quiere decir que se puede implementar un mecanismo mediante el cual se establezcan configuraciones iniciales del sistema. Estas configuraciones iniciales estarían de acuerdo con los intereses y la *forma de hacer* de los clientes. El sistema Cedrux cuenta con un mecanismo como el descrito anteriormente, como es el caso del módulo de Configuración General. Otra forma de verticalizar un software puede estar dada por la particularización teniendo en cuenta elementos más técnicos. Es decir, desarrollar las funcionalidades en una tecnología definida o solicitada por el usuario, o incluir paquetes específicos también a solicitud de los mismos.

Este último tipo de verticalización por lo general conlleva a realizar modificaciones en la arquitectura del sistema, por lo que puede generar implicaciones mayores.

De acuerdo con la definición de verticalización dada por Manuel Baraza en (BARAZA 2009) : "*verticalizar un software, es hacer de este un sistema más accesible, es decir, adaptarlo más y/o mejor a la capacidad de los usuarios, a la capacidad de entendimiento para quien consulta la aplicación, y capacidad de entendimiento y de ejecución para quien consulta y edita.*", además de lo explicado en los párrafos anteriores, se entiende en este contexto la verticalización como sigue:

Verticalizar es la adición, eliminación o modificación de módulos y las relaciones entre estos con el fin de adaptar el sistema a las necesidades o solicitudes de un cliente. Esto se puede traducir en adición, eliminación o modificación de componentes y sus conexiones (dadas por las dependencias entre ellos) para empaquetar un conjunto de las funcionalidades originales del sistema además de otras que pudieran ser incluidas.

Para llevar a cabo el proceso descrito, se deben realizar variaciones en la estructura de componentes del sistema Cedrux. Las variaciones arquitectónicas que se tratan en este trabajo están dadas por las siguientes operaciones sobre la estructura de componentes:

- Creación de uno o varios componentes (adición de componentes)
- Destrucción de uno o varios componentes (eliminación de componentes)
- Vinculación de componentes (crear enlace entre componentes: crea dependencia)
- Desvinculación de componentes (destruir enlace entre componentes: elimina dependencia)

1.3.1. Arquitectura de Cedrux

La arquitectura de Cedrux está descrita mediante nueve (9) vistas arquitectónicas. Este trabajo se centra en la vista de sistema, específicamente en las vistas de integración y la de componentes. Estas brindan (entre otros) los siguientes elementos (FERNÁNDEZ. 2011):

Vista de Integración: Encargada de los procesos de integración a nivel de sistema.

- Integración interna (entre componentes de un mismo subsistema[17]).
- Integración externa (entre distintos subsistemas).

Vista de Componentes: Encargada de las definiciones de la taxonomía de componentes.

- Definiciones de los tipos de componentes, especificación de sus características, así como la composición interna de cada uno de estos componentes.

La definición de los tipos de componentes está dada por dos clasificaciones: a) atendiendo a la estructura de empaquetamiento y b) según la función dentro del sistema.(FERNÁNDEZ. 2011) En este trabajo se

[17] Subsistema: Es un conjunto de componentes y sus relaciones que se encuentran estructuralmente y funcionalmente, dentro de un sistema mayor.

tiene en cuenta solamente la primera clasificación porque es la que está asociada directamente con la estructura de componentes y no con el comportamiento o función en el sistema de los mismos.

De acuerdo con esa clasificación, existen dos tipos de componentes:

1. *Componente simple*: abstracción de parte o la totalidad de determinadas áreas funcionales, reglas del negocio o del sistema, definiciones tecnológicas implementadas o flujos definidos como parte del proceso de configuración del sistema, es la menor unidad existente en la estructura de empaquetamiento del sistema, autónomo en su funcionamiento y capaz de implementar interfaces o contratos de comunicación. (FERNÁNDEZ. 2011)
2. *Componente contenedor*: En el marco de este trabajo se amplía la definición dada por (FERNÁNDEZ. 2011) como sigue: Estructura formada por un conjunto de componentes simples y/o otros componentes contenedores que interactúan entre ellos, cuya agrupación se lleva a cabo para dar respuesta a un área funcional o parte de la misma, que como área tiene razón de existencia, independencia funcional e interacción con otros elementos.

La comunicación entre estos componentes se realiza mediante un conector propio de la arquitectura en la que se basa Cedrux. Este conector se conoce como ioc y tiene dos variantes. La primera variante se usa para la comunicación entre componentes que no tienen los mismos componentes *ancestros,* es decir, no están incluidos, en su nivel jerárquico, dentro de un mismo componente. Se conoce como ioc-externo. La segunda variante se usa para la comunicación entre componentes que no cumplen el requisito anterior. Es decir, descienden de un mismo componente en algún nivel de la jerarquía a la que pertenecen.

Cada uno de estos componentes tiene características que permiten realizar análisis sobre ellos con el fin de tomar decisiones. Estas características son las siguientes: (FERNÁNDEZ. 2011)

- Cantidad de requisitos funcionales
- Complejidad promedio de los requisitos funcionales
- Cantidad de restricciones de diseño
- Complejidad promedio de las restricciones de diseño
- Cantidad de servicios que brinda el componente
- Cantidad de componentes dependientes del mismo

Estas características influyen en la implicación o el peso que tienen los componentes dentro de la configuración o estructura de componentes. A partir de ellas se pueden tomar decisiones asociadas con la priorización de los componentes para el desarrollo de los mismos, el proceso de soporte y, (entre otras) el proceso de verticalizaciones.

1.3.2. Análisis del epígrafe

A partir de la contextualización de las verticalizaciones en Cedrux y la breve descripción de las vistas arquitectónicas en las que se incide, se realiza un análisis para exponer de forma breve las pretensiones de este trabajo.

A partir de las operaciones que se realizan sobre la estructura de componentes, pueden cambiar los valores de las propiedades de los componentes que forman dicha estructura. Esto afecta de manera directa el comportamiento de indicadores que se tienen en cuenta para estimar el peso que tienen los mismos dentro de dicha estructura. A continuación se muestran estos indicadores y la fórmula para calcularlos a partir de las propiedades de los componentes: (FERNÁNDEZ. 2011)

Tamaño del componente:

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

Donde:

TC: Tamaño del componente

Kr: Cantidad de requisitos funcionales

Krd: Cantidad de restricciones del diseño.

Ks: Cantidad de servicios que brinda el componente

Complejidad de componente:

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

Donde:

Coc: Complejidad del Componente

Cr: Complejidad promedio de los requisitos funcionales

Crd: Complejidad promedio de las restricciones del diseño

TC: Tamaño del componente

Criticidad del Componente

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

Donde:

Crc: Criticidad del Componente

Kcd : Cantidad de componentes dependientes directos

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

Pretensiones de la herramienta a implementar:

1. Automatizar el cálculo de las fórmulas anteriores a partir de la modelación de la estructura de componentes en Acme con AcmeStudio.
2. Proponer una fórmula para estimar la implicación de los resultados de calcular cada una de las fórmulas anteriores sobre una estructura de componentes dada. Esta fórmula definiría la Factibilidad de dicha variación al compararla con otras.

3. Automatizar la modelación de variaciones a una estructura de componentes en AcmeStudio.
    a. Mantener la relación entre la variación y la estructura de componentes base, es decir, de la cual la primera es una variación.
    b. Dar la posibilidad al usuario de seleccionar los componentes de la estructura de componentes base hacia la variación.
4. Realizar comparaciones de Factibilidad de variaciones a partir de una selección de estas.
5. Mostrar reportes que tributen a tomar decisiones de las mencionadas con anterioridad en este mismo epígrafe.

En el resto de este capítulo se define qué se va a utilizar (herramientas, técnicas, metodología) para cumplimentar las pretensiones anteriores.

1.4. Herramientas a utilizar

1.4.1. Entorno de Desarrollo Integrado

Entorno de Desarrollo Integrado *o Integrated Development Environment* (IDE), es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien, puede utilizarse para varios.(EDWIN 2006)

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación. Consiste en un editor de código, un compilador, un depurador y un constructor de Interfaz Gráfica de Usuario o *Graphical User Interface* (GUI). Muchos de los IDEs modernos llevan integrados además, un navegador de clases, un inspector de objetos y diagramas de herencia de clases para ser usados en el desarrollo orientado a objetos(EDWIN 2006)

Eclipse

En noviembre de 1998, una de las transnacionales más grandes del mundo IBM (del inglés *International Business Machine*), otorgó la tarea a uno de sus grupos de desarrollo de software denominado *Object Technology International* (OTI), de crear una plataforma de desarrollo común para confeccionar sus

productos, basados en el lenguaje de programación Java. Fue así como surgió el denominado proyecto Eclipse (OBREGÓN 2008).

Eclipse es una plataforma porque no es una aplicación acabada de por sí, pero está diseñada para ser extendida indefinidamente con herramientas sofisticadas. Es un IDE porque provee herramientas para administrar áreas de trabajo (o *workspaces* en inglés), para construir, lanzar y depurar aplicaciones, para compartir artefactos con un equipo y versionar el código fuente. Eclipse es abierto porque su diseño le permite ser extendido fácilmente por terceras partes. Es apropiado para todo, ya que ha sido utilizado exitosamente para construir ambientes para un amplio rango de temas, como el desarrollo en lenguaje de programación Java, Servicios Web, certámenes de programación de juegos. Eclipse no es para nada en particular, pues no se enfoca en ningún dominio específico (JOHN ARTHORNE Addison-Wesley Professional 2004)

Como se muestra en este concepto, Eclipse es un producto singular por la versatilidad que ofrece. Lo que le permite a Eclipse tener estas características es la poderosa arquitectura basada en plugins con el cual fue diseñado.

1.4.2. Arquitectura de Eclipse

Cuando la plataforma Eclipse fue donada a la comunidad *open source* por IBM[18] en el año 2001, la noticia no fue notable en el mundo por la enorme suma de dinero que IBM declaró que se había gastado en el desarrollo de la misma (40 millones de dólares); sino debido al resultado obtenido con esta millonaria inversión: un producto inigualable por su arquitectura madura, bien diseñada y extensible basada en plugins (GALLARDO 2002)

Con la excepción de un pequeño núcleo o *kernel* denominado *Platform Runtime*, todos los demás componentes que conforman la plataforma Eclipse son plugins. Entre ellos encontramos el *Workbench*, el *Workspace*, el Help y el *Team* como se muestra en la figura 1.1. (OBREGÓN 2008)

.

[18] IBM: Maquina de Negocios Internacional.

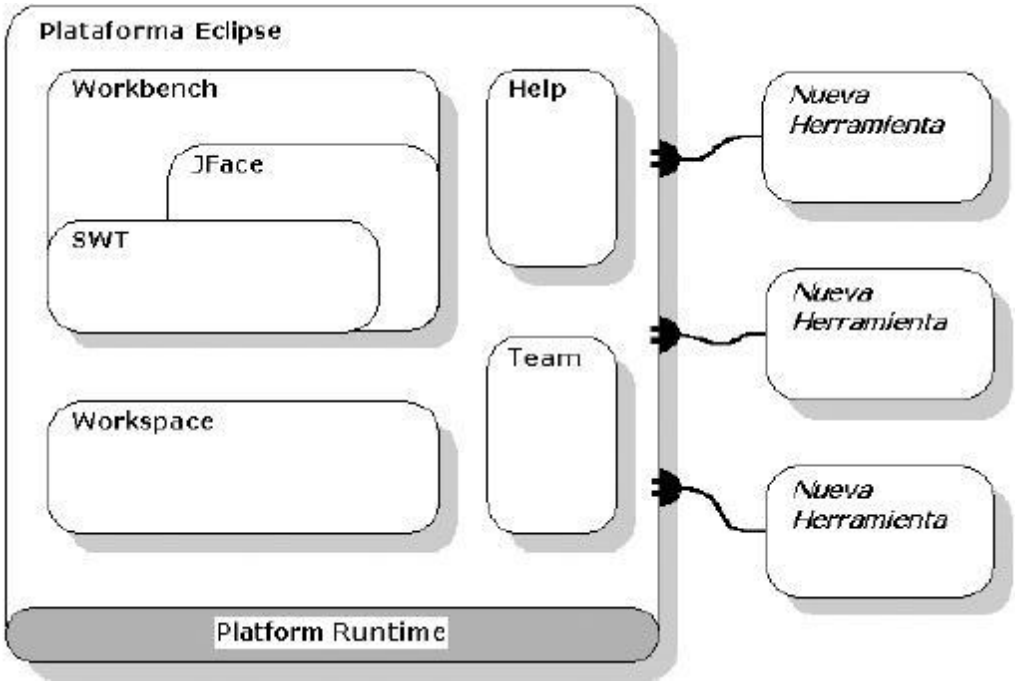


Figura 3: Arquitectura basada en plugins.(OBREGÓN 2008)

