UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA

FACULTAD DE INGENIERÍA DE MINAS GEOLOGÍA Y CIVIL

ESCUELA DE FORMACIÓN PROFESIONAL DE INGENIERÍA DE SISTEMAS

"INFLUENCIA DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS EN LA MANTENIBILIDAD DEL SOFTWARE"

Tipo de investigación	:	Tecnológica.
Área de investigación	:	Tecnologías de Información
Ejecutor	:	Bach. Emiliano Espinoza Quispe
Asesor	:	Ing. Manuel A. Lagos Barzola.

Ayacucho-Perú

2015

"INFLUENCIA DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS EN LA MANTENIBILIDAD DEL SOFTWARE"

RECOMENDADO: **11 DE AGOSTO DEL 2015**

APROBADO : **01 DE OCTUBRE DEL 2015**

ING. JUAN C. CARREÑO GAMARRA
PRESIDENTE (E)

ING. ELINAR CARRILLO RIVEROS
MIEMBRO

ING. MANUEL A. LAGOS BARZOLA
MIEMBRO

ING. FLORO N. YANGALI GUERRA
SECRETARIO DOCENTE

Según el acuerdo constatado en el Acta. levantada el 01 de octubre del 2015, en la sustentación de Tesis presentado por el Bachiller en Ingeniería de sistemas Sr. Emiliano Espinoza Quispe, con la Tesis Titulado "INFLUENCIA DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS EN LA MANTENIBILIDAD DEL SOFTWARE", fue calificado con la nota de QUINCE(15) por lo que se da la respectiva APROBACIÓN.

RECOMENDADO: 11 DE AGOSTO DEL 2015

APROBADO : 01 DE OCTUBRE DEL 2015

ING. JUAN C. CARREÑO GAMARRA
PRESIDENTE (E)

ING. ELINAR CARRILLO RIVEROS
MIEMBRO

ING. MANUEL A. LAGOS BARZOLA
MIEMBRO

ING. FLORO N. YANGALI GUERRA
SECRETARIO DOCENTE

DEDICATORIA:

A las personas que se levantaron una y otra vez sin importar cuan fuerte fue la caída.

AGRADECIMIENTOS:

Agradezco a mis padres
por brindarme su apoyo incondicional. A mis amigos
por su amistad y a los profesores por su comprensión

CONTENIDO

CAPÍTULO III

METODOLOGÍA DE LA INVESTIGACIÓN

CAPÍTULO IV

ANÁLISIS Y RESULTADOS DE LA INVESTIGACIÓN

CAPÍTULOV

CONCLUSIONES Y RECOMENDACIONES

RESUMEN

La presente investigación trata sobre la influencia de la programación orientada a aspectos en la mantenibilidad del software. Se trata de evaluar el grado de mantenibilidad de dos productos software: un producto software desarrollado mediante la programación orientada a objetos y otro desarrollado usando la programación orientado a aspectos.
El mantenimiento del software se torna costoso, cuando el sistema es complejo, causando gastos adicionales a las compañías propietarias del software como a las consultoras desarrolladoras de software.
El objetivo principal del trabajo de tesis es medir la calidad de ambos productos, esto es específicamente en el grado de la mantenibilidad, y ver la posibilidad de que la programación orientada a aspectos mejore la calidad del software en la sub característica de la mantenibilidad.
El resultado esperado es que la programación orientada a aspectos mejore la mantenibilidad del software.
Los beneficiarios principales serán la comunidad de desarrollo de software, los estudiantes y docentes universitarios, los cuales podrán conocer las bondades de la programación orientada a aspectos.

PALABRAS CLAVE

Programación orientada a aspecto (POA), aspectos, programación orientada a objetos (POO), mantenibilidad del software.

INTRODUCCIÓN

A medida que pasa el tiempo el desarrollo de software se ha vuelto más complejo debido a que las empresas han ido creciendo. Según Irrazábal E. y Garzás J. (2010) la calidad del software influye directamente sobre los costes finales del desarrollo del software y la sub característica mantenibilidad tiene un impacto directo sobre el mantenimiento del producto software.

Según Reina (2000) en los primeros años del desarrollo de los lenguajes de programación se tenía un código en el que no había separación de conceptos, datos y funcionalidad. Luego nació la programación estructurada, la cual logra una separación básica entre datos y funcionalidad, apareciendo posteriormente la Programación Orientada a Objetos (POO). El problema de la Programación Orientada a Objetos es que las funciones quedan esparcidas por todo el código.

La propuesta para solucionar los problemas anteriormente mencionados se basa en un nuevo paradigma denominado "Programación Orientada a Aspectos", el cual se fundamenta en una nueva unidad llamada "Aspecto" que tiene como objetivo modularizar y encapsular los conceptos entrecruzados y de esta manera resolver de forma eficiente los conflictos derivados de este problema.

Esta investigación se realiza con el motivo de contribuir con la comunidad de estudiosos y desarrolladores de software.

El problema de investigación se centra en comparar el grado de mantenibilidad de dos productos software desarrollados bajo dos paradigmas diferentes, de esta manera se estará contribuyendo en la investigación de este nuevo paradigma de programación y abrir nuevos campos de investigación en este tema.

CAPÍTULO I
PLANTEAMIENTO DE LA INVESTIGACIÓN

1.1 DIAGNÓSTICO Y ENUNCIADO DEL PROBLEMA

A medida que crece la empresa, los sistemas de información que utilizan generalmente crecen por sus requerimientos adicionales, debido a este fenómeno el desarrollo del producto software se vuelve cada vez más complejo, de tal forma que el mantenimiento de dichos sistemas se toma dificultoso.

"Estudios previos señalan el mantenimiento como la fase que más recursos requiere a lo largo del ciclo de vida del producto, dos veces superior a los costes de desarrollo" (Irrazábal y Garzás , 2010, p.58).

El mantenimiento del software se torna costoso, cuando el sistema es complejo, causando gastos adicionales a las compañías propietarias del software y en otros casos a las consultoras desarrolladoras de software.

Según Ruiz F. y Polo M. (2001) el esfuerzo de mantenimiento de software se puede reducir si se produce software de calidad y también se podrán reducir los costes futuros, si el mantenimiento se realiza utilizando técnicas que mejoren alguna de sus características de calidad.

"La programación orientada a aspectos (POA) es una nueva metodología de programación que aspira a soportar la separación de competencias para los aspectos tanto en el diseño como en la implementación de software." (Baigorria, Montejano y Riesco, 2006, p.368).

Actualmente no se tiene información de empresas que esté usando este paradigma de programación en el desarrollo de sus proyectos software, pero si podemos asegurar que algunos frameworks los agregaron como parte de sus componentes, por ejemplo en el caso del framework spring, la plataforma .net también lo agregó como parte de

sus componentes.

Por tanto en esta tesis se tratara de determinar si la programación orientada a aspectos, puede mejorar o no el esfuerzo de mantenimiento del producto software respecto a la programación orientada a objetos

1.2 FORMULACIÓN DEL PROBLEMA DE INVESTIGACIÓN

1.2.1 PROBLEMA PRINCIPAL

¿Cómo la aplicación de la programación orientada a aspectos influye en la mantenibilidad del software, 2015?

1.2.2 PROBLEMAS SECUNDARIOS

a. ¿Cómo es la mantenibilidad del software antes de la aplicación de la programación orientada a aspectos?
b. ¿Cómo es la mantenibilidad del software después de la aplicación de la programación orientada a aspectos?
c. ¿Cuál es la diferencia de la mantenibilidad del software, antes y después de aplicarse la programación orientada a aspectos?

1.3 OBJETIVOS DE LA INVESTIGACIÓN

1.3.1 OBJETIVO GENERAL

Determinar la influencia de la programación orientada a aspectos en la mantenibilidad del producto software, 2015. Mediante la evaluación de la calidad del software, en la sub característica mantenibilidad.

1.3.2 OBJETIVOS ESPECÍFICOS

a. Precisar el nivel de mantenibilidad del software, antes de la aplicación de la programación orientada a aspectos.
b. Determinar la mantenibilidad del software después de la aplicación de la

programación orientada a aspectos.

c. Determinar la diferencia de la mantenibilidad del software antes y después de aplicarse la programación orientada a aspectos.

1.4 HIPÓTESIS DE LA INVESTIGACIÓN

La aplicación de la Programación Orientada a Aspectos influye significativamente en la mantenibilidad del software.

1.5 JUSTIFICACIÓN Y DELIMITACIÓN DE LA INVESTIGACIÓN

1.5.1 IMPORTANCIA DEL TEMA

IMPORTANCIA TÉCNICA

En la actualidad la POA no se está usando y esto debido al desconocimiento de las potencialidades de este nuevo paradigma de programación. Es posible que mediante el uso de la programación orientada a aspectos los costos de mantenimiento disminuyan, el código de software sea más ordenado y la implementación de la seguridad sea menos engorroso y el manejo de roles del software sea más sencillo porque la POA permite desarrollar software seguro y mantenible.

La investigación servirá para determinar la influencia de la programación orientada a aspectos sobre la calidad del software en la sub característica mantenibilidad.

IMPORTANCIA SOCIOECONÓMICA

Las empresas podrán conocer las bondades de la Programación Orientada a Aspectos y es posible que con el uso de este paradigma dichas empresas puedan disminuir el coste de mantenimiento de sus aplicaciones.

1.5.2 JUSTIFICACIÓN

Los costes del mantenimiento del producto software son muy elevados en la actualidad irá aumentando con el paso del tiempo. Existen varios factores para que dicho coste sea elevado y uno de ellos es que el código del producto software no está ordenado y estructurado adecuadamente; por tanto surge la necesidad de una solución

para dicho problema y una de ellas es el nuevo paradigma de la Programación Orientada a Aspectos (POA).

1.5.3 DELIMITACIÓN

Se desarrollará una aplicación web con la Programación Orientada a Objetos y luego a partir de este último se desarrollará otra aplicación con la Programación Orientada a Aspectos, para luego evaluar el grado de calidad de la mantenibilidad de ambas aplicaciones, específicamente en el caso de uso registrar documento para puntualizar la investigación. Los gastos de la investigación corren por cuenta del investigador.

Se evaluará los productos software utilizando el modelo de evaluación de calidad de código fuente; propuesto por la empresa Arisa.

CAPÍTULO II
MARCO DE REFERENCIA DE LA INVESTIGACIÓN

2.1 ANTECEDENTES

En la Universidad del Bío-Bío (Almonacid y Hernández, 2008) realizaron una investigación en el que se plantean como desarrollar un software de calidad y el cual tratan de resolverlo realizando una evaluación del enfoque de Programación Orientada a Aspectos a través del estudio minucioso de sus antecedentes teóricos y elementos conceptuales además de la realización de una aplicación basada tanto en la Programación Orientada a Aspectos como en la Programación Orientada a Objetos. En la evaluación de ambas aplicaciones usaron las métricas propuestas por Manuel F. Bertoa y Antonio Vallecillo, pertenecientes al departamento de Ciencias de la Computación de la Universidad de Málaga España. Esas métricas son beneficio poa, la *limpieza*, medición en líneas de código de una aplicación. La primera se refiere a cuan eficiente es la POA frente a la POO y la segunda se refiera a la pureza del código (cohesión) entre ambas aplicaciones. AL final concluye que tanto como el beneficio poa y la limpieza de la POA son superiores de la POO.

En la Universidad Estatal de Campinas (Ferreira, 2006) realizó una investigación sobre el manejo de excepciones en la Programación Orientada a Aspectos y para alcanzar este objetivo construye una aplicación orientada a la POA y otra aplicación orientada a la POO. Se encontró que la POA mejora la separación entre el código de manejo de excepciones y código funcional de la aplicación.

2.2 MARCO TEÓRICO

2.2.1 PROGRAMACIÓN ORIENTADA A ASPECTOS

Según Kiczales (1997) la programación orientada a aspectos es un nuevo paradigma de programación que aspira a soportar la separación de competencias para los aspectos, es decir intenta separar los componentes y aspectos unos de otros.

A.1 ASPECTO

Para Kiczales (1997) un aspecto es una unidad modular que se dispersa por la estructura de otras unidades funcionales. Un aspecto de diseño es una unidad modular que se entremezcla en la estructura de otras partes del diseño.

Para Walls (2008) un aspecto es la fusión de notificación y puntos de corte. Tomados juntos, notificaciones y puntos de corte definen todo lo que hay que saber sobre un aspecto: qué es lo que hace, dónde y cuándo lo hace.

Para Cáceres (2004) hay conceptos que por su naturaleza no pueden ser encapsulados dentro de una sola unidad funcional, estos son denominados crosscutting concerns (asuntos transversales) o aspectos. Algunos ejemplos de aspectos son: chequeo de errores, seguridad, sincronización, logging o registro de la actividad de una aplicación, etc.

A.2 PUNTO DE UNIÓN

"Un punto de unión es un punto en la ejecución de la aplicación en el que se puede insertar un aspecto. Este punto puede ser la llamada a un método, una excepción o incluso la modificación de un campo." (Walls, 2008/2009, p. 146).

A.3 PUNTO DE CORTE

"Una definición de punto de corte se corresponde con uno o más puntos de unión en los que debe incrustarse una notificación." (Walls, 2008/2009, p. 146).

A.4 DESTINATARIO

"El destinatario (target) es el objeto que está siendo notificado. Puede ser un objeto que usted escriba o un objeto de terceros al que quiere añadir un comportamiento personalizado." (Walls, 2008/2009, p. 147).

A.5 INTRODUCCIÓN

"Una introducción (introduction) le permite añadir nuevos métodos o atributos a clases existente." (Walls, 2008/2009, p. 147).

A.6 RESULTANTE

Walls (2008/2009) afirma que:

Un resultante (proxy) es un objeto creado después de aplicar una notificación al objeto destinatario. En tanto concierne a los objetos clientes, el objeto destinatario (pre-AOP) y el objeto proxy (post-AOP) son el mismo, como debería ser. Es decir, el resto de la aplicación no tendrá que cambiar para dar soporte al objeto proxy (p. 147).

A.7 WEAVING

"Weaving es el proceso de aplicar aspectos a un objeto destinatario para crear un nuevo objeto resultante. Los aspectos se entretejen en el objeto destinatario en los puntos de unión especificados." (Walls, 2008/2009, p. 147).

A.8 RELACIÓN ENTRE ASPECTOS Y CLASES

Para Baigorria et al. (2006). Se refiere a que un aspecto puede relacionarse con una o varias clases. Esta relación es la que muestra que una clase se ve afectada por dicho aspecto.

A.9 COHESIÓN

"La cohesión es una indicación cualitativa del grado que tiene un módulo para centrarse en una sola cosa." (Pressman, 2002, p. 230).

A.10 CANTIDAD DE ASPECTOS

Baigorria et al. (2006) se refiere al número de aspectos presentes en un software. Si la cantidad de aspectos es muy alta la metodología nos beneficia en cambio si la cantidad de aspectos es baja se puede complicar el diseño en vez de mejorarlo.

A.11 PUNTOS DE ENLACE

Para Baigorria et al. (2006) un punto de enlace es la interfaz entre los aspectos y los módulos del lenguaje de componentes, es decir, es un lugar del código en el que se puede aumentar comportamientos adicionales agregados en un aspecto.

2.2.2 MANTENIBILIDAD DEL SOFTWARE

Según APA (2006) es el conjunto de atributos que soporta el esfuerzo necesario para realizar modificaciones especificadas.

La Mantenibilidad es parte de las 6 características de calidad de software, los cuales son funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad, portatilidad.

La mantenibilidad a su vez se divide en 5 sub características que son: analizabilidad del software, cambiabilidad del software, estabilidad del software, testeabilidad del software y conformidad de facilidad de mantenimiento.

La norma estándar ISO9126 enfoca la calidad en dos tipos:

Calidad Interna.- Puede ser medida y evaluada por medio de atributos estáticos de documentos tales como: especificación de requerimientos, arquitectura o diseño, piezas de código fuente etc. En etapas tempranas del ciclo de vida del software es posible medir, evaluar y controlar la calidad interna de estos productos.

Calidad Externa: Puede ser medida y evaluada por medio de propiedades dinámicas del código ejecutable en un sistema de computación, es decir la aplicación completa es ejecutado en una computadora lo más cercanamente posible a un ambiente real. En fases tardías del ciclo de vida del software es posible medir, evaluar y controlar la calidad externa de estos productos ejecutables.

2.2.3 ANALIZABILIDAD DEL SOFTWARE

Según APA (2006) evalúan atributos relacionados con el esfuerzo del mantenedor o el usuario o los recursos gastados para determinar deficiencias o causas de fallos, o para identificar las partes que deben ser cambiadas.

2.2.4 CAMBIABILIDAD DEL SOFTWARE

Según APA (2006) miden atributos relacionados con el esfuerzo del mantenedor o el usuario para medir la conducta del mantenedor, el usuario o el sistema software cuando se intenta llevar a cabo una modificación determinada.

2.2.5 ESTABILIDAD DEL SOFTWARE

Según APA (2006) miden atributos relacionados con la conducta inesperada del sistema software cuando dicho software es probado u operado después de una modificación.

2.2.6 TESTEABILIDAD DEL SOFTWARE

Según APA (2006) evalúan propiedades relacionados con el esfuerzo del mantenedor o el usuario para medir la conducta del mantenedor, el usuario o el sistema software cuando se intenta probar el software.

2.2.7 CONFORMIDAD DE FACILIDAD DE MANTENIMIENTO

Según APA (2006) es la capacidad del producto software de satisfacer los estándares o convenciones relativas con la mantenibilidad.

2.2.8 MÉTRICAS DE CALIDAD DEL SOFTWARE RELACIONADOS A LA MANTENIBILIDAD.

Según la Empresa Arisa las métricas relacionadas a la mantenibilidad son.