El *Workbench* es el componente de la plataforma responsable de la presentación y la coordinación de la interfaz gráfica de usuario (GUI), además de que ofrece la estructura básica de la misma para Eclipse y los *puntos de extensión* necesarios para extender las funcionalidades de la GUI por otros plugins. Es el encargado, de mostrar todos los menús, editores de textos, vistas y perspectivas de Eclipse. El *Workspace* se caracteriza por ser el responsable de manejar los recursos de los usuarios, los cuales se organizan en uno o más proyectos a un nivel superior. Cada proyecto corresponde a un subdirectorio del directorio de trabajo de Eclipse (en inglés *workspace*) y en él se pueden encontrar carpetas y ficheros. También puede proporcionar el código para configurar los recursos del proyecto según sea necesario y como en el caso del *Workbench,* contiene además los puntos de extensión para que otros plugins interactúen con los recursos de los usuarios. (OBREGÓN 2008)

Los *plugins* que anteriormente mencionamos son los componentes esenciales de la plataforma. Pero además de ellos, la misma contiene integrados otros como el plugin *Team* y el *Help*. El plugin *Team* facilita el uso del control de versiones (o gestión de configuración) de los sistemas para administrar los recursos en los proyectos, además de que define el flujo de trabajo necesario para salvar y recuperar datos de un repositorio. El componente *Help*, no es más que un sistema de documentación, extensible

como la misma plataforma Eclipse, el cual mediante el uso de herramientas provistas por la plataforma, se le puede añadir documentación en formato HTML y usando XML puede definir una estructura de navegación.(OBREGÓN 2008)

Además de los plugins ya referidos se pueden encontrar otros que aunque no son parte integral de la plataforma, se encuentran integrados en el Eclipse SDK, ellos son el *Java Development Tooling* (JDT) y el Ambiente de Desarrollo de Plugin o en inglés Plug-in *Development Environment* (PDE), que como describe su nombre, es un plugin que facilita el desarrollo de plugins para de esta forma extender las funcionalidades de la misma plataforma. (OBREGÓN 2008)

1.4.3. Plugins

Se puede imaginar un gigantesco rompecabezas, al principio algunas de las piezas cuando se comienza a armar el mismo, ya han sido conectadas (esto conformaría el núcleo en torno al cual el resto del rompecabezas se construye). Si se percata, los bordes de las piezas están especialmente conformados para que cada una pueda conectarse a otra perfectamente.

Si Eclipse fuera el rompecabezas, los plugins serían las piezas que lo conforman. Un *plugin* es la unidad más atómica y extensible de Eclipse. Está escrito puramente en el lenguaje de programación Java y típicamente consiste en código Java empaquetado en un archivo de Java (JAR), con algunos archivos de solo lectura, y otros recursos como imágenes, catálogo de mensajes, etc. (OBREGÓN 2008).

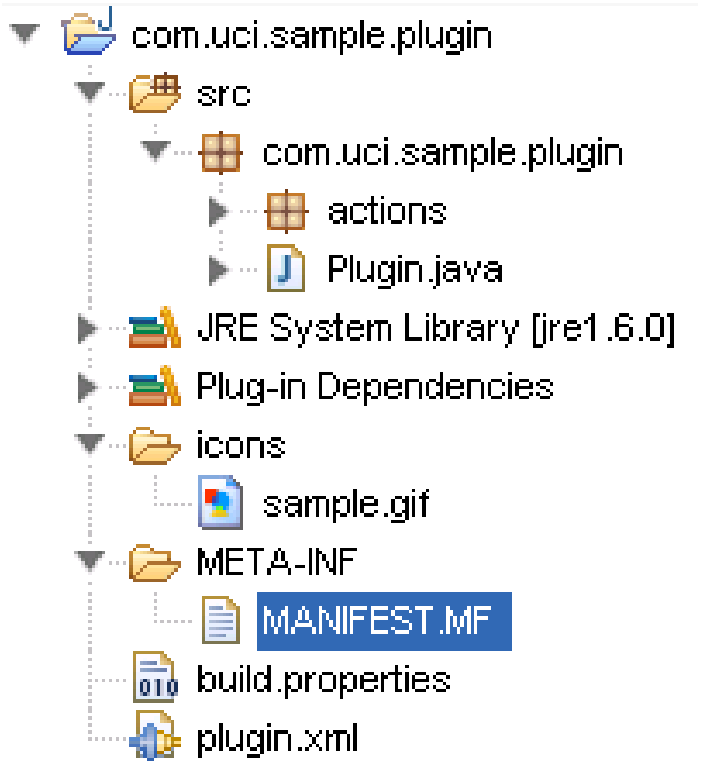


Figura 4: Estructura simple de un plugin de Eclipse. (OBREGÓN 2008)

Anatomía de un plugin

Cada plugin posee un manifiesto del plugin (en inglés, *plugin manifest*) en el cual se describe su interconexión con otros plugin, 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 (Ver en la figura 1-2). El *MANIFEST.MF* y el plugin.xml. A continuación se hace una breve descripción de estos:

MANIFEST.MF: este fichero es generado dentro del directorio *META-INF* cuando es creado un proyecto plugin. Es un manifiesto del paquete OSGi en donde se describen las dependencias que posee el plugin con otros plugins en tiempo de ejecución y sus atributos como son: el nombre que posee el plugin, el proveedor, la versión del mismo entre otros. (MARIÑO 2010)

plugin.xml: se encuentra en la misma raíz del proyecto y consiste en un archivo de formato XML que describe todas las extensiones realizadas por el plugin y declara además los puntos de extensión que el propio plugin ha definido.

Además de estos elementos en el directorio de un plugin se encuentran otros archivos y carpetas como se muestra a continuación:

icons: directorio en el cual se encuentran las imágenes o iconos que va a utilizar el plugin. Usualmente son archivos en formato GIF.

Plugin.properties: contiene cadenas de caracteres externalizadas que son referenciadas en el plugin.xml.

about.html: archivo en formato HTML que es utilizado para mostrar informaciones relacionadas con licencias (MARIÑO 2010).

En este trabajo se utilizará el Eclipse en su versión 3.2.

1.5. Selección de las tecnologías de desarrollo.

Luego de un análisis de la plataforma de desarrollo Eclipse así como su arquitectura basada en plugins y los lenguajes de descripción de arquitecturas, se procede a hacer una selección de los métodos, tecnologías y herramientas a usar para desarrollar la propuesta de solución. Los epígrafes que se enuncian a continuación proveen las características más significativas que llevaron a la selección de estas herramientas.

1.5.1. Entorno de desarrollo integrado Eclipse.

Eclipse es una plataforma de herramienta universal, un IDE de código abierto y extensible, para todo y nada en particular. Su valor real proviene de sus plugins que se integran a la plataforma permitiendo el uso de herramientas aprovechando las facilidades y comodidades que provee Eclipse, ejemplo de ello se tienen herramientas para desarrollo Web, herramientas para el diseño de clases, herramientas para el control de versiones, entre otras. La plataforma es muy flexible y extensible.

Eclipse cuenta con un gran corrector de errores que, aparte de identificar las palabras reservadas del lenguaje también puede detectar, y marcar sobre el código de un programa, los lugares donde se pueden producir errores de compilación o de sintaxis. Posee un excelente sistema para completar códigos

facilitando el trabajo al programador. Eclipse incorpora una herramienta para realizar automáticamente el formateo del código de acuerdo a unos criterios preestablecidos.

La compilación es una tarea que se lanza automáticamente al guardar los cambios realizados en el código. Por esta razón es prácticamente innecesario controlar manualmente la compilación de los proyectos. También integra un cliente para el sistema de gestión de versiones CVS. Asimismo, a través de plugins libremente disponibles es posible añadir otros como *Subversion y Git*.

JDT

El Java *Develop Tooling* (JDT) es uno de los principales subproyectos de Eclipse y es el que permite que la plataforma Eclipse se transforme en un potente IDE de Java, uno de los más reconocidos del mundo. Es el conjunto de plugin más avanzado y elaborado que posee Eclipse. El JDT asiste a los usuarios en los procesos de escribir, compilar, probar, depurar y editar programas escritos en el lenguaje de programación Java, incluyendo los plugins de Eclipse. El JDT está estructurado en tres componentes principales que contienen los paquetes del API:

JDT Core: define el núcleo de los elementos Java y provee clases útiles para manipular archivos de extensión .class y el modelo de elementos Java.

JDT UI: contiene la interfaz de usuario que provee el IDE.

JDT Debug: brinda un conjunto de clases que permiten correr y depurar código Java.

PDE

El Plugin *Development Enviroment* (PDE) es un ambiente de desarrollo para plugin, que provee Eclipse y uno de los principales subproyectos del mismo, junto con JDT. Está compuesto por un conjunto de herramientas que cubren todo el ciclo de vida completo del desarrollo de un plugin. Facilita todo el proceso de crear, desarrollar, probar, depurar, construir y desplegar plugins de Eclipse.

1.5.2. Lenguaje de Programación Java

Java, es un lenguaje de programación simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico. Trabaja con sus

datos como objetos y con interfaces a esos objetos. “Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo. Realiza verificaciones en busca de problemas, tanto en tiempo de compilación, como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo”.

“Una característica distintiva de Java, es su capacidad multiplataforma. Esto lo consigue, no solo a nivel de código fuente, sino también a nivel de código compilado. Java podrá ejecutarse en cualquier sistema operativo que tenga una máquina virtual Java compatible”.

1.5.3. Máquina Virtual de Java

La Máquina Virtual Java (JVM) es el entorno en el que se ejecutan los programas Java, su misión principal es la de garantizar la portabilidad de las aplicaciones Java. Su función es amplia pero podemos señalar que entre sus tareas está proporcionar la vista de un nivel de abstracción superior, permitiendo la independencia de la plataforma. En este trabajo se atizará como máquina virtual de java la JDK 1.6.

1.6. Metodologías estudiadas.

La metodología de desarrollo es un aspecto de gran importancia para el proceso de desarrollo del software, manifiesta más trascendencia si el proyecto a realizar es complejo aunque todo desarrollo de software es riesgoso y difícil de controlar. Es necesario llevar una metodología adecuada, con el propósito de evitar que tanto los clientes como los desarrolladores queden insatisfechos con el resultado obtenido.(ROBERTH G. FIGUEROA 2007)

EXtreme Programing

La programación extrema o *EXtreme Programing* (XP) es una metodología reciente en el desarrollo de software. La filosofía de XP es satisfacer al completo las necesidades del cliente, por eso lo integra como una parte más del equipo de desarrollo. Fue inicialmente creada para el desarrollo de aplicaciones donde el cliente no sabe muy bien lo que quiere, lo que provoca un cambio constante en los requisitos que debe cumplir la aplicación.

XP está diseñada para el desarrollo de aplicaciones que requieran un grupo de programadores pequeño, donde la comunicación sea más factible que en grupos grandes de desarrollo. La comunicación es un

punto importante y debe realizarse entre los programadores, los jefes de proyecto y los clientes. Las características esenciales de esta metodología son las siguientes:

Comunicación: Los programadores están en constante comunicación con los clientes para satisfacer sus requisitos y responder rápidamente a los cambios de los mismos. Muchos problemas que surgen en los proyectos se deben a que después de concretar los requisitos que debe cumplir el programa no hay una revisión de los mismos, pudiendo dejar olvidados puntos importantes.

Simplicidad: Codificación y diseños simples y claros. Muchos diseños son tan complicados que cuando se quieren ampliar resulta imposible hacerlo, por lo que se tienen que desechar y partir de cero.

Realimentación (Feedback): Mediante la realimentación se ofrece al cliente la posibilidad de conseguir un sistema apto a sus necesidades, ya que se le va mostrando el proyecto a tiempo para poder ser cambiado y poder retroceder a una fase anterior para rediseñarlo a su gusto.

(NÉSTOR DÍAZ VALENCIA 2009)

OpenUP

Es una versión más ágil de lo que es el *Rational Unified Process* (RUP), es un proceso unificado que aplica propuestas iterativas e incrementales dentro del ciclo de vida, tratando de ser manejable en relación con RUP. Plantea que se debe tener un software ya funcional, o lo que es lo mismo, un proyecto ejecutable en poco tiempo. Plantea que se debe utilizar sólo los procesos que sean necesarios, sin demasiados artefactos y sobre todo que el proyecto debe acoplarse a las necesidades del usuario, pudiendo ser éste modificado, mejorado y extendido. *OpenUP* tiene dos ventajas importantes, este tipo de método disminuye los riesgos y además puede utilizarse tanto en proyectos pequeños como en proyectos grandes, aunque está concebida para proyectos pequeños. Si se maneja con cuidado y con profesionalismo se puede desarrollar un software de gran calidad, a pesar de que se le diseñe en poco tiempo y con poca documentación. Se utiliza preferentemente en proyectos pequeños, dígase proyectos de 3 a 6 personas. OpenUP contiene las características esenciales de RUP, que incluye el desarrollo iterativo de casos de uso, además de proporcionar un acercamiento a la arquitectura central del sistema. El resultado es un proceso mucho más simple que sigue fielmente los principios de RUP. OpenUP se organiza en dos dimensiones diferentes: Método y Proceso.

***Método*:** Los roles, las tareas y los artefactos están definidos, sin tener en cuenta cómo son aplicados en el ciclo de vida del proyecto.

***Proceso*:** Es cuando los elementos del proceso son aplicados en el sentido conductual, donde el mismo brinde los roles, las tareas y los artefactos. Pueden crearse ciclos de vida diferentes para proyectos diferentes. (NÉSTOR DÍAZ VALENCIA 2009)

Análisis del estudio de las metodologías

Después de analizar estas dos metodologías se ha decidido utilizar OpenUP.

Se ha seleccionado como metodología de desarrollo OpenUP por ser una metodología ágil que aplica un enfoque iterativo e incremental dentro de su estructura del ciclo de vida y que puede adaptarse para desarrollar diversos tipos de proyectos. OpenUP 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. Permite detectar errores tempranos a través de un ciclo iterativo, evita la elaboración de documentación, diagramas e iteraciones innecesarios requeridos en la metodología RUP. Por ser una metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas.

OpenUP cuenta con 4 fases de desarrollo como se muestra en la figura (NÉSTOR DÍAZ VALENCIA 2009)

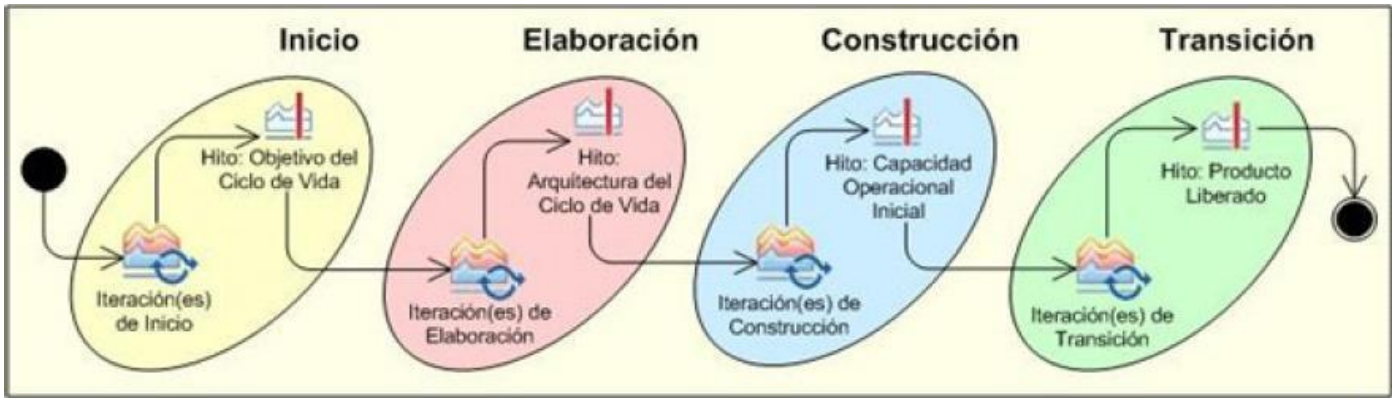


Figura 5: Fases de la metodología de desarrollo OpenUP.

Inicio: Primera de las 4 fases en el ciclo de vida del proyecto, en esta fase se realiza un entendimiento de los propósitos y objetivos, obteniendo suficiente información para confirmar qué debe hacerse en el proyecto. El objetivo de ésta fase es capturar las necesidades de los *stakeholders* [19] en los objetivos del ciclo de vida para el proyecto.

[19] Stakeholders: Partes interesadas ya sea inversionistas, clientes, gobiernos, comunidades locales o empleados vinculados a una empresa u organización.

Elaboración: Es el segundo de las 4 fases del ciclo de vida del OpenUP donde se tratan los riesgos significativos para la arquitectura. El propósito de esta fase es establecer la base de la elaboración de la arquitectura del sistema.

Construcción: Esta fase está enfocada al diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de esta fase es completar el desarrollo del sistema basado en la Arquitectura definida.

Transición: Es la última fase, su propósito es asegurar que el sistema sea entregado a los usuarios, y evalúa la funcionalidad y rendimiento del último entregable de la fase de construcción.

Conclusiones del capítulo.

En este capítulo se estableció un marco teórico donde se mencionaron las características fundamentales, respecto al modelado arquitectónico, de los principales ADL que poseen una herramienta gráfica y que pudieran servir de ayuda para el modelado de la arquitectura de Cedrux. Se realizó un análisis de estos ADLs y tomándose partido sobre los resultados obtenidos se sentaron las bases para el desarrollo de la solución. Finalizando con la adecuada selección de las herramientas, tecnologías y metodologías necesarias para implementar el sistema propuesto.

CAPITULO 2 PROPUESTA DE SOLUCIÓN

INTRODUCCIÒN

En el presente capítulo se presenta el diseño de la herramienta y se especifican las funcionalidades que se implementan como extensión de AcmeStudio. Se exponen los artefactos necesarios que fueron creados siguiendo la metodología OpenUP a lo largo del desarrollo del plugin. Además, se describen las características y estructura del plugin que se desarrollará, luego de identificar las principales funcionalidades de este.

1.7. Propuesta del Sistema

El plugin a desarrollar tiene como objetivo extender la herramienta AcmeStudio para proveer a esta con funcionalidades para realizar comparaciones entre distintas variantes de un modelo arquitectónico y así agilizar el proceso de verticalizaciòn de Cedrux. Se automatizarán las funcionalidades de: crear, modificar y evaluar escenarios arquitectónicos, así como guardar reporte.

1.8. Metodología y artefactos

La metodología definida como guía en el proceso de desarrollo de la herramienta, es OpenUp, debido a que es una metodología ágil, apropiada para proyectos pequeños y de bajos recursos. Provee un conjunto simplificado de artefactos, roles, tareas y guías de trabajo. Evita la elaboración de documentación, diagramas e iteraciones innecesarios requeridos en la metodología RUP.

La metodología OpenUP propone una serie de actividades que sirven de guía durante el ciclo de vida del desarrollo de un software. Las siguientes actividades se realizan teniendo en cuenta esta metodología:

- Modelación del dominio.
- Captura y refinamiento de requisitos.
- Diseño de la solución.
- Implementación de la herramienta.
- Validación de la solución.

1.9. Artefactos generados

La metodología OpenUp, especifica diferentes roles a los cuales se les asocia la creación de determinados artefactos como se muestra en la Figura 6**.**

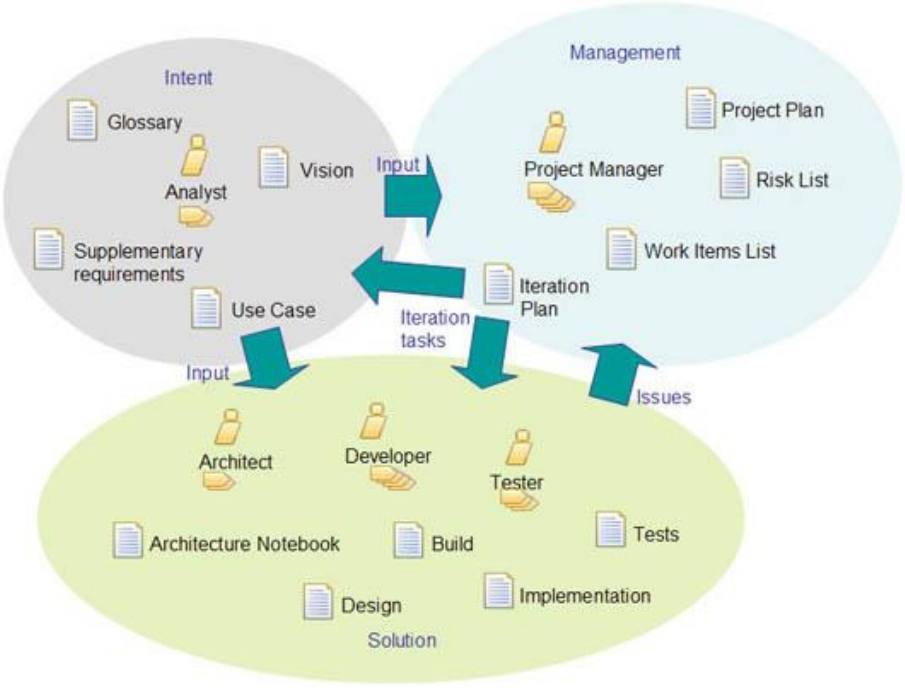


Figura 6. Roles definidos y artefactos asociados.

La primera actividad que propone esta metodología es la modelación del dominio y como artefacto un glosario de términos, el cual no se incluye en este trabajo pues a todas las palabras en inglés o de difícil entendimiento se le especifica su significado en una nota al pie.

1.9.1. Documento Visión

Este es uno de los artefactos que propone OpenUp. Tiene como objetivo lograr concordancia entre todos los *stakeholders* de los objetivos del ciclo de vida del proyecto. Proporciona una visión completa del sistema de software en desarrollo y los soportes del contrato entre los clientes y la organización de desarrollo. Cada proyecto necesita una fuente para capturar todas las expectativas de los *stakeholders* y es escrito desde la perspectiva de los clientes, enfocado en las características esenciales del sistema y niveles aceptables de calidad.(Tecnología y Synergix 2008). **Ver Anexo 1**

1.9.2. Requisitos suplementarios

Las especificaciones suplementarias capturan todos los requisitos que no pueden ser expresados en los casos de uso. Contienen todos los requisitos funcionales genéricos que no están asociados con ningún caso de uso específico (ROJAS 2011). Se puede entender por tanto, que los requisitos suplementarios son características no funcionales de alto nivel a implementarse en el proyecto en términos de necesidades de usuarios finales. Lo anterior permite hacer una analogía entre este término y los requisitos no funcionales.

Seguidamente, se presentan especificaciones de requisitos suplementarios, organizados por tipo de requerimiento:

Usabilidad:

- Las funcionalidades tendrán nombres sugerentes de acuerdo con su objetivo.
- La información será exhibida en lenguaje español.

Escalabilidad:

- Contará con un diseño que permita agregar nuevas funcionalidades sin necesidad de cambios drásticos en los elementos del dominio y estructuras de datos.
- Se integrará con la herramienta AcmeStudio a través de un *plugin*.

Interfaz de Usuario:

- Estará acorde al diseño visual de la herramienta AcmeStudio.

Software:

- JDK 1.5 o superior.
- Eclipse 3.2 o superior.
- Bibliotecas:
    - org.acmestudio.*
    - org.eclipse.resources.*

1.9.3. Requisitos funcionales

Para el levantamiento de los requisitos funcionales del *software*, se utilizaron varias técnicas: entrevistas, talleres y tormenta de ideas.

Entrevistas: Son prácticamente inevitables en cualquier desarrollo ya que es una de las formas de comunicación más natural entre personas. La entrevista es buena para obtener conocimientos acerca del trabajo actual en el dominio dado y los problemas presentes.

Talleres: Se usan para cubrir casos donde los usuarios y los desarrolladores se reúnen para diseñar o perfilar el sistema futuro. El término “trabajo cooperativo” se aplica a esta técnica.

Tormenta de ideas: La tormenta de ideas es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios. Las sesiones suelen estar formadas por un número de cuatro a diez participantes, uno de los cuales es el *facilitador* de la sesión, encargado más de comenzar la sesión que de controlarla. Esta técnica tiene ventajas como: a) es muy fácil aprender y b) requiere poca organización, por lo que se aplica generalmente al principio, cuando las ideas son muy difusas. Es recomendable, por esto último, combinarla luego con otras técnicas como las explicadas con anterioridad.

Se realizaron tormentas de ideas, combinadas con entrevistas y talleres con la participación de los autores y el tutor de este trabajo que a la vez es el cliente, donde se acordó el levantamiento de los requisitos que a continuación se listan, producto de la aplicación de las técnicas anteriores por parte del equipo de desarrollo.