		Main property	Maintainability			
		Sub Property	Analyzability	Changeability	Stability	Testability
Category	Sub-Category	Metric				
Complexity	size	LOC	--	--	-	--
	interface C.	NAM	--	--	-	--
		NOM	--	--	-	--
	structural C.	WMC	--	--	-	--
		RFC	--	--	-	--
Architecture & Structure	Inheritance	DIT	--	--	-	--
		NOC	-	--	-	-
	Coupling	CBO	--	--	--	--
		DAC	--	--	--	--
		LD	++	++	++	++
		MPC	--	--	--	--
	Cohesion	LCOM	--	--	--	--
		ILCOM	--	--	--	--
		TCC	++	++	++	++
Design guidelines & Coding conventions	Documentation	LOD	--	--	-	--
	Guidelines	LEN	--	--	--	--
		CYC	--	--	--	--

Figura N° 2.1 Relación de las métricas con la mantenibilidad del producto software, fuente: manual de la herramienta VizzMaintenance.

a) Cantidad de instrucciones (LOC)

Según Ebert, C. et al. (2005) es la cantidad de líneas físicas de código sin considerar las líneas blancas y comentarios.

b) Numero de atributos y métodos (NAM)

Según el compendio de VizzMaintenance, NAM es el número de atributos y métodos de una clase. Es una métrica orientada a objetos.

e) Numero de métodos (NOM)

Según el compendio de VizzMaintenance, NOM es el número de métodos declarados en una clase.

d) Peso de los métodos por clase (WMC)

Según el compendio de VizzMaintenance, es la suma del peso de métodos implementados dentro de una clase.

Originalmente fue definido como una métrica orientada a objetos, puede fácilmente ser adaptado para sistemas poco orientados a objetos.

e) Respuesta de una clase (RFC)

Según Stephen H. Kan (2003) especifica el número de métodos de la propia clase más el número de métodos externos que utiliza la clase en cuestión.

Según Stephen H. Kan (2003) es la suma de las complejidades de los métodos, dicha complejidad está relacionado a la complejidad ciclomática.

f) Profundidad del árbol de herencia (DIT)

Según Stephen H. Kan (2003) especifica la máxima longitud del camino entre la clase y la clase padre o raíz.

g) Número de hijos (NOC)

Según Stephen H. Kan (2003) especifica el número directo de sucesores de la clase.

h) Acoplamiento entre objetos (CBO)

Según Stephen H. Kan (2003) es el número de clases acopladas. Una clase objeto esta acoplada a otro cuando una de ellas llama una función o usa la variable de la otra clase objeto (herencia).

i) Acoplamiento de datos abstractos(DAC)

Según el compendio de VizzMaintenance, mide la complejidad del acoplamiento causado por datos abstractos.

j) Localidad de datos(LD)

Según el compendio de VizzMaintenance, esta métrica está relacionada con la cantidad de datos locales usados por la misma clase.

k) Acoplamiento de mensajes enviados (MPC)

Según el compendio de VizzMaintenance, mide el número de llamadas que hace un método, definido en una clase, a métodos definidos en otras clases.

l) Ausencia de cohesión (LCOM)

Según Stephen H. Kan (2003) cada método de una clase tiene acceso a uno o más atributos. LCOM calcula el conjunto de atributos comunes en los métodos de una clase. Dos métodos son similares si comparten al menos un atributo de la clase. A mayor número de atributos similares, mayor cohesión hay en la clase.

m) Valor de cohesión entre clases (TCC)

Según el compendio de VizzMaintenance, mide la cohesión entre métodos públicos de una clase.

n) Falta de documentación en las clases (LOD)

Según el compendio de VizzMaintenance, mide el número de comentarios faltantes en una clase. Se considera que un comentario por clase y por método

es lo más óptimo.

o) Longitud de Nombres(LEN)

Según el compendio de VizzMaintenance, es la longitud del nombre de una clase.

p) Numero de dependencias cíclicas (CYC)

Según el compendio de VizzMaintenance, es número de dependencias cíclicas entre clases.

2.2.9 ICONIX

Según Rosenberg y Stephens (2007) el proceso ICONIX es un "recetario" en el que se describe una serie de pasos específicos que trabajan muy bien en diferentes proyectos. Sin embargo, no prescribe el lado del ciclo de vida del proyecto en la forma en que la mayoría de las otras metodologías de desarrollo que lo hacen.

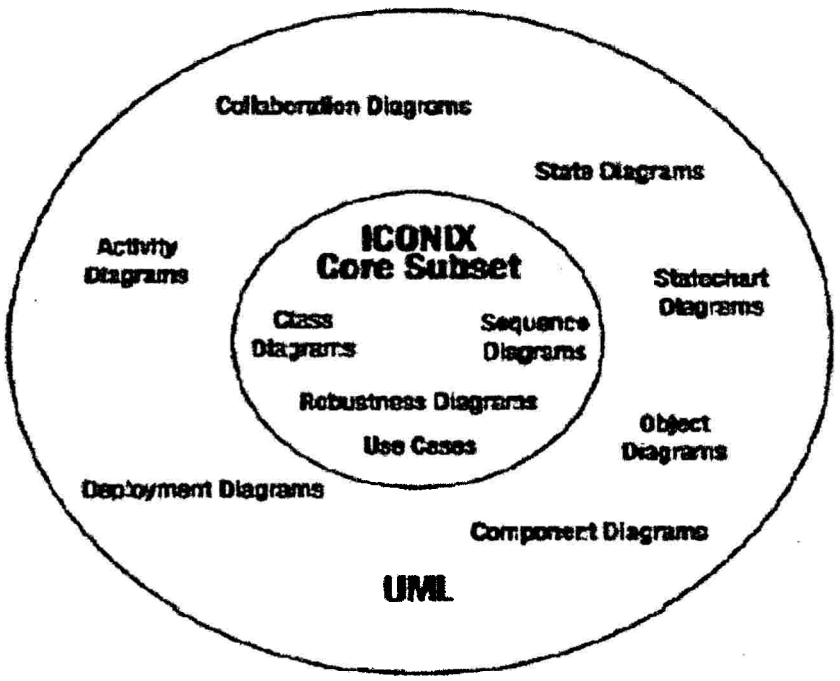

Figura N° 2.2 Diagramas que usa ICONIX del Estándar UML (Rosenberg y Stephens, 2007)

El proceso se compone de las siguientes actividades:

Etapa 1: Revisión de requisitos.

Etapa 2: Revisión preliminar de diseño.

Etapa 3: Revisión crítica y al detalle del diseño.

Etapa 4: Entrega

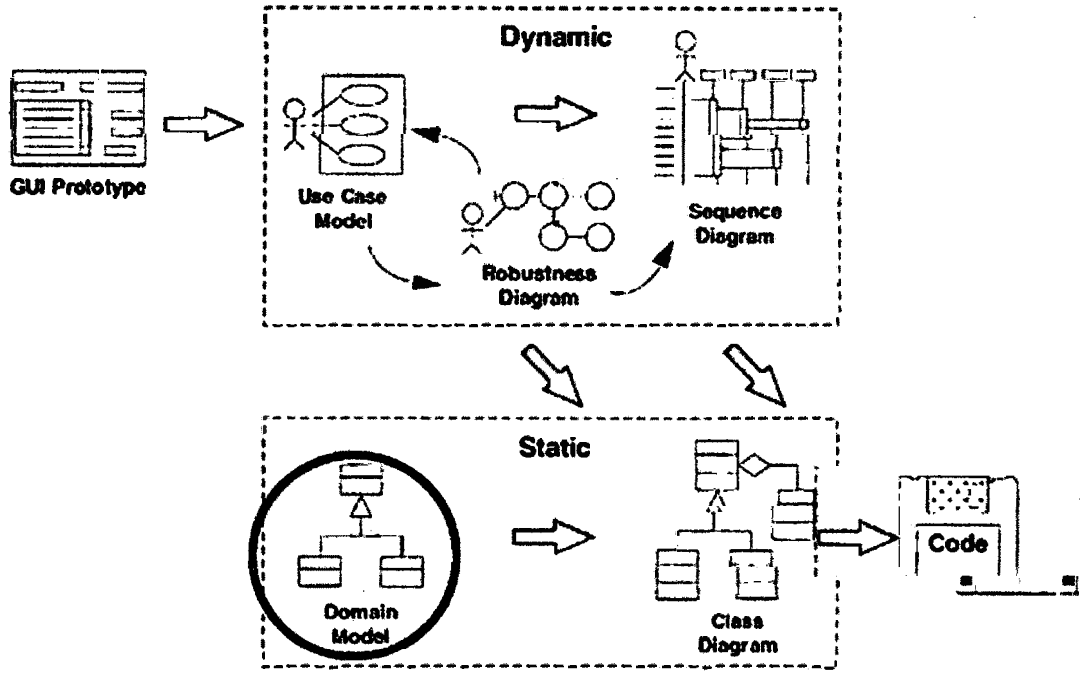

Figura N° 2.3 Proceso ICONIX (Rosenberg y Stephens, 2007)

En resumen, los aspectos principales del Proceso ICONIX son:

a. Requerimientos (Etapa1: revisión de requerimientos).
b. Análisis y diseño preliminar.
c. Etapa 2: Revisión del diseño preliminar.
d. Diseño.
e. Etapa 3: Revisión crítica del diseño.
f. Implementación.
g. Etapa 4: Entrega.

2.2.10 ADMINISTRADOR DE BASE DE DATOS

Según Teory, Lightstone y Nadeau (2006) es un sistema de software que permite manipular base de datos también soporta una vista lógica (shema, subshema), una vista física (métodos de acceso, clastering de datos), un lenguaje de definición de

datos, un lenguaje de manipulación de datos y otras utilidades.

Según Bai (2010) un sistema administrador de base de datos se usa para almacenar, ganar acceso, y manipular los datos en la base de datos de tal manera que los detalles de almacenamiento de datos y el mantenimiento están escondidos del usuario. El usuario interactúa con la base de datos a través del sistema de gestión de base de datos. Un usuario puede interactuar directamente con el sistema de gestión de base de datos o por un programa escrito a en un lenguaje de programación como C + +, C#, Java, o el Visual Basic.

Los administradores de bases de datos más usados son:

a) Oracle
b) Microsoft SQLServer
c) PostgreSql
d) MySql
e) Otros

2.2.11 PROGRAMACIÓN ORIENTADA A OBJETOS

Según Joyanes (2003) es un método de implementación donde los programas se organizan como colecciones cooperativas de objeto el cual representa una instancia de una clase, las clases son miembros de una jerarquía de clases unidas por una relación de herencia.

Los lenguajes de programación orientada a objetos más populares son: Java, C#, C++, etc.

Java

Para Poo, Kiong y Ashock (2007) Java fue introducida en 1995 como un lenguaje de programación orientada a objetos. Después de que Java fue lanzado, hubo aproximadamente unos 400,000 programadores Java y más de 100 libros de programación en Java. Una de las bondades de Java es que la aplicación desarrollada en este lenguaje es portátil. En relación a la Internet, los applets de Java le han dado

lugar a una nueva época de las aplicaciones distribuidas con distribución baja del software y los costes de mantenimiento.

C#

Para Bai (2010) es un lenguaje de programación orientado a objetos orientado a que admite conceptos de encapsulación, herencia y polimorfismo. Fue desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma.NET, similar al de Java aunque incluye mejoras derivadas de otros lenguajes (entre ellos Delphi).

C++

Para Stroustrup (1992) es un lenguaje imperativo orientado a objetos derivado del C es un superconjunto de C, que nació para añadirle cualidades y características de las que carecía. El resultado es que como su ancestro, sigue muy ligado al hardware subyacente, manteniendo una considerable potencia para programación a bajo nivel, pero se la han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción.

2.3 DEFINICIÓN DE TÉRMINOS BÁSICOS

2.3.1 PROGRAMACIÓN ORIENTADA A ASPECTO (POA)

Según Kiczales (1997) la programación orientada a aspectos es un nuevo paradigma de programación que intenta separar los componentes y aspectos unos de otros.

2.3.2 ASPECTOS

Para Kiczales (1997) un aspecto es una unidad modular que se dispersa por la estructura de otras unidades funcionales.

2.3.3 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

Según Joyanes (2003) es un método de implementación donde los programas se organizan como colecciones cooperativas de objeto el cual representa una instancia de una clase, las clases son miembros de una jerarquía de clases unidas por una relación de herencia.

2.3.4 MANTENIBILIDAD DEL SOFTWARE

Según APA (2006) es el conjunto de atributos que soporta el esfuerzo necesario para realizar modificaciones especificadas.

CAPITULO III

METODOLOGÍA DE LA INVESTIGACIÓN

3.1 TIPO DE INVESTIGACIÓN

El tipo de investigación es tecnológica puesto según Sánchez H. y Reyes, C. (2002) una investigación tecnológica responde a problemas técnicos y en la Programación Orientada a Objetos existen problemas técnico de mantenimiento de software los cuales trata de resolver la Programación Orientada a Aspectos.

3.2 NIVEL DE INVESTIGACIÓN

Según Bernal (2006) el nivel de investigación es descriptivo; desde el punto de vista científico una investigación descriptiva muestra, narran, reseñan o identifican hechos, situaciones, rasgos, características de un objeto de estudio o se diseñan productos, modelos, prototipos, guías, etcétera.

3.3 DISEÑO DE LA INVESTIGACIÓN

Según Carrasco (2006) el diseño de investigación pre-experimental de preprueba – posprueba con una sola medición consiste en aplicar a un grupo una prueba previa al estímulo o tratamiento experimental y luego administrar el tratamiento y después aplicar la prueba o medición posterior, tiene el diagrama siguiente:

G	O	X	O

Al tener la información previa del nivel o situación real de la variable dependiente, se podrá determinar los cambios experimentales con el estímulo (X), en la post prueba.

Por lo descrito anteriormente, el diseño de investigación correspondiente a esta investigación es el pre-experimental de preprueba-postprueba:

G	O	X	O

G: Software para gestión contable.

O: Medición del grado de mantenibilidad del software contable con Programación Orientada Objetos.

X: Aplicación de la Programación Orientada Aspectos al software contable.

O: Medición del grado de mantenibilidad del software contable con POA.

3.4 POBLACIÓN Y MUESTRA

No aplicable para este tipo de investigación

3.5 VARIABLES E INDICADORES

3.5.1 DEFINICIÓN CONCEPTUAL DE LAS VARIABLES

Variable independiente:

Programación orientada a aspectos.- Es una es un nuevo paradigma de programación que separa claramente componentes y aspectos unos de otros.

Variable dependiente

Mantenibilidad del software.- Se refiera al conjunto de atributos que soporta el esfuerzo necesario para realizar modificaciones especificadas.

3.5.2 INDICADORES DE LA VARIABLE PROGRAMACIÓN ORIENTADA A ASPECTOS

a) **Puntos de enlace.**- Es la interfaz entre los aspectos y los módulos del lenguaje de componentes, es decir, es un lugar del código en el que se puede aumentar comportamientos adicionales agregados en un aspecto.

b) **Relación entre aspectos y clases.**- Se refiere a que un aspecto puede

relacionarse con una o varias clases. Esta relación es la que muestra que una clase se ve afectada por dicho aspecto.

c) **Cohesión**.- La cohesión es una indicación cualitativa del grado que tiene un módulo para centrarse en una sola cosa

3.5.3 INDICADORES DE LA VARIABLE MANTENIBILIDAD DEL SOFTWARE

a) **Analizabilidad**.- Evalúan atributos relacionados con el esfuerzo del mantenedor o el usuario o los recursos gastados para determinar deficiencias o causas de fallos, o para identificar las partes que deben ser cambiadas.

b) **Cambiabilidad**.- Miden atributos relacionados con el esfuerzo del mantenedor o el usuario para medir la conducta del mantenedor, el usuario o el sistema software cuando se intenta llevar a cabo una modificación determinada.

c) **Estabilidad**.- Miden atributos relacionados con la conducta inesperada del sistema software cuando dicho software es probado u operado después de una modificación.

d) **Testeabilidad**.- Evalúan propiedades relacionados con el esfuerzo del mantenedor o el usuario para medir la conducta del mantenedor, el usuario o el sistema software cuando se intenta probar el software.

3.5.4 DEFINICIÓN OPERACIONAL DE LAS VARIABLES

a. Variable independiente:

Programación orientada a aspectos

Indicadores:

Puntos de enlace.

Relación entre aspectos y clases.

Cohesión

b. Variable dependiente

Mantenibilidad del software

Indicadores:

Analizabilidad

Cambiabilidad

Estabilidad

Testeabilidad

3.6 TÉCNICAS E INSTRUMENTOS

3.6.1 TÉCNICAS

Se utilizó la técnica de análisis documental para conceptualizar los términos de Programación Orientado a Aspectos, Programación Orientado a objetos y mantenibilidad del producto software. También se utilizó la técnica de observación para ver la variación de la mantenibilidad del producto software. Para medir el grado de mantenibilidad del producto software se utilizó la técnica de evaluación de software mediante una herramienta de análisis de código fuente.

3.6.2 INSTRUMENTOS

Análisis documental.

Ficha Bibliográfica	
Autor(a):	**Editorial:**
Título:	**Ciudad, país:**
Año:	
Resumen del contenido:	
Número de edición o Impresión:	

Relación de las métricas de calidad de software con la mantenibilidad.

Tabla N° A.1 Métricas de calidad de software relacionadas a la mantenibilidad del software.

Métricas	Analizabilidad	Cambiabilidad	Estabilidad	Capacidadde Prueba
CBO	--	--	--	--
CYC_Classes	--	--	--	--
DAC	--	--	--	--
DIT	-	-	-	-
ILCOM	--	--	--	--
LCOM	--	--	--	--
LD	++	++	++	++
LEN	-	-	-	-
LOC	--	--	-	--
LOD_Class	--	--	-	-
MPC	--	-	--	-
NAM	--	--	-	--
NOC	-	--	-	-
NOM	--	--	-	--
RFC	--	--	-	--
TCC	++	++	++	++
WMC	--	--	-	--

Dónde:

"-" indica que la métrica esta inversamente relacionada a la subcarateristica correspondiente;

"--" indica que la métrica está muy inversamente relacionada a la subcaraterística correspondiente.