- Crear variación arquitectónica
    - Clonar configuración
    - Crear variación a partir de selección
- Determinar grado de factibilidad
- Determinar variaciones más factibles
- Mostrar reporte
    - Promedio de criticidad de componentes dada una configuración
    - Promedio de complejidad de componentes dada una configuración

  - Promedio de tamaño de componentes dada una configuración
  - Listado ordenado de variaciones por promedio de criticidad de componentes
  - Listado ordenado de variaciones por promedio de complejidad de componentes
  - Listado ordenado de variaciones por promedio de tamaño de componentes
- Guardar reporte

Paquete de requisito	**Descripción**
Crear variación arquitectónica	Crea una variación a partir de una configuración arquitectónica. Tiene dos (2) flujos alternativos en dependencia de los datos que se desea tomar de la configuración original. Los requisitos que contiene este paquete tienen una complejidad media, dado que toman los datos de un diseño realizado en la herramienta AcmeStudio y deben mantener una sincronización en todo momento de los datos que se analizan y dicho modelo.
Determinar grado de factibilidad	Calcula el grado de factibilidad de cada configuración a partir de la fórmula definida para este indicador. Dicha fórmula se muestra más adelante en este documento. Debe permitir seleccionar las configuraciones que se desea incluir en dicho análisis.
Determinar variaciones más factibles	Calcula o toma el valor de factibilidad de cada configuración seleccionada y devuelve las que tienen un mejor valor de dicho indicador. En caso de existir más de una con grado de factibilidad igual, y dicho grado es el mejor (de acuerdo con la fórmula definida), devuelve todas las que cumplen con el

	mismo.
Mostrar reporte	Muestra el reporte seleccionado. Existen varios tipos de estos.
Guardar reporte	Permite guardar en formato pdf el reporte seleccionado.

En la tabla 1 se muestra una breve descripción de los paquetes de requisitos definidos anteriormente.

Tabla 1: Descripción de paquetes de requisitos.

Modelo de casos de uso del sistema

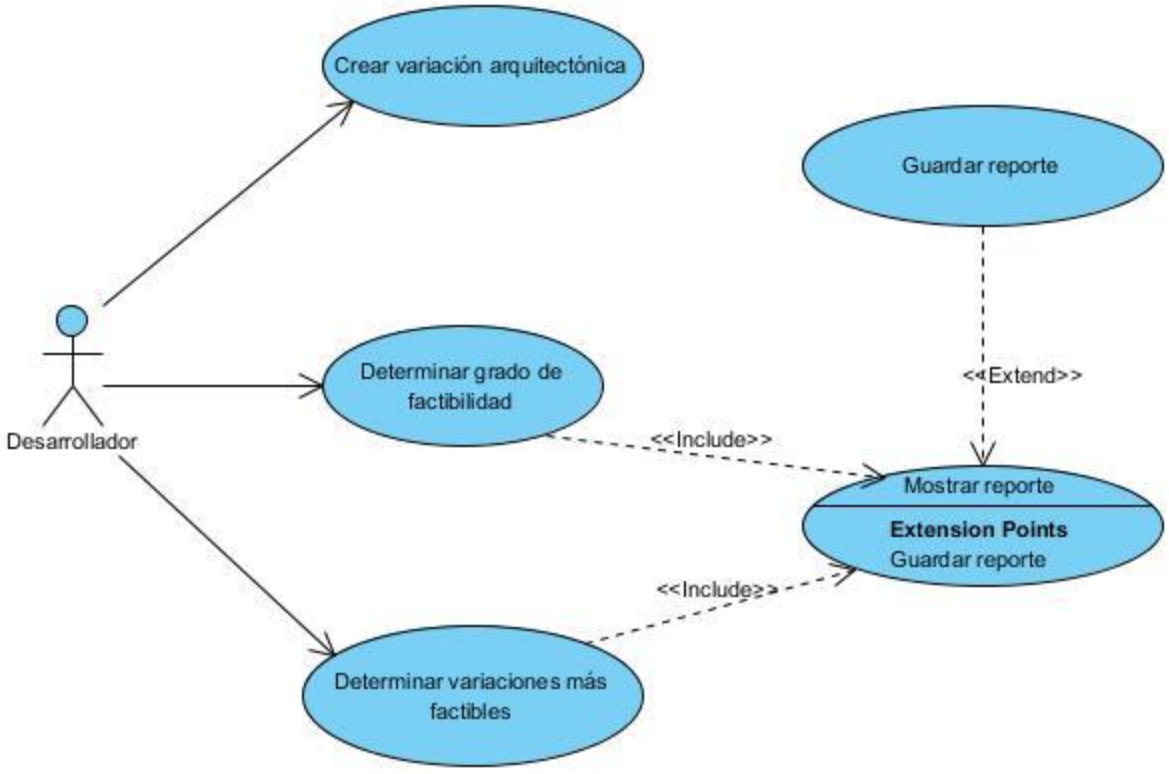


Figura 7: Modelo de casos de uso del sistema

El Actor Desarrollador representa al arquitecto de *software* que interactúa con el sistema. Este puede iniciar cualquiera de los casos de uso: Crear variación arquitectónica, Determinar grado de factibilidad, Determinar variaciones más factibles; y al realizarse cualquiera de los dos últimos se realiza el caso de uso Mostrar Reporte. El Desarrollador decide si iniciar o no el caso de uso Guardar Reporte.

Tipos de Relaciones

Se utiliza <<*extends*>> cuando se presenta una variación del comportamiento normal, e <<*include*>> cuando se repite un comportamiento en dos casos de uso y queremos evitar dicha repetición. Una relación << *extends*>> es una relación de dependencia entre dos casos de uso que denota que un caso de uso es una especialización de otro. En una relación de este tipo el actor que lleve a cabo el caso de uso base puede realizar o no sus extensiones (JACOBSON 1992) por lo tanto en este caso de uso el desarrollador luego de Mostrar Reporte puede realizar o no el caso de uso Guardar Reporte. En una relación <<*include*>> el actor que realiza el caso de uso base también realiza el caso de uso incluido, por tanto el Desarrollador luego de realizar cualquiera de los casos de uso Determinar grado de factibilidad o Determinar variaciones más factibles realizará luego Mostrar Reporte.

En la figura 7 se pueden observar las relaciones que existen entre los casos de uso.

La especificación de cada uno de estos casos de uso puede encontrarse en el **Anexo 2.**

1.9.4. Prototipo de interfaz de usuario

La interfaz de usuario se basa en la interfaz de AcmeStudio al cual se le añadirá, con la implementación de la herramienta, el menú: Control de Escenarios como se muestra en la siguiente figura.

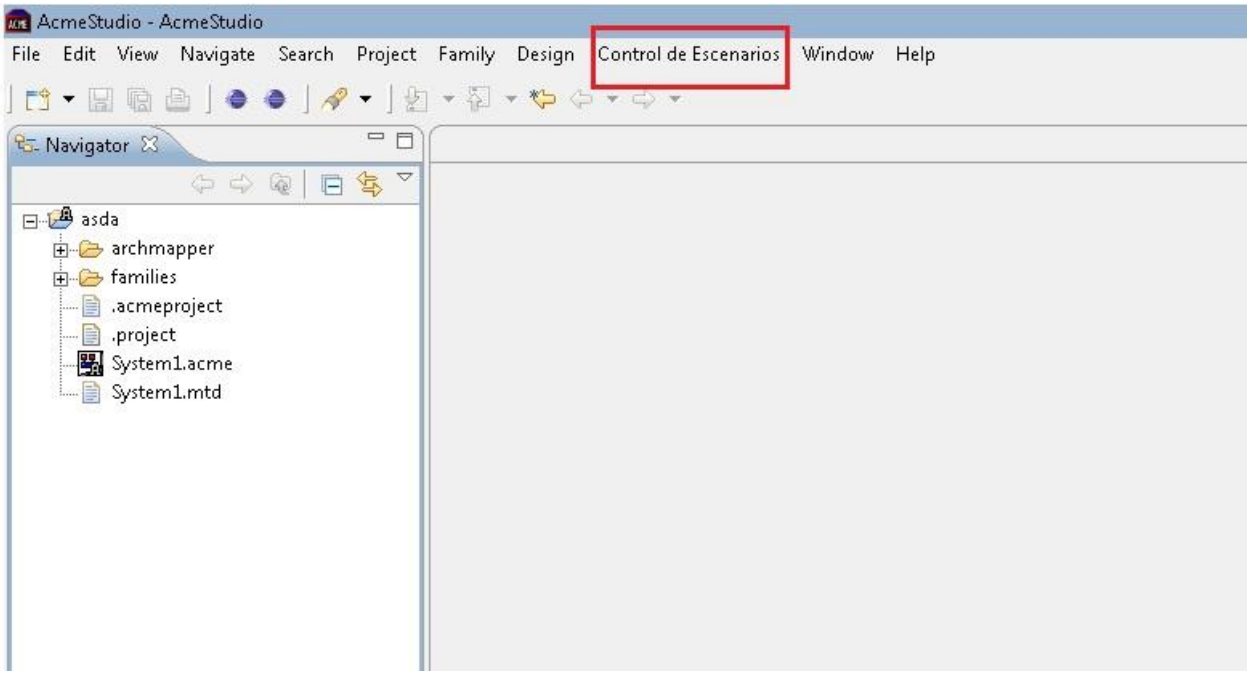


Figura 8: Menú "Control de Escenarios".

Este menú contendrá los submenús: "Crear Variación Arquitectónica, "Determinar variaciones más factibles" , "Determinar grado de factibilidad", "Promedio de criticidad de componentes"," Promedio de complejidad de componentes", "Promedio de tamaño de componentes", "Listado por promedio de criticidad de componentes", ‘‘Listado por promedio de complejidad de componentes", ‘‘Listado por promedio de tamaño de componentes".

Si se hace clic en el submenú "Crear Variación Arquitectónica" (Anexo 5) (Figura 9) y se seleccionan algunos elementos (componentes, conectores, roles, etc.) dentro del sistema o escenario que se encuentra activo se creará en una nueva pestaña o editor un nuevo escenario con estos elementos seleccionados; en caso de que no se señalen se realiza la copia del escenario completo al nuevo editor.

Al hacer clic en el segundo submenú (Anexo 5) (Figura 10) y seleccionar o activar un sistema se determina el promedio de criticidad de esta configuración y se muestra un reporte con la criticidad de cada componente y su respectivo promedio.(Anexo 5) (Figura 11)

Si se selecciona uno o varios sistemas y se hace clic en el submenú Listado ordenado de promedio de criticidad (Anexo 5) (Figura 12) se genera un reporte que contiene un listado de los sistemas seleccionados ordenado de mayor a menor por su criticidad. (Anexo 5) (Figura 13)

Al hacer clic en el submenú Determinar grado de factibilidad (Anexo 5) (Figura 14) después de seleccionar uno o varios sistemas se realiza un análisis de los estos y se muestra en un reporte ordenado de mayor a menor su grado de factibilidad (Anexo 5) (Figura 15).

Si se selecciona el submenú: “Determinar variaciones más factibles” (Anexo 5) (Figura 16) después de haberse seleccionado dos o más sistemas se realiza una comparación entre los grados de factibilidad de estos y se muestra en un reporte él o los más factibles, o sea el de mayor grado de factibilidad (Anexo 5) (Figura 17).

Todos los reportes antes mencionados cuentan con dos botones: Guardar Reporte y Cancelar, el primero de estos botones como su nombre lo indica brinda la opción de guardar el reporte en formato PDF, y al hacer clic en él se podrá especificar en una nueva interfaz denominada: “Guardar Reporte” la dirección donde se desea guardar (Anexo 5) (Figura 18); el segundo botón ofrece la opción de cerrar el reporte.

1.9.5. Diagrama de paquetes

Los diagramas de paquetes se usan para reflejar la organización de paquetes y sus elementos. Los usos más comunes de estos son; para organizar diagramas de casos de uso y diagramas de clases, además de ser grandes contenedores de clases. Muestra como un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema. Este artefacto describe la realización de las funcionalidades requeridas del sistema y sirve como una abstracción del código fuente, es decir, describe los elementos del sistema para que puedan ser examinados y entendidos de una forma que no es posible leyendo el código fuente.