"+" indica que la métrica está directamente relacionada a la subcaraterística correspondiente.

"++" indica que la métrica está muy directamente relacionada a la subcaraterística correspondiente.

Para el proceso de observación:

Ficha Observación	
Objetivo:	
Producto software:	**Producto software:**

Técnica programación:	Técnica programación:
Métrica:	Métrica:
Valor de la métrica:	Valor de la métrica:
Variación de la métrica:	
Interpretación:	

3.6.3 HERRAMIENTAS PARA EL TRATAMIENTO DE DATOS E INFORMACIÓN

Tabla Nº A.2 Herramientas para el tratamiento de datos

Nombre	Fabricante	Tipo Licencia	Descripción
Windows8	Microsoft Corporation	Licencia propietaria	Sistema Operativo
Microsoft Word 2010	Microsoft Corporation	Licencia propietaria	Procesador de texto
Microsoft Excel 2010	Microsoft Corporation	Licencia propietaria	Hoja de calculo
Enterprise Architect	Sparx Systems	Licencia propietaria	Herramienta case para el modelado de software.
Mysql	Sun-Oracle	Freeware	Sistema Gestor de base de datos.
Java	Sun-Oracle	Freeware	Lenguaje de programación
Netbeans	Sun-Oraclo	Freeware	Interfaz de desarrollo integral para programadores
Eclipse	Eclipse Foundation	Freeware	Interfaz de desarrollo integral para programadores

VizzMaintenance	Arisa	Freeware	Programa para medir la calidad de software

CAPITULO IV

ANÁLISIS Y RESULTADOS DE LA INVESTIGACIÓN

4.1.1 IMPLEMENTACIÓN ORIENTADA A OBJETOS

El análisis y diseño del sistema contable se ha realizado usando la metodología iconix cuyo desarrollo se encuentra en el anexo C, en las páginas siguientes se muestra la implementación con la programación orientada a objetos.

A. Ventanas Principales

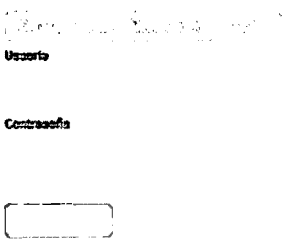

Figura N° 4.1 Ventana inicial para ingresar al Sistema, software Contabilidad Nueva Acrópolis

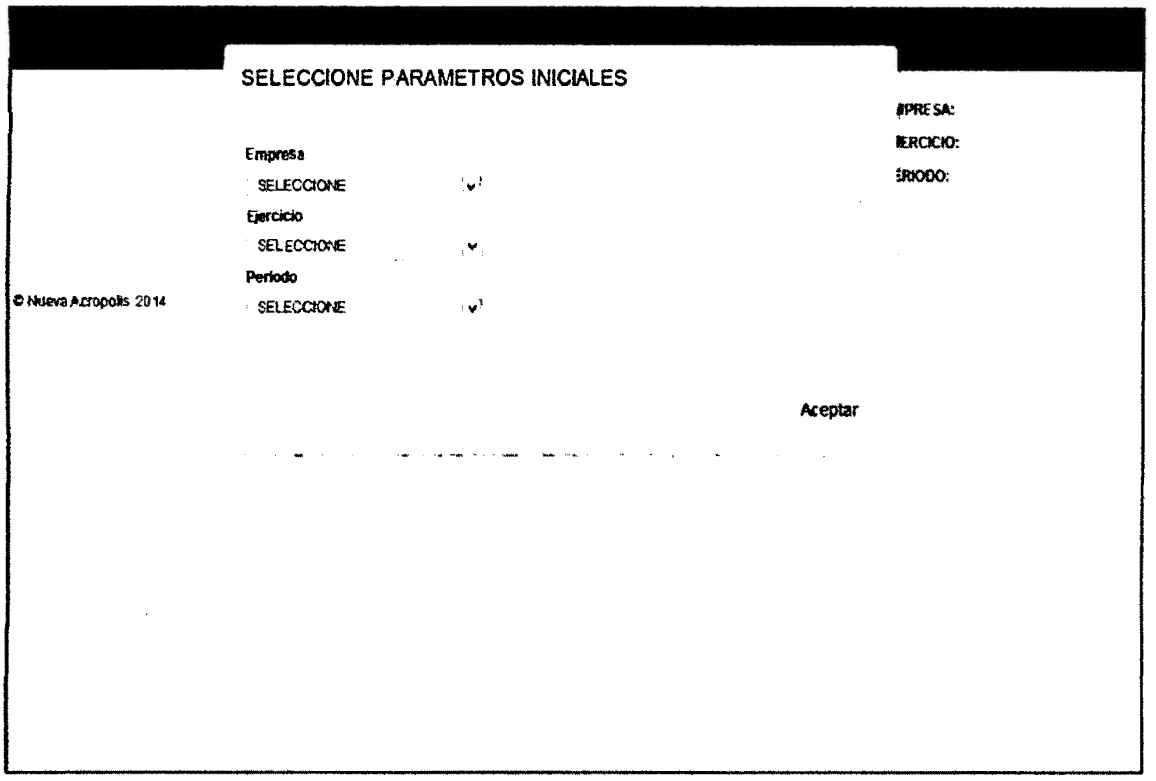

Figura N° 4.2 Ventana principal del Sistema, software Contabilidad Nueva Acrópolis

C. Implementación del caso de uso 04 Registrar documento

C.1 Capa Modelo

Implementación de la clase Documento

```
/**
 *
 * @author Emiliano
 */
public class Documento extends Generico {

    private int biIdRegDocumento;
    private int iIdMaeEmpresa;
    private int biIdMaeDetPeriodo;
    private int iIdMaeTipoOperacion;
    private int iIdRegBeneficiario;
    private int iIdMaeBanco;
    private int iIdMaeTipDocumento;
    private int iIdMaeSede;
    private int iIdMaeMoneda;
    // private int iIdMaeTipDocBeneficiario;
    private int iIdMaeTipDocReferencia;
    private String cCodRegistro;
    private Date dtFecOperacion;
    private String vNumDocumento;
    private String vNumSerie;
    private String vNumDeposito;
    private Date dtFecDeposito;
    private String vConcepto;
    private boolean bPago;
    private String vNumeracion;
    private String vNumSerReferencia;
    private String vNumDocReferencia;
    private Date dtFecDocumento;
    private float deImporte;
    private float deTipCambio;
    private float deImporteTotal;
    private String vRegManual;
    private List<DetalleDocumento> lstRegDetDocumento;
    private boolean bIndBancarizacion;
```

Figura N° 4.4 Código fuente de la clase Documento, software Contabilidad Nueva Acrópolis

C.2 Capa DataSource

Implementación de la clase Conexión

```
/**
 *
 * @anthor Emiliano
 */
public class Conexion {

    public static Connection conectarBaseDatos() {
        Connection cn = null;
        try {
            Context ctx = new InitialContext();
            DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/poolTest");
            cn = ds.getConnection();

        } catch (Exception ex) {
            Loger.escribirLog("CtrRegCuenta", "iniciarSesionSeguridad", ex.getMessage());
        }
        return cn;
    }
}
```

Figura N° 4.5 Código fuente de la clase Conexión, software Contabilidad Nueva Acrópolis

C.3 Capa DAO

Implementación del dao DaoNtxDocumento

```
public class DaoNtxDocumento extends DaoBase {

    private Connection oConnection;

    public DaoNtxDocumento() {
        this.oConnection = Conexion.conectarBaseDatos();
    }

    public static DaoNtxDocumento getInstance() {
        return new DaoNtxDocumento();
    }

    private CallableStatement agregarParametros(Documento poRegDocumento) throws Exception {
        CallableStatement cs= null;
        String pProcedure = "{ call regDocumento_lst(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?
        oConnection.setAutoCommit(false);
        cs= oConnection.prepareCall(pProcedure);
        /*Inicio parametros por defecto*/
        cs.setObject("piOpSp", poRegDocumento.getiOpSp());
        cs.setObject("piPaginaActual", poRegDocumento.getiPaginaActual());
        cs.setObject("piTamanioPagina", poRegDocumento.getiTamanioPagina());
        cs.registerOutParameter("piTotalRegistros", Types.INTEGER);
        cs.registerOutParameter("piTotalPaginas", Types.INTEGER);
        /*Fin parametros por defecto*/
        cs.setObject("pbiIdRegDocumento", poRegDocumento.getBiIdRegDocumento(), Types.INTEGER);
        cs.setObject("piIdMaeEmpresa", poRegDocumento.getiIdMaeEmpresa(), Types.INTEGER);
        cs.setObject("pbiIdMaeDetPeriodo", poRegDocumento.getBiIdMaeDetPeriodo(), Types.INTEGER);
        cs.setObject("piIdMaeTipoOperacion", poRegDocumento.getiIdMaeTipoOperacion(), Types.INTEGER);
        cs.setObject("piIdRegBeneficiario", poRegDocumento.getiIdRegBeneficiario(), Types.INTEGER);
        cs.setObject("piIdMaeBanco", poRegDocumento.getiIdMaeBanco(), Types.INTEGER);
        cs.setObject("piIdMaeTipDocumento", poRegDocumento.getiIdMaeTipDocumento(), Types.INTEGER);
        cs.setObject("piIdMaeSede", poRegDocumento.getiIdMaeSede(), Types.INTEGER);
        cs.setObject("piIdMaeMoneda", poRegDocumento.getiIdMaeMoneda(), Types.INTEGER);
```

Figura N°4.6 Código fuente de la clase DaoNtxDocumento, software Contabilidad Nueva Acrópolis

C.4 Capa Controlador

Implementación del controlador CtrDocumento

```
@Controller
@RequestMapping("/CtrDocumento")
@SessionAttributes({"oSession"})
public class CtrDocumento {

    @RequestMapping(value = "/validarAcceso.htm", method = RequestMethod.POST)
    public @ResponseBody
    int validarAcceso(@ModelAttribute("oSession") Session oSession) {
        try {
            if (oSession.getMapPermiso().containsKey(ParamSeguridad.FRM_DOCUMENTOS)) {
                return 1;
            } else {
                return 0;
            }
        } catch (Exception ex) {
            Loger.escribirLog("CtrDocumento", "validarAcceso", ex.getMessage());
            return 0;
        }

    }

    @RequestMapping(value = "/guardarDatos.htm", method = RequestMethod.POST)
    public @ResponseBody
    int guardarDatos(@RequestBody Map<String, Object> param, @ModelAttribute("oSession") Session oSession) {
        try {
            Documento poRegDocumento = HelperMapper.getParam(param, "poRegDocumento", Documento.class);
            List plstRegDetDocumento = HelperMapper.getParam(param, "plstDetDocumento", List.class);
            completarParametros(poRegDocumento, oSession);
            if (oSession.getOperacion() == ParameterEnu.GUARDAR.getId()) {
                if (!oSession.getMapPermiso().containsKey(ParamSeguridad.CONT_DOCUMENTOS_001)) {
                    return 0;
                }
```

Figura N° 4.7 Código fuente de la clase CtrDocumento, software Contabilidad Nueva Acrópolis

4.2.2 IMPLEMENTACIÓN ORIENTADA A ASPECTOS

A. Implementación del aspecto AspectThrowing

```
/**
 *
 * @author Emiliano
 *
@Aspect
public class AspectThrowing {

    @AfterThrowing(
            pointcut = "execution(* kayros.seguridad.controller.*.*(..)) || "
            + "execution(* kayros.contabilidad.controller.*.*(..)) || "
            + "execution(* kayros.contabilidad.reporte.controller.*.*(..))",
            throwing = "ex")
    public void controlExcepcion(JoinPoint jp, Exception ex) {
        String clase= jp.getTarget().getClass().getName();
        String metodo = jp.getSignature().getName();
        Loger.escribirLog(clase, metodo, ex.getMessage());
    }
}
```

Figura N° 4.9 Código fuente del aspecto AspectThrowing, software Contabilidad Nueva Acrópolis

B. Implementación del aspecto AspectSeguridad

```
/**
 *
 * @author Edilians
 *
@Aspect
public class AspectSeguridad {

    @Around("execution(* kayros.contabilidad.controller.*.*validar*(..)) ||"
            + " execution(* kayros.contabilidad.reporte.controller.*.*validar*(..))")
    public int validarAccesoForm(ProceedingJoinPoint pjp) throws Throwable {...}

    @Around("execution(* kayros.contabilidad.controller.*.*mantener*(..))")
    public int validarAccesoControl(ProceedingJoinPoint pjp) throws Throwable {
        int retVal = 0;
        Session oSession = (Session) pjp.getArgs()[1];
        Map param = (Map) pjp.getArgs()[0];
        int pIdForm = HelperMapper.getParam(param, "pIdForm", Integer.class);
        switch (pIdForm) {
            case 1: //Documentos
                if (oSession.getOperacion()== ParameterEnu.GUARDAR.getId()) {
                    if (oSession.getMapPermiso().containsKey(ParamSeguridad.CONT_DOCUMENTOS_001)) {
                        retVal = (Integer) pjp.proceed();
                    }
                }
                if (oSession.getOperacion() == ParameterEnu.ACTUALIZAR.getId()) {
                    if (oSession.getMapPermiso().containsKey(ParamSeguridad.CONT_DOCUMENTOS_002)) {
                        retVal = (Integer) pjp.proceed();
                    }
                }
                if (oSession.getOperacion() == ParameterEnu.ELIMINAR.getId()) {
                    if (oSession.getMapPermiso().containsKey(ParamSeguridad.CONT_DOCUMENTOS_003)) {
                        retVal = (Integer) pjp.proceed();
                    }
                }
                break;
```

Figura Nº 4.10 Código fuente del aspecto AspectSeguridad, software Contabilidad Nueva Acrópolis

4.2 EVALUANDO LAS DOS APLICACIONES CON UNA HERRAMIENTA DE ANÁLICES DE CÓDIGO FUENTE

4.3.1 EVALUACIÓN DE HERRAMIENTAS PARA DETERMINAR LA MANTENIBILIDAD

Según Irrazábal E. y Garzás J. (2010) existen varias herramientas para medir la mantenibilidad del software y se muestra en la siguiente tabla.

la Nº 4.1 Herramientas que permiten medir el valor de las métricas relacionadas a la mantenibilidad.

Herramientas	Métricas calculadas
JavaNCSS	CC, LOC
PMD/CPD	VCF,CDU

CheckStyle	VST
FindBugs	VCF, CDU
JDepend	CYC,D,CAF,CEF
CCCC	CBO,DIT,NOM, NOC ,NOM ,RFC, LOC, PCOM, CC, WMC, D,CEF, CAF
StyleCop	VCF
FxCop	VST
JUnit	UTE
CPPUnit	UTE
NUnit	UTE
EMMA	COB
Analyst4j	CC, LOC, PLOC, PCOM, CBO,DIT,LCOM,NOC,NOM,RFC,WMC
C&K Java Metrics	CBO, DIT, LCOM, NOC, NOM, RFC
Dependency Finder	LOC, NNA, CCD, DIT, NOC, NOM
Eclipse Metrics	CC,LOC,CAF,CEF,D,LCOM,CEF,DIT,NOC,NOM
OOMeter	CBO, DIT, LCOM, NOC
Semmie	DIT, LCOM, NOC, NOM, RFC
Understand for Java	CBO,DIT, LCOM, NOC, NOM
VizzAnalyzer (VizzMaintenance)	LOC, CBO, DIT, LCOM, NOC, NOM, RFC, WMC
CodePro AnalytiX	CC, LOC, PLOC, PCOM, NCB, NSI, NNA, CCD, D, CYC, CAF, CEF, CBO, DIT, LCOM,NOC, NOM, RFC, WMC, CDU

De la tabla anterior se escogen las herramientas que abarquen la mayor cantidad de métricas asociadas a la mantenibilidad, y luego de hacer la investigación respectiva por google y otros buscadores se obtiene la siguiente tabla:

a Nº 4.2 Resultado de la evaluación de las herramientas de la tabla 4.6

Herramienta	N° de métricas relacionados a la mantenibilidad	Observación
Analyst4j	11	Está en desuso
C&K Java Metrics	6	No abarca las métricas relacionadas a la mantenibilidad.
Dependency Finder	6	No abarca las métricas relacionadas a la mantenibilidad.
Eclipse Metrics	10	No abarca las métricas relacionadas a la mantenibilidad.
OOMeter	4	No abarca las métricas relacionadas a la mantenibilidad.
Semmie	5	Está en desuso, no se encontró el instalador
Understand for Java	5	Está en desuso, no se encontró el instalador
VizzAnalyzer (VizzMaintenance)	17	Es freware, pero también hay una versión gratuita, abarca la mayor parte métricas relacionadas a la mantenibilidad más el índice de mantenibilidad
CodePro AnalytiX	20	Abarca las métricas relacionadas a la mantenibilidad, pero el software muestra falsos positivos y no está actualizado desde al año 2012.