En la Ilustración 3 se muestra el diagrama de paquetes que contiene de manera general las clases del diseño. Es necesario en este apartado resaltar que entre los requisitos no funcionales de la herramienta a desarrollar, se define que esta debe ser escalable e integrarse con la herramienta AcmeStudio. Estos paquetes que se presentan en la Ilustración que sigue, responden a dichos requisitos. Lo anterior se puede explicar de la manera siguiente: al comenzar el desarrollo de esta herramienta ya existe una estructura de paquetes base, con un conjunto de clases que soportan las funcionalidades fundamentales de la misma. La herramienta debe ser desarrollada teniendo en cuenta dicha estructura, organizando las clases de acuerdo a la responsabilidad de cada una en cada uno de los paquetes que aquí se muestran.

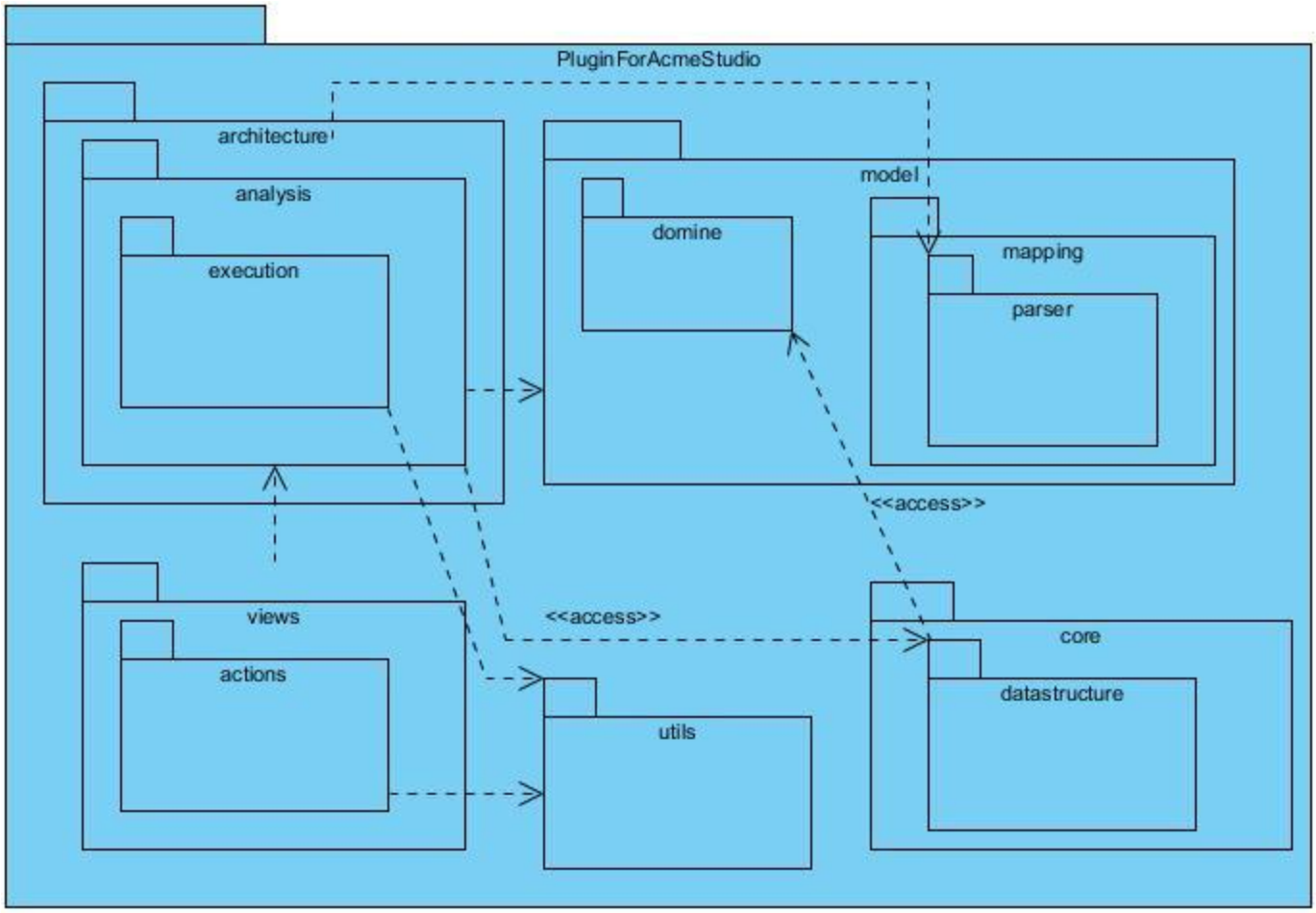


Figura 15: Diagrama de paquetes

Para un mejor entendimiento, se considera oportuno dar una paronímica sobre la relación que existe entre los paquetes utilizados en la solución.

Execution: contiene las clases *ComponentProerties* y *ExecutionControler,* las cuales se encargan de propiciarle al paquete *Parser* las propiedades de los componentes para que este se encargue de parsearlas, para así poder realizar los cálculos sobre la estructura de componentes.

Actions**:** contiene las clases *CompareAction*, *CloneAction* y *Factibility* siendo estas las que muestran gráficamente al usuario toda la estructura y el trabajo con los escenarios, luego de que el paquete

Execution le brinde las propiedades de los componentes para facilitar la vista de los resultados de las operaciones.

***Parser*:** contiene la clase *Parser,* esta se responsabiliza en obtener la configuración específica de los elementos que conforman una variación arquitectónica.

Utils: posee las clases *Obtainer y BasicOperations* las cuales se encargan de obtener la ubicación física del Acme File. Siendo función principal de este paquete brindarle dicha ubicación al paquete *Execution* para así poder obtener las propiedades de los componentes que conforman ese Acme File. También le brinda esta ubicación al paquete *Actions* para que este realice sus cálculos sobre ese Acme File en específico.

1.9.6. Diagrama de clases

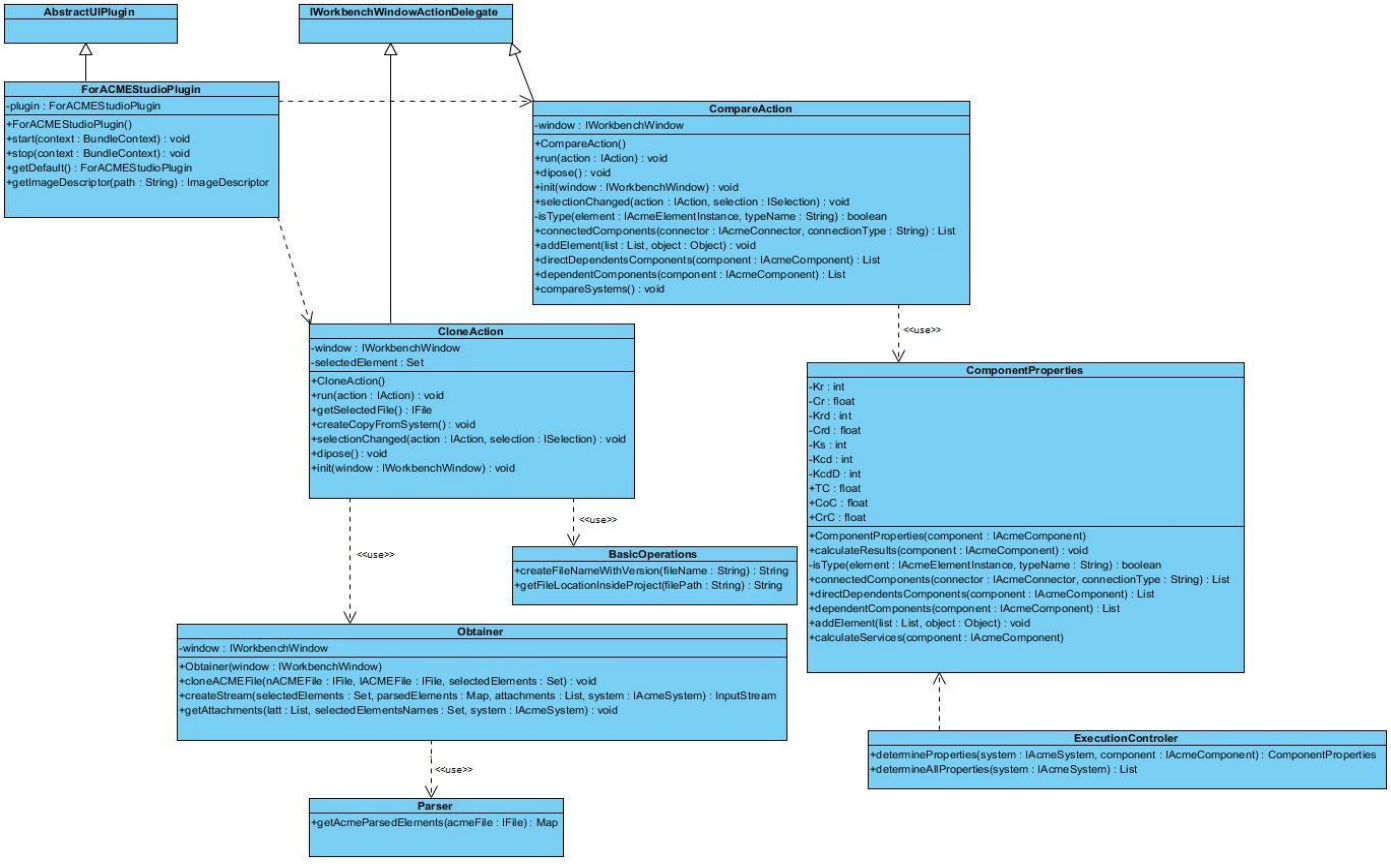


Figura 16: Diagrama de Clases del Sistema.

En el diagrama de clases se puede observar la relación que existe entre las clases anteriormente mencionadas en el diagrama de paquetes. Se considera oportuno realizar una breve explicación para un mejor entendimiento del diagrama.

Como se observa en la figura las relaciones en las que una clase usa a la otra están representadas por líneas discontinuas. Las relaciones de dependencia directa se representan con una línea continua. Las clases a las que no se les definen métodos son propias del lenguaje utilizado en el desarrollo de la aplicación, lo cual se hizo necesario durante la implementación para la utilización de las facilidades que brinda el lenguaje.

1.10. Implementación

Haciendo énfasis en uno de los objetivos específicos de la solución se propone a continuación la fórmula para determinar la factibilidad de una variación arquitectónica. Ayudando así a la toma de decisiones sobre la estructura de componentes.

$$F= \frac{0.3*TC + 0.4*CoC + 0.5*CrC}{3}$$

Para mayor aceptación en cuanto al valor de factibilidad se decidió ponderar las propiedades que presentan los componentes de una variación arquitectónica. Identificándose como propiedad más importante a tener en cuenta la Criticidad de los componentes, como segundo indicador a tener en cuenta se identificó la Complejidad de estos componentes y por ultimo su Tamaño. De esta forma el cálculo de la factibilidad tiene como principal criterio a evaluar la criticidad, en segundo lugar la complejidad y en tercer lugar el tamaño de los componentes.

Conclusiones del capítulo.

Siguiendo la metodología OpenUP a través de una óptima planificación y diseño guiados por los artefactos generados, se ha logrado obtener el plugin *PluginForAcmeStudio*. A lo largo del capítulo se resume el trabajo realizado donde se exponen las principales características y funcionalidades del plugin, además de

los paquetes de clases que lo conforman y su estructura. Como resultado, el plugin *PluginForAcmeStudio* cumple con todos los requisitos y funcionalidades definidas.

CAPITULO 3 VALIDACIÓN Y PRUEBAS

INTRODUCCIÓN

Este capítulo se centra en la actividad de realización de las pruebas y validación de la aplicación, con el objetivo de comprobar las funcionalidades del plugin, verificando en todos los casos que los resultados de las pruebas sean los esperados. Se expondrán además las métricas aplicadas al diseño propuesto en el capítulo anterior, en aras de garantizar la calidad de dicha solución.

3.1. Validación del diseño

El desarrollo de software es algo muy complejo y la medida de su calidad real no es automatizable. Por esta razón la aplicación de métricas de calidad, Tamaño Operacional de la Clase (TOC) y Relaciones entre Clases (RC), para evaluar el diseño orientado a objeto posee gran importancia. A continuación, se presenta un estudio que brinda un esquema sencillo de implementar y que a la vez cubre los principales atributos de calidad de software, siendo esto la principal razón de la concepción de las métricas (PRESSMAN 2002)

Los atributos de calidad que se abarcan según (Herrera, y otros, 2007), se pueden consultar en el **Anexo1, Tabla 1.**

Tamaño Operacional de la Clase

Esta métrica está relacionada con el número de métodos asignados a una clase, teniendo en cuenta los atributos que se presentan en la **Tabla 1.**

Atributo que afecta	Modo en que lo afecta
Responsabilidad	Un aumento de TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Un aumento de TOC implica un aumento de la complejidad de implementación de la clase.