4.3.2 EVALUANDO AMBAS APLICACIONES CON VIZZMAINTENANCE

Se escogió esta herramienta para medir el grado de la mantenibilidad del producto software, porque tiene una amplia documentación; abarca la mayor cantidad de métricas relacionadas a la mantenibilidad ver aparado 3.6.2.

a) Software sin programación orientada a aspectos

me	Maintainability	CBO	CYC Classes	DAC	DIT	ILCOM	LCOM	LD	LEN	LOC	LOD Class	MPC	NAM	NOC	NOM	RFC	TCC	WMC
se	0,1174	1	1	1	1	2	0	0,0000	4	25	0,8000	0	6	0	4	4	0,3333	4
neficiario	0,2348	1	1	1	1	7	140	0,0000	12	70	0,9333	0	21	0	14	14	0,0769	14
ntroCosto	0,1174	1	1	1	1	2	0	0,0000	11	25	0,8000	0	6	0	4	4	0,3333	4
mision	0,1366	1	1	1	0	0	1	0,0000	8	19	0,5000	1	1	0	1	2	0,0000	1
ntrol	0,2348	1	1	1	1	5	60	0,0000	7	52	0,9091	0	15	0	10	10	0,1111	10
rBase	0,1366	5	1	4	0	0	1	0,0000	7	23	0,5000	4	1	0	1	5	0,0000	1
rBeneficiario	0,1857	11	1	9	0	0	100	0,0000	15	172	0,9091	47	10	0	10	34	0,0000	18
rCentroCosto	0,1366	5	1	4	0	0	1	0,0000	14	22	0,5000	4	1	0	1	5	0,0000	1
rControl	0,1857	10	1	8	0	0	49	0,0000	10	123	0,8750	43	7	0	7	33	0,0000	10
rCuenta	0,2540	11	1	10	0	0	100	0,0000	9	175	0,9091	51	10	0	10	34	0,0000	15
rCuentaContable	0,3224	14	1	11	0	0	100	0,0000	17	169	0,9091	50	10	0	10	40	0,0000	17
rDetalleCentroCosto	0,2050	6	1	5	0	0	1	0,0000	21	24	0,5000	5	1	0	1	6	0,0000	1
rDetalleDocumento	0,1366	8	1	6	0	0	9	0,0000	19	57	0,7500	25	3	0	3	19	0,0000	3
rDetallePeriodo	0,1366	7	1	6	0	0	1	0,0000	17	23	0,5000	6	1	0	1	7	0,0000	1
rDocumento	0,3807	14	1	11	0	8	100	0,0000	12	184	0,9091	70	10	0	10	41	0,0000	19
rEmpresa	0,1366	6	1	5	0	0	1	0,0000	10	21	0,5000	5	1	0	1	6	0,0000	1
rFormulario	0,3224	12	1	10	0	0	64	0,0000	13	128	0,8889	39	8	0	8	29	0,0000	11
rParametro	0,1366	3	1	3	0	0	1	0,0000	12	21	0,5000	2	1	0	1	3	0,0000	1
rPeriodo	0,1366	6	1	5	0	0	1	0,0000	10	21	0,5000	5	1	0	1	6	0,0000	1
rPermisoControl	0,1366	9	1	7	0	0	25	0,0000	17	90	0,8333	24	5	0	5	18	0,0000	9
rPermisoFormulario	0,1366	9	1	7	0	0	25	0,0000	20	90	0,8333	24	5	0	5	18	0,0000	7
rPrincipal	0,1366	6	1	5	0	0	1	0,0000	12	37	0,5000	9	1	0	1	10	0,0000	1
rReporte	0,1366	3	1	3	0	0	9	0,0000	10	57	0,7500	6	3	0	3	5	0,0000	6
rRol	0,1366	6	1	5	0	0	1	0,0000	6	22	0,5000	5	1	0	1	6	0,0000	1
rSede	0,3224	13	1	11	0	0	81	0,0000	7	147	0,9000	42	9	0	9	31	0,0000	16
rTipDocBeneficiario	0,2050	6	1	5	0	0	1	0,0000	21	21	0,5000	5	1	0	1	6	0,0000	1
rTipoDocumento	0,3224	13	1	11	0	0	81	0,0000	16	153	0,9000	42	9	0	9	31	0,0000	16
rTipoMoneda	0,1366	6	1	5	0	0	1	0,0000	13	21	0,5000	5	1	0	1	6	0,0000	1
rTipoOperacion	0,1366	9	1	8	0	0	16	0,0000	16	64	0,8000	15	4	0	4	14	0,0000	4

Figura N° 4.11 Resultado de la evaluación del producto software codificado sin utilizar POA, software Contabilidad Nueva Acrópolis

b) Software con programación orientada a aspectos

me	Maintainability	CBO	CYC Classes	DAC	DIT	ILCOM	LCOM	LD	LEN	LOC	LOD Class	MPC	NAM	NOC	NOM	RFC	TCC	WMC
spectSeguridad	0,1366	4	1	4	0	0	4	0,0000	15	157	0,6667	4	2	0	2	5	0,0000	42
spectThrowing	0,1366	1	1	1	0	0	1	0,0000	14	18	0,5000	1	1	0	1	2	0,0000	1
se	0,1174	1	1	1	1	2	0	0,0000	4	25	0,8000	0	6	0	4	4	0,3333	4
eneficiario	0,2348	1	1	1	1	7	140	0,0000	12	70	0,9333	0	21	0	14	14	0,0769	14
entroCosto	0,1174	1	1	1	1	2	0	0,0000	11	25	0,8000	0	6	0	4	4	0,3333	4
omision	0,1366	0	1	0	0	0	1	0,0000	8	14	0,5000	0	1	0	1	1	0,0000	1
ontrol	0,2348	1	1	1	1	5	60	0,0000	7	52	0,9091	0	15	0	10	10	0,1111	10
trBase	0,1366	4	1	3	0	0	1	0,0000	7	19	0,5000	3	1	0	1	4	0,0000	1
trBeneficiario	0,1857	9	1	7	0	0	100	0,0000	15	114	0,9091	36	10	0	10	32	0,0000	18
trCentroCosto	0,1366	4	1	3	0	0	1	0,0000	14	17	0,5000	3	1	0	1	4	0,0000	1
trControl	0,1857	9	1	7	0	0	49	0,0000	10	94	0,8750	37	7	0	7	32	0,0000	10
trCuenta	0,2199	10	1	9	0	0	100	0,0000	9	123	0,9091	42	10	0	10	33	0,0000	13
trCuentaContable	0,2882	12	1	9	0	0	100	0,0000	17	116	0,9091	42	10	0	10	38	0,0000	13
trDetalleCentroCosto	0,2050	5	1	4	0	0	1	0,0000	21	19	0,5000	4	1	0	1	5	0,0000	1
trDetalleDocumento	0,1366	7	1	5	0	0	9	0,0000	19	42	0,7500	22	3	0	3	18	0,0000	3
trDetallePeriodo	0,1366	6	1	5	0	0	1	0,0000	17	18	0,5000	5	1	0	1	6	0,0000	1
trDocumento	0,3565	12	1	9	0	0	100	0,0000	12	132	0,9091	60	10	0	10	39	0,0000	15
trEmpresa	0,1366	5	1	4	0	0	1	0,0000	10	16	0,5000	4	1	0	1	5	0,0000	1
trFormulario	0,2540	11	1	9	0	0	64	0,0000	13	92	0,8889	32	8	0	8	28	0,0000	11
trParametro	0,1366	2	1	2	0	0	1	0,0000	12	15	0,5000	1	1	0	1	2	0,0000	1
trPeriodo	0,1366	5	1	4	0	0	1	0,0000	10	16	0,5000	4	1	0	1	5	0,0000	1
trPermisoControl	0,1366	8	1	6	0	0	25	0,0000	17	72	0,8333	20	5	0	5	17	0,0000	7
trPermisoFormulario	0,1366	8	1	6	0	0	25	0,0000	20	70	0,8333	20	5	0	5	17	0,0000	7
trPrincipal	0,1366	5	1	4	0	0	1	0,0000	12	30	0,5000	8	1	0	1	9	0,0000	1
trReporte	0,1366	1	1	1	0	0	9	0,0000	10	27	0,7500	0	5	0	5	5	0,0000	5
trRol	0,1366	5	1	4	0	0	1	0,0000	6	17	0,5000	4	1	0	1	5	0,0000	1
trSede	0,2540	11	1	9	0	0	81	0,0000	7	97	0,9000	32	9	0	9	29	0,0000	12
trTipDocBeneficiario	0,2050	5	1	4	0	0	1	0,0000	21	16	0,5000	4	1	0	1	5	0,0000	1
trTipoDocumento	0,2540	11	1	9	0	0	81	0,0000	16	100	0,9000	32	9	0	9	29	0,0000	12

Figura N° 4.12 Resultado de la evaluación del producto software codificado utilizando POA, software Contabilidad Nueva Acrópolis

Tabla Nº 4.3 Cálculo del valor promedio de cada uno de las métricas de la aplicación web desarrollado con POO

Cálculo del promedio de las métricas de la aplicación web con POO			
Métricas	**Suma total**	**Cantidad total clases**	**Promedio(Suma/Total clases)**
Maintainability	19.0286		19.0300
CBO	179.0000	101	1.7723
CYC_Classes	101.0000	101	1.0000
DAC	144.0000	101	1.4257
DIT	27.0000	101	0.2673
ILCOM	98.0000	101	0.9703
LCOM	7515.0000	101	74.4059
LD	0.0000	101	0.0000
LEN	1482.0000	101	14.6733
LOC	7103.0000	101	70.3267
LOD_Class	84.7950	101	0.8396
MPC	316.0000	101	3.1287
NAM	677.0000	101	6.7030
NOC	27.0000	101	0.2673
NOM	579.0000	101	5.7327
RFC	847.0000	101	8.3861
TCC	3.5515	101	0.0352
WMC	732.0000	101	7.2475