Reutilización	Un aumento del TOC implica una disminución en el grado de reutilización de la clase.

Tabla 1. Tamaño Operacional de la Clase.

Para ver instrumento de medición de la métrica TOC, consultar en el **Anexo 3, Tabla 2** y para ver tabla de resultado al aplicar el instrumento de medición de la métrica TOC, consultar **Anexo 3, Tabla 3.**

Luego de realizado un análisis profundo en cada una de las clases utilizadas, la evaluación de la métrica TOC, reflejó los siguientes resultados:

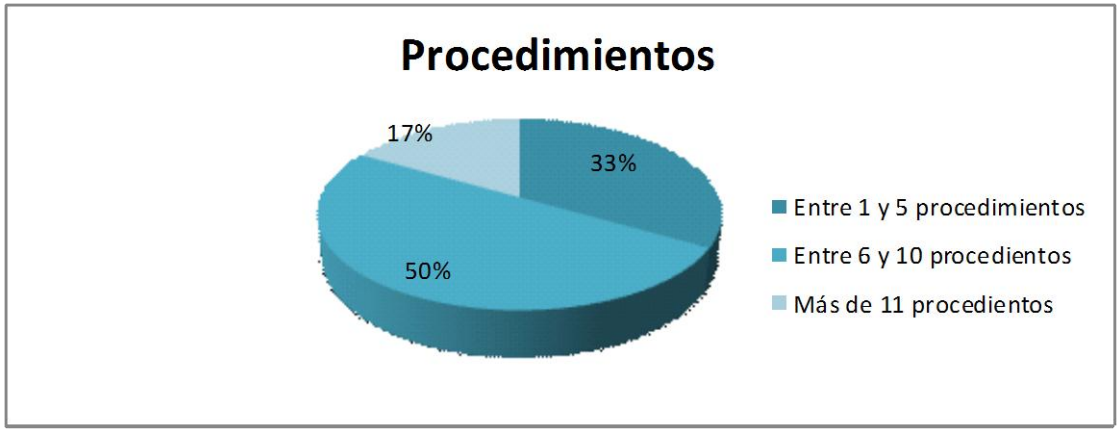


Figura 1. Representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

Figura 2. Representación de la incidencia de los resultados de la métrica TOC para el atributo Responsabilidad.

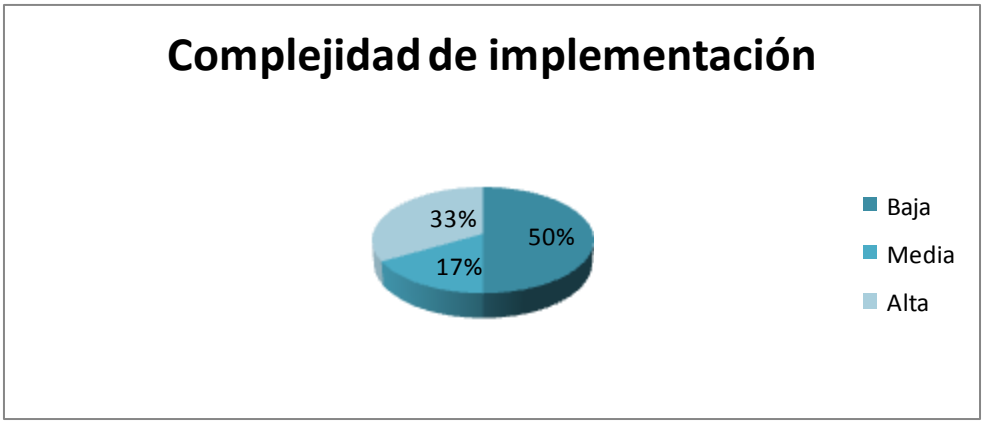


Figura 3. Representación de la incidencia de los resultados de la métrica TOC para el atributo Complejidad de implementación

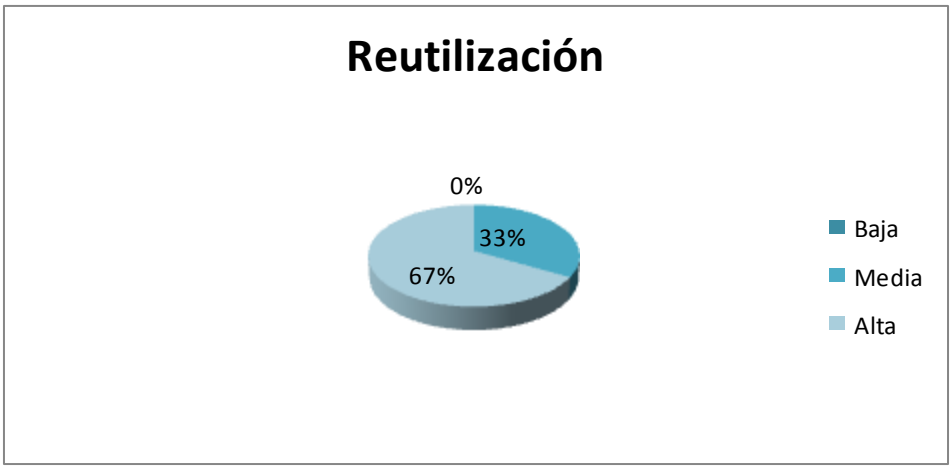


Figura 4. Representación de la incidencia de los resultados de la métrica TOC para el atributo Reutilización.

Al aplicar la métrica TOC, se determinó que la aplicación consta de 6 clases y un total de 41 procedimientos, con promedio de 7 procedimientos aproximadamente por clases. La **Figura 1** muestra el promedio de procedimientos por clases donde se puede observar que el 33% de las clases poseen a lo

sumo 5 procedimientos. La **Figura 2**, **Figura 3** y **Figura 4,** muestran los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC en los diferentes atributos de calidad. Estos resultados demuestran que la mayoría de las clases presentan un 50% de nivel medio de Responsabilidad, un 50% de dichas clases presentan baja Complejidad de Implementación y un 67% de alta Reutilización, lo que brinda como resultado que el diseño propuesto para la herramienta a desarrollar, tiene una calidad aceptable, siendo éste lo más simple posible, permitiendo una sencilla implementación y realización de pruebas con mayor facilidad.

Relaciones entre clases

Esta métrica está dada por el número de relaciones de uso de una clase con otras, teniendo en cuenta los atributos que se presenta en la **Tabla 2**.

Atributo que afecta	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del acoplamiento de la clase.
Complejidad del mantenimiento	Un aumento de RC implica un aumento de la complejidad del mantenimiento de la clase.
Cantidad de pruebas	Un aumento del RC implica aumento de la cantidad de pruebas de unidad necesarias para probar una clase
Reutilización	Un aumento del RC implica una disminución en el grado de reutilización de la clase.

Tabla 2. Relaciones entre clases (RC).

Para ver el instrumento de medición de la métrica RC, consultar el **Anexo 4, Tabla 2** y para ver tabla de resultado al aplicar el instrumento de medición de la métrica RC, consultar **Anexo 4, Tabla 3.**

Luego de realizado un análisis profundo en cada una de las clases utilizadas y sus dependencias, la evaluación de la métrica RC, reflejó los siguientes resultados:

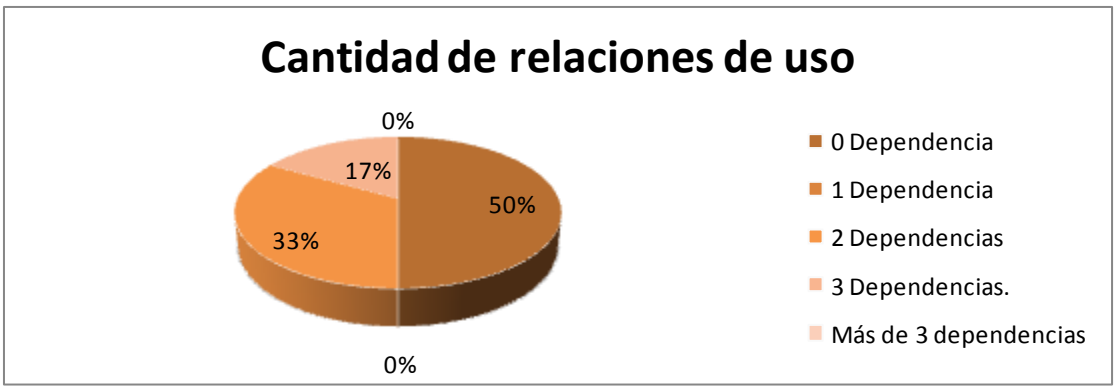


Figura 5. Representación de los resultados obtenidos en el instrumento, agrupados en los intervalos definidos.

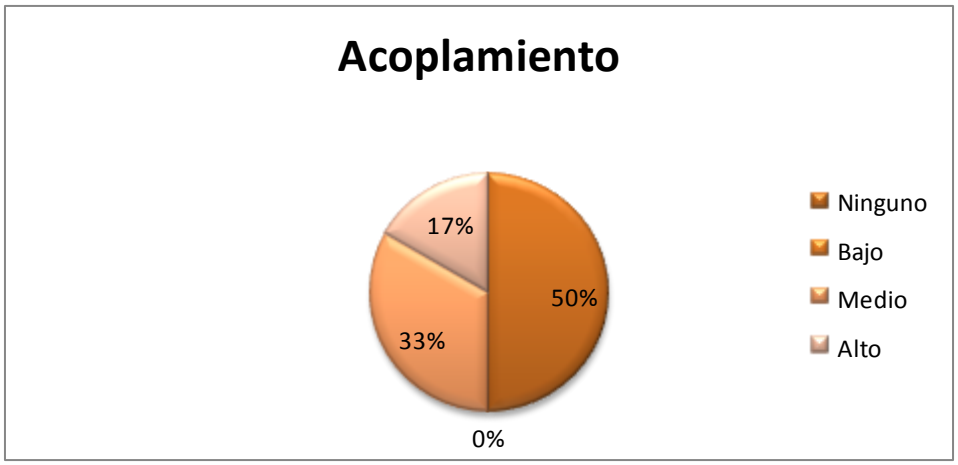


Figura 6. Representación de la incidencia de los resultados de la evaluación de la métrica Relaciones entre clases del atributo Acoplamiento.

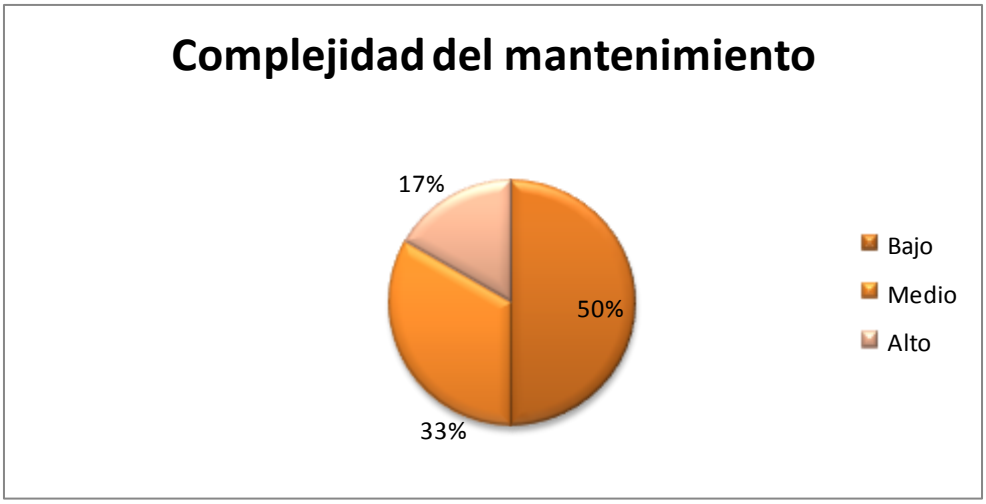


Figura 7. Representación de la incidencia de los resultados de la evaluación de la métrica Relaciones entre clases del atributo Complejidad del mantenimiento.

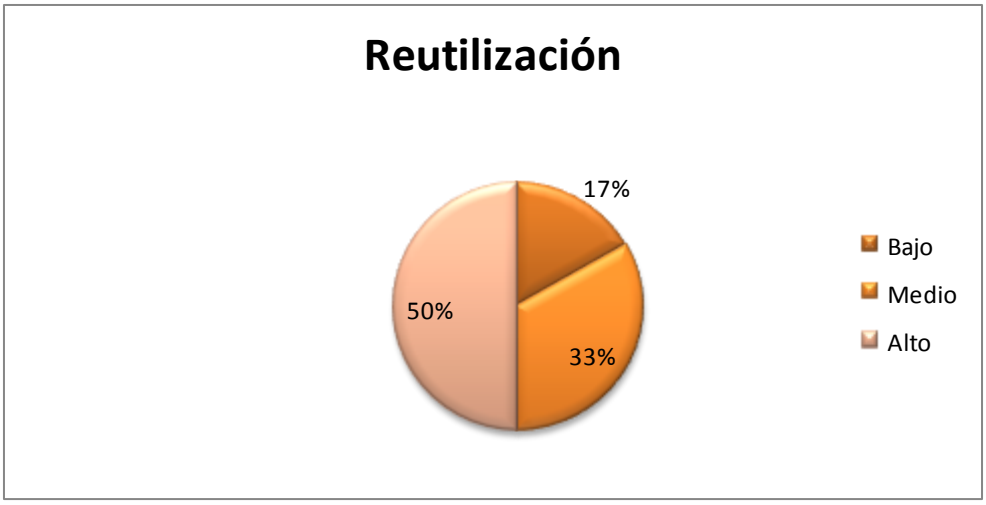


Figura 8. Representación de la incidencia de los resultados de la evaluación de la métrica Relaciones entre clases del atributo Reutilización.

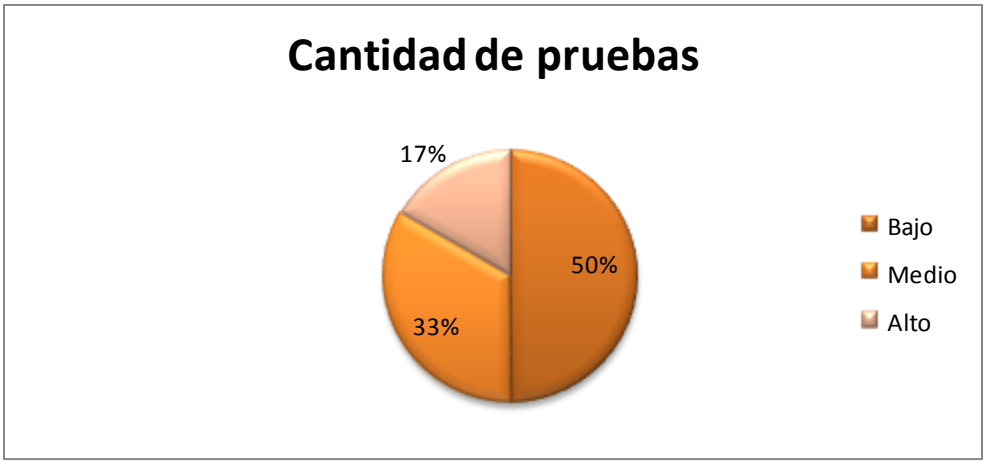


Figura 9. Representación de la incidencia de los resultados de la evaluación de la métrica Relaciones entre clases del atributo Cantidad de pruebas.

Al aplicar la métrica RC, se determinó que la aplicación consta de 6 clases y un total de 7 dependencias, con promedio de 1 dependencia aproximadamente por clase, ver **Anexo 2, Tabla 3**. La **Figura 5** , muestra el promedio de dependencias por cantidad de clases, donde se puede observar que el 50% de las clases no poseen dependencias. La **Figura 6** , **Figura 7**, **Figura 8** y **Figura 9**, muestran los resultados obtenidos, en los diferentes atributos de calidad. Estos resultados demuestran que un 50% del total de clases no presenta acoplamiento, un 50% de baja complejidad de mantenimiento, un 50% de alta reutilización y un 50% de baja cantidad de pruebas, esto trae como consecuencia que el diseño propuesto para la herramienta a desarrollar, tiene una calidad aceptable, lo que favorece a una alta reutilización de las clases.

Al aplicar las métricas a las clases del diseño definidas para dar cumplimiento a la herramienta a desarrollar, se cumple que las métricas TOC y RC, favorecen las restricciones para la aceptación de las clases.

3.2. Pruebas de software

Después de concluida la actividad de implementación, el software debe ser probado, para detectar y corregir errores, antes de entregar el producto. Las pruebas aplicadas a la herramienta, son las pruebas de caja negra, también denominadas prueba de comportamiento, las cuales se centran en los requisitos funcionales del software según (PRESSMAN 2002). Estas pruebas se llevan a cabo sobre la interfaz del

software. O sea los casos de pruebas pretenden demostrar que las funciones de software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto. La misión de estas pruebas es detectar errores como:

- Funciones incorrectas o ausentes.
- Errores de Interfaz
- Errores de Rendimiento

Las pruebas tienen gran importancia en el desarrollo de un software, ya que mediante éstas se pueden detectar y corregir errores tempranamente. Garantizan que antes de entregar el producto final al cliente el software cumpla con todos los requerimientos y se hayan corregido todos los errores. Tienen el objetivo de encontrar el mayor número posible de errores. Las pruebas deben conducirse sistemáticamente y los casos de prueba deben diseñarse utilizando técnicas definidas. Un caso de prueba especifica cómo probar un Caso de Uso o un escenario específico del mismo. Incluye la verificación del resultado de la interacción entre actores y el sistema.(MARIÑO 2010)

CASO DE PRUEBA CREAR VARIACIÓN ARQUITECTÓNICA

Identificador de la HU a probar
CPCNE01- Crear Variación Arquitectónica.
Descripción de la Funcionalidad:
La funcionalidad "Crear Variación Arquitectónica" permite añadir nuevos escenarios arquitectónicos y clonar existentes.
Condiciones de Ejecución:
El AcmeStudio se encuentra abierto.
Flujo Central:
El usuario abre el sistema.
El sistema muestra la vista de inicio.
El usuario escoge en la barra de herramientas la opción Control de Escenarios el submenú "Crear Nueva Variación Arquitectónica".
El sistema crea una nueva variación arquitectónica en un nuevo editor.

Clases Válidas	Resultado Esperado	Resultado de la Prueba	Observaciones

El usuario presiona la opción Crear Variación Arquitectónica y se encuentra un escenario activo dentro del sistema.	El sistema muestra en un nuevo editor una copia del escenario.	El sistema muestra una copia en un nuevo editor.	
El usuario señala algunos elementos (componentes, conectores, etc.) dentro de la configuración y presiona la opción crear variación arquitectónica.	El sistema muestra en un nuevo editor una nueva variación con los elementos señalados en la configuración.	El sistema muestra un nuevo escenario con los elementos que se señalaron en un nuevo editor.	

CASO DE PRUEBA DETERMINAR GRADO DE FACTIBILIDAD

Identificador de la HU a probar

CPCNE02-Determinar grado de factibilidad.

Descripción de la Funcionalidad:

La funcionalidad "Determinar grado de factibilidad" calcula el grado de factibilidad de una configuración a partir de la fórmula definida para este indicador.

Condiciones de Ejecución:

Se encuentra abierto el AcmeStudio.

Flujo Central:

El usuario abre el sistema.

El sistema muestra la vista de inicio.

El usuario escoge en la barra de herramientas la opción Control de Escenario el submenús “Determinar grado de factibilidad”.

El sistema calcula el grado de factibilidad de una configuración.

Clases Válidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario presiona la opción Determinar grado de factibilidad.	El sistema muestra el grado de factibilidad.	El sistema muestra el grado de factibilidad.	
El usuario selecciona un sistema y presiona el botón aceptar.	El sistema muestra el grado de factibilidad del sistema seleccionado.	El sistema muestra el grado de factibilidad del sistema seleccionado.	

CASO DE PRUEBA DETERMINAR VARIACIONES MÁS FACTIBLES

Identificador de la HU a probar

CPCNE03- Determinar variaciones más factibles.

Descripción de la Funcionalidad:

La funcionalidad "Determinar variaciones más factibles" calcula o toma el valor de factibilidad de cada configuración seleccionada y devuelve las que tienen un mejor valor de dicho indicador.

Condiciones de Ejecución:

Se encuentra abierto el AcmeStudio.

Flujo Central:

El usuario abre el sistema.

El sistema muestra la vista de inicio.

El usuario escoge en la barra de herramientas la opción Control de Escenario el submenú "Determinar variaciones más factibles".

El sistema calcula las variaciones más factibles.

Clases Válidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario presiona la opción Determinar variaciones más factibles.	El sistema muestra un listado ordenado por el grado de factibilidad de los sistemas.	El sistema muestra un listado ordenado por el grado de factibilidad de los sistemas.	
El usuario selecciona dos o más sistemas y presiona el botón aceptar.	El sistema muestra un listado ordenado por el grado de factibilidad.	El sistema muestra un listado ordenado por el grado de factibilidad.	

CASO DE PRUEBA MOSTRAR REPORTE: DETERMINAR VARIACIONES MÁS FACTIBLES

Identificador de la HU a probar

CPCNE04- Determinar variaciones más factibles.

Descripción de la Funcionalidad:

La funcionalidad “Determinar variaciones más factibles” lista las variaciones más factibles.

Condiciones de Ejecución:

Se encuentra abierto el AcmeStudio.

Flujo Central:

El usuario abre el sistema.

El sistema muestra la vista de inicio.

El usuario escoge en la barra de herramientas la opción Control de Escenario y hace clic en el submenú “Determinar variaciones más factibles”

El sistema muestra un reporte.

Clases Válidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario presiona la opción Determinar variaciones más factibles.	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	

CASO DE PRUEBA: DETERMINAR GRADO DE FACTIBILIDAD

Identificador de la HU a probar

CPCNE04- Determinar grado de factibilidad.

Descripción de la Funcionalidad:

La funcionalidad “Determinar grado de factibilidad” devuelve el grado de factibilidad de una configuración.

Condiciones de Ejecución:

Se encuentra abierto el AcmeStudio.

Flujo Central:

El usuario abre el sistema.

El sistema muestra la vista de inicio.

El usuario escoge en la barra de herramientas la opción Control de Escenario y hace clic en el submenú “Determinar grado de factibilidad”

El sistema muestra un reporte.

Clases Válidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario presiona la opción Determinar grado de factibilidad	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	

CASO DE PRUEBA: PROMEDIO DE CRITICIDAD DE COMPONENTES DADA UNA CONFIGURACIÓN

Identificador de la HU a probar

CPCNE04- Promedio de criticidad de componentes dada una configuración.

Descripción de la Funcionalidad:

La funcionalidad “Promedio de criticidad de componentes dada una configuración” lista el promedio de la criticidad de los componentes de una configuración.

Condiciones de Ejecución:

Se encuentra abierto el AcmeStudio.

Flujo Central:

El usuario abre el sistema.

El sistema muestra la vista de inicio.

El usuario escoge en la barra de herramientas la opción Control de Escenario y hace clic en el submenú “Promedio de criticidad de componentes dada una configuración”.

El sistema muestra un reporte.

Clases Válidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario presiona la opción Promedio de criticidad de componentes dada una configuración	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	

CASO DE PRUEBA: PROMEDIO DE COMPLEJIDAD DE COMPONENTES

Identificador de la HU a probar

CPCNE04 Promedio de complejidad de componentes.

Descripción de la Funcionalidad:

La funcionalidad “Promedio de complejidad de componentes” lista el promedio de la complejidad de los componentes de una configuración.

Condiciones de Ejecución:

Se encuentra abierto el AcmeStudio.

Flujo Central:

El usuario abre el sistema.

El sistema muestra la vista de inicio.

El usuario escoge en la barra de herramientas la opción Control de Escenario y hace clic en el submenú “Promedio de complejidad de componentes”.

El sistema muestra un reporte.

Clases Válidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario presiona la opción Promedio de complejidad de componentes	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	

CASO DE PRUEBA: PROMEDIO DE TAMAÑO DE COMPONENTES

Identificador de la HU a probar

CPCNE04 Promedio de tamaño de componentes.

Descripción de la Funcionalidad:

La funcionalidad “Promedio de tamaño de componentes” lista el promedio del tamaño de los componentes de una configuración.

Condiciones de Ejecución:

Se encuentra abierto el AcmeStudio.

Flujo Central:

El usuario abre el sistema.

El sistema muestra la vista de inicio.

El usuario escoge en la barra de herramientas la opción Control de Escenario y hace clic en el submenú “Promedio de tamaño de componentes”.

El sistema muestra un reporte.

Clases Válidas	Resultado Esperado	Resultado de la Prueba	Observaciones

El usuario presiona la opción Promedio de tamaño de componentes	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	

CASO DE PRUEBA: LISTADO POR PROMEDIO DE CRITICIDAD DE COMPONENTES

Identificador de la HU a probar

CPCNE04 Listado por promedio de criticidad de componentes.

Descripción de la Funcionalidad:

La funcionalidad “Listado por promedio de criticidad de componentes” lista el promedio de la criticidad de los componentes de una configuración.

Condiciones de Ejecución:

Se encuentra abierto el AcmeStudio.

Flujo Central:

El usuario abre el sistema.

El sistema muestra la vista de inicio.

El usuario escoge en la barra de herramientas la opción Control de Escenario y hace clic en el submenú “Listado por promedio de criticidad de componentes”.

El sistema muestra un reporte.

Clases Válidas	Resultado Esperado	Resultado de la	Observacio

		Prueba	nes
El usuario presiona la opción Listado por promedio de criticidad de componentes	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	

CASO DE PRUEBA: LISTADO POR PROMEDIO DE COMPLEJIDAD DE COMPONENTES

Identificador de la HU a probar

CPCNE04 Listado por promedio de complejidad de componentes.

Descripción de la Funcionalidad:

La funcionalidad “Listado por promedio de complejidad de componentes” lista el promedio de la complejidad de los componentes de una configuración.

Condiciones de Ejecución:

Se encuentra abierto el AcmeStudio.

Flujo Central:

El usuario abre el sistema.

El sistema muestra la vista de inicio.

El usuario escoge en la barra de herramientas la opción Control de Escenario y hace clic en el submenú “Listado por promedio de complejidad de componentes”.

El sistema muestra un reporte.

Clases Válidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario presiona la opción Listado por promedio de complejidad de componentes	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	

CASO DE PRUEBA: LISTADO POR PROMEDIO DE TAMAÑO DE COMPONENTES

Identificador de la HU a probar

CPCNE04 Listado por promedio de tamaño de componentes.

Descripción de la Funcionalidad:

La funcionalidad “Listado por promedio de tamaño de componentes” lista el promedio del tamaño de los componentes de una configuración.

Condiciones de Ejecución:

Se encuentra abierto el AcmeStudio.

Flujo Central:

El usuario abre el sistema.

El sistema muestra la vista de inicio.

El usuario escoge en la barra de herramientas la opción Control de Escenario y hace clic en el submenú “Listado por promedio de tamaño de componentes”.

El sistema muestra un reporte.

Clases Válidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario presiona la opción Listado por promedio de tamaño de componentes	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	El sistema muestra en una ventana el reporte con los resultados de la operación señalada.	

CASO DE PRUEBA GUARDAR REPORTE

Identificador de la HU a probar

CPCNE05- Guardar reporte.

Descripción de la Funcionalidad:

La funcionalidad "Guardar Reporte" guarda el reporte de evaluación en formato pdf.

Condiciones de Ejecución:

Se encuentra abierto el AcmeStudio.

Flujo Central:

El usuario abre el sistema.

El sistema muestra la vista de inicio.

El usuario escoge en la barra de herramientas la opción Control de Escenario y hace clic en cualquiera de los submenús: "Determinar variaciones más factibles", "Determinar grado de factibilidad", "Promedio de criticidad de componentes dada una configuración"," Promedio de complejidad de componentes",

“Promedio de tamaño de componentes”, “Listado por promedio de criticidad de componentes”, ‘‘Listado por promedio de complejidad de componentes”, ‘‘Listado por promedio de tamaño de componentes”.

El sistema calcula las variaciones más factibles o el grado de factibilidad y muestra la interfaz Reporte.

El usuario hace clic en el botón Guardar Reporte.

El sistema muestra la interfaz Guardar Reporte.

El usuario selecciona la dirección donde desea guardar el reporte y hace clic en el botón guardar.

Clases Válidas	**Resultado Esperado**	**Resultado de la Prueba**	**Observaciones**
El usuario presiona la opción Guardar Reporte de la interfaz Reporte	El sistema muestra la interfaz Guardar Reporte.	El sistema muestra la interfaz Guardar Reporte.	
El usuario selecciona la dirección donde guardará el reporte y hace clic en el botón guardar.	El sistema guarda el reporte, en formato pdf, en la dirección indicada.	El sistema guarda el reporte, en formato pdf, en la dirección indicada.	
El usuario hace clic en el botón Cancelar	El sistema cierra la interfaz Guardar Reporte.		

3.3. Validación de la Solución

Para validar la herramienta se realiza una comparación a la hora de hacer las verticalizaciones de Cedrux con el uso de la herramienta y prescindiendo de esta.

Luego de realizar entrevistas y talleres con desarrolladores y personas documentadas e involucradas con el proceso de verticalizaciòn, se llegó a la conclusión que las verticalizaciones del sistema Cedrux, aunque actualmente se realizan, son un proceso muy engorroso. A continuación se muestra una tabla con los resultados obtenidos.

Indicadores	**Tiempo y esfuerzo**
Modificar el diseño	La realización de este proceso puede demorar horas, convirtiéndose este en un proceso muy abrumador.
Identificar dependencias	Puede demorar algunos días, incluso semanas, siendo muy trabajoso para los desarrolladores realizar este proceso.
Realizar cálculos de complejidad y criticidad	Estos cálculos se realizan, pero sin un profundo análisis de la estructura de componentes. Demoran días en realizarse, por lo que es un proceso muy trabajoso.
Comparar variaciones por criterio de factibilidad.	Este proceso no se realiza actualmente en Cedrux.

En la siguiente tabla se muestran los resultados obtenidos en tiempo y esfuerzo, luego de realizar el proceso de verticalización utilizando el plugin.

Indicadores	**Tiempo y esfuerzo**
Modificar el diseño	Es uno de los procesos más sencillos de realizar y solo tomaría minutos, incluso segundos, dependiendo de la complejidad de la estructura de componentes.

Identificar dependencias	Es realizado prácticamente de forma instantánea, garantizando la identificación del 100 % de las dependencias.
Realizar cálculos de complejidad y criticidad	Se realiza mediante un profundo análisis de la estructura de componentes y solo demora unos segundos, por lo que se convierte en algo muy sencillo para el usuario.
Comparar variaciones por criterio de factibilidad.	Con el uso de la herramienta este proceso si es posible realizarlo, demorándose para ello solo unos pocos segundos.

Teniendo en cuenta los resultados obtenidos, se observa que el uso de la herramienta reduce considerablemente el tiempo en que son implementadas las verticalizaciones del sistema, evitando así el rediseño y la reimplementación de estas.

Conclusiones del capítulo.

Luego de validar la propuesta de solución aplicando las métricas para la validación del diseño se obtuvieron resultados satisfactorios, demostrando la efectividad de la solución. Se le aplicaron pruebas de funcionalidad al software con el objetivo de garantizar el correcto funcionamiento de este. La solución realizada minimizó el tiempo en que se realiza el proceso de verticalización en el sistema Cedrux.

CONCLUSIONES

El presente trabajo permite arribar a las siguientes conclusiones:

- La definición del marco teórico de la investigación, permitió conocer las principales características de los ADLs estudiados para sentar las bases para el desarrollo del plugin.
- El *plugin* implementado se integra correctamente con la herramienta AcmeStudio, dotándola de funcionalidades que permiten dar cumplimiento al objetivo del trabajo.
- Una vez realizada las pruebas al software, este se aplicó a un caso de estudio donde se analizó el resultado arrojado, permitiendo así la validación del trabajo.

RECOMENDACIONES

Se recomienda:

- Investigar para tener en cuenta más indicadores a la hora de realizar los cálculos sobre la estructura de componentes.
- Optimizar la métrica para calcular la factibilidad de las variaciones arquitectónicas, puesto que la métrica actual no incluye todos los atributos de calidad.

Referencias

BARAZA, M., 2009: http://www.channelbiz.es/.

DAVID GARLAN, R. M. Y. D. W., GARY T. LEAVENS Y MURALI SITARAMAN "Acme: Architectural description of component-based systems". Foundations of Component- Based Sys-tems Cambridge University Press, pp. 2000.

EDWIN Network Comunidad Gamer, 2006: http://foro.ignetwork.net/showthread.php?15188-IDE-Entorno-integrado-de-desarrollo-%15128Concepto-importante%15129

ERIC DASHOFY, H. A., SCOTT HENDRICKSON, GIRISH SURYANARAYANA, JOHN GEORGAS, RICHARD TAYLOR] ArchStudio 4: An Architecture-Based Meta-modeling Environment., Department of Informatics, University of California, Irvine. 2007.

FERNÁNDEZ., I. O. L. Propuesta metodológica para la obtención de los componentes de software en los proyectos del sistema Cedrux., 2011.

GALLARDO, D. Developing Eclipse plug-ins., 2002: http://www.ibm.com/developerworks/java/library/os-ecplug/.

GUGLIELMETTI, M. Definición de Arquitectura Software, 2010: http://www.mastermagazine.info/termino/3916.php.

JACOBSON, I., P. JONSSON, M. CHRISTERSON AND G. OVERGAARD Ingeniería de Software Orientada a Objetos - Un acercamiento a través de los casos de uso, 1992.

JOHN ARTHORNE, C. L. Official Eclipse 3.0 Faqs. s.l., Addison-Wesley Professional 2004.

KICILLOF, C. R.-N. Lenguajes de descripción arquitectónica de Sofware (ADL), Universidad de Buenos Aires 2004.

MARIÑO, R. Diseño e implementación de un plugin de Eclipse que agilice el desarrollo de aplicaciones Web que utilicen la arquitectura única Dalas, 2010.

MORALES, E. C. V. Y. Y. M. Selección de un Lenguaje de descripción Arquitectonica para el modelado arquitectónico del proyecto ERP Cuba, 2009: http://www.ecured.cu

NÉSTOR DÍAZ VALENCIA, D. V. S. Morfeo Fromación - Metodologías ágiles, 2009: http://formacion.morfeo-project.org/wiki/index.php/PT3:GPSL:UF6.

OBREGÓN, G. O. Plugin de Eclipse para desarrollar proyectos web integrados con ArBaWeb, 2008.

PRESSMAN, R. Ingeniería de Software. Un Enfoque Práctico. Quinta Edición. McGraw Hill., 2002.

ROBERTH G. FIGUEROA, C. J. S. Metodologías Tradicionales vs. Metodologías Ágiles, 2007: http://www.mygnet.net/manuales/software/metodologias_tradicionales_vs_dot_metodologias_agiles.1515.

ROJAS, M. Análisis y Diseño de Sistemas, 2011: http://mrojas-ayds2011.blogspot.com/2011/2008/requerimientos-suplementarios.html.

Tecnología y Synergix, 2008: http://synergix.wordpress.com/2008/2006/2025/plantillas-documento-de-vision-del-sistema/.

VESTAL, S. An Overview and Comparison of Four Architecture Description Languages, Informe técnico. Honeywel Tecnology Centerl 1993.

Prieto., Félix. Patrones de diseño. 2009.

Fernández González, Mairelys y Zorrilla Rivera, Osley. Diseño e implementación del componente Ajuste al Costo del Subsistema Costos y Procesos del Sistema Integral de Gestión de Entidades CEDRUX. Habana : UCI, 2010.

CRITERIO DE EXPERTOS: METODO DELPHY. 2006. pág. 21.

Carmina Lizeth Torres Flores, Germán Harvey Alférez Salinas. Establecimiento de una Metodología de Desarrollo de Software para la Universidad de Navojoa Usando OpenUP. Ciudad de Navojoa : s.n., 2008. pág. 2.

www.visual-paradigm.com. UML CASE Tools - Free for Learning UML, Cost-Effective for Business Solutions. [En línea] [Citado el: 12 de 02 de 2012.] http://www.visual-paradigm.com/product/vpuml/..

Larman, Craig. UML y Patrones, 2da Edición. Vancouver,Canadá : s.n. pág. 162, pdf.

Rodríguez, Juan José Rosales. Desarrollo de la versión 2.0 del importador de datos del producto auditoría CEDRUX. La Habana, Cuba. : s.n., 2011. pág. 16.

epf.eclipse.org. Introduction to OpenUp. [En línea] [Citado el: 13 de 2 de 2012.] http://epf.eclipse.org/wikis/openup/index.htm.

OpenUp. [En línea] [Citado el: 28 de 2 de 2012.] http://epf.eclipse.org/wikis/openup/index.htm.

Pressman, Roger S. Ingeniería del Software: Un enfoque práctico. 2005.

Mairelys Fernández González, Osley Zorrilla Rivera. Biblioteca de la Universidad de las Ciencias Informática. [En línea] 2010. [Citado el: 2 de mayo de 2012.] http://biblioteca2.uci.cu/.