Tabla Nº 4.4 Cálculo del valor promedio de cada uno de las métricas de la aplicación web desarrollado con POA

Cálculo del promedio de las métricas de la aplicación web con POA			
Métricas	**Suma total**	**Cantidad total clases**	**Promedio(Suma/Total clases)**
Maintainability	19.4000		19.4000
CBO	178.0000	101	1.7282
CYC_Classes	103.0000	101	1.0000
DAC	143.0000	101	1.3883
DIT	27.0000	101	0.2621
ILCOM	98.0000	101	0.9515
LCOM	7520.0000	101	73.0097
LD	0.0000	101	0.0000
LEN	1521.0000	101	14.7670

LOC	6694.0000	101	64.9903
LOD_Class	86.7950	101	0.8427
MPC	316.0000	101	3.0680
NAM	680.0000	101	6.6019
NOC	27.0000	101	0.2621
NOM	582.0000	101	5.6505
RFC	850.0000	101	8.2524
TCC	3.5515	101	0.0345
WMC	752.0000	101	7.3010

4.3 RESULTADOS DE LA EVALUACION DE LAS DOS APLICACIONES

De las tablas 4.3 y 4.4 se obtiene la tabla 4.5:

Tabla N° 4.5 Valores de las métricas y su variación de las dos Aplicaciones.

Metricas	Aplicación Contabilidad con POO	Aplicación Contabilidad con POA	Dif
Maintainability	19.0300	19.4000	0.3700
CBO	1.7723	1.7282	-0.0441
CYC_Classes	1.0000	1.0000	0.0000
DAC	1.4257	1.3883	-0.0374
DIT	0.2673	0.2621	-0.0052
ILCOM	0.9703	0.9515	-0.0188
LCOM	74.4059	73.0097	-1.3962
LD	0.0000	0.0000	0.0000
LEN	14.6733	14.7670	0.0937
LOC	70.3267	64.9903	-5.3364
LOD_Class	0.8396	0.8427	0.0031
MPC	3.1287	3.0680	-0.0608
NAM	6.7030	6.6019	-0.1010
NOC	0.2673	0.2621	-0.0052
NOM	5.7327	5.6505	-0.0822
RFC	8.3861	8.2524	-0.1337
TCC	0.0352	0.0345	-0.0007
WMC	7.2475	7.3010	0.0534

Tabla Nº 4.6 Valores del comportamiento de las métricas según la sub- característica de la mantenibilidad

Métricas	Analizabilidad	Cambiabilidad	Estabilidad	Capacidad de Prueba
CBO	--	--	--	--
CYC_clase	--	--	--	--
DAC	--	--	--	--
DIT	--	--	-	--
ILCOM	--	--	--	--
LCOM	--	--	--	--
LD	++	++	++	++
LEN	--	--	--	--
LOC	--	--	-	--
LOD_Clase	--	--	-	--
MPC	--	--	--	--
NAM	--	--	-	--
NOC	-	--	-	-
NOM	--	--	-	--
RFC	--	--	-	--
TCC	++	++	++	++
WMC	--	--	-	--

A continuación se muestran datos observados respecto a los resultados de la evaluación de la mantenibilidad del producto software realizado con la herramienta VizzMaintenance:

Ficha Observación N° 1	
Objetivo: verificar la variación de la métrica CBO antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica: CBO	**Métrica:** CBO
Valor de la métrica: 1.7723	**Valor de la métrica:** 1.7282
Variación de la métrica: -0.0441	

Interpretación: en la tabla 4.6 se observa que si el valor de la métrica CBO disminuye entonces la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba aumentan (relación indirecta fuerte --) y como la métrica CBO en la tabla 4.5 ha disminuido de 1.7723 (valor con POO) a 1.7282 (valor con POA), por tanto podemos decir que la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba (mantenibilidad) ha mejorado respecto a la métrica CBO.

Ficha Observación N° 2	
Objetivo: verificar la variación de la métrica CYC antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica: CYC	**Métrica:** CYC
Valor de la métrica: 1.0000	**Valor de la métrica:** 1.0000
Variación de la métrica: 0.0000	
Interpretación: la valor de la variación de esta métrica es 0.0000 por tanto la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba (mantenibilidad) no han mejorado.	

Ficha Observación N° 3	
Objetivo: verificar la variación de la métrica DAC antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica: DAC	**Métrica:** DAC
Valor de la métrica: 1.4257	**Valor de la métrica:** 1.3883
Variación de la métrica: -0.0374	
Interpretación: en la tabla 4.6 se observa que si el valor de la métrica DAC disminuye entonces la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba aumentan (relación indirecta fuerte --) y como la métrica DAC en la tabla 4.5 ha disminuido de 1.4257 (valor con POO) a 1.3883 (valor con POA), por tanto podemos decir que la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba (mantenibilidad) ha mejorado respecto a la métrica DAC.	

Ficha Observación N° 4	
Objetivo: verificar la variación de la métrica DIT antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa

Técnica programación: POO	**Técnica programación: POA**
Métrica: DIT	**Métrica:** DIT
Valor de la métrica: 0.2673	**Valor de la métrica:** 0.2621
Variación de la métrica: -0.0052	
Interpretación: en la tabla 4.6 se observa que si el valor de la métrica DIT disminuye entonces la analizabilidad, cambiabilidad y capacidad de prueba (relación indirecta fuerte --) y la estabilidad (relación indirecta débil -) aumentan, como la métrica DIT en la tabla 4.5 ha disminuido de 0.2673 (valor con POO) a 0.2621 (valor con POA), por tanto podemos decir que la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba (mantenibilidad) ha mejorado respecto a la métrica DIT.	

Ficha Observación N° 5	
Objetivo: verificar la variación de la métrica ILCOM antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica: ILCOM	**Métrica:** ILCOM
Valor de la métrica: 0.9703	**Valor de la métrica:** 0.9515
Variación de la métrica: -0.0188	
Interpretación: en la tabla 4.6 se observa que si el valor de la métrica ILCOM disminuye entonces la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba aumentan (relación indirecta fuerte --) y como la métrica ILCOM en la tabla 4.5 ha disminuido de 0.9703 (valor con POO) a 0.9515 (valor con POA), por tanto podemos decir que la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba (mantenibilidad) ha mejorado respecto a la métrica ILCOM.	

Ficha Observación N° 6	
Objetivo: verificar la variación de la métrica LCOM antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica: LCOM	**Métrica:** LCOM
Valor de la métrica: 74.4059	**Valor de la métrica:** 73.0097
Variación de la métrica: -1.3962	
Interpretación: en la tabla 4.6 se observa que si el valor de la métrica	

LCOM disminuye entonces la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba aumentan (relación indirecta fuerte –) y como la métrica LCOM en la tabla 4.5 ha disminuido de 74.4059 (valor con POO) a 73.0097 (valor con POA), por tanto podemos decir que la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba (mantenibilidad) ha mejorado respecto a la métrica LCOM.

Ficha Observación N° 7	
Objetivo: verificar la variación de una métrica antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica: LD	**Métrica:** LD
Valor de la métrica: 0.0000	**Valor de la métrica:** 0.0000
Variación de la métrica: 0.0000	
Interpretación: la variación de esta métrica es 0.00 por tanto no hubo variación respecto a la métrica LD.	

Ficha Observación N° 8	
Objetivo: verificar la variación de la métrica LEN antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica: LEN	**Métrica:** LEN
Valor de la métrica: 14.6733	**Valor de la métrica:** 14.7670
Variación de la métrica: 0.0937	
Interpretación: en la tabla 4.6 se observa que si el valor de la métrica LEN disminuye entonces la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba aumentan (relación indirecta fuerte –) y como la métrica LEN en la tabla 4.5 ha aumentado de 14.6733 (valor con POO) a 14.7670 (valor con POA), por tanto podemos decir que la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba (mantenibilidad) no ha mejorado respecto a la métrica LEN.	

Ficha Observación N° 9	
Objetivo: verificar la variación de la métrica LOC antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa

Técnica programación: POO	**Técnica programación: POA**
Métrica: LOC	**Métrica:** LOC
Valor de la métrica: 70.3267	**Valor de la métrica:** 64.9903
Variación de la métrica: -5.3364	
Interpretación: en la tabla 4.6 se observa que si el valor de la métrica LOC disminuye entonces la analizabilidad, cambiabilidad y capacidad de prueba (relación indirecta fuerte --) y la estabilidad (relación indirecta débil-) aumentan, como la métrica LOC en la tabla 4.5 ha disminuido de 70.3267 (valor con POO) a 64.9903 (valor con POA), por tanto podemos decir que la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba (mantenibilidad) ha mejorado respecto a la métrica LOC.	

Ficha Observación N° 10	
Objetivo: verificar la variación de la métrica LOD antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica: LOD	**Métrica:** LOD
Valor de la métrica: 0.8396	**Valor de la métrica:** 0.8427
Variación de la métrica: 0.0031	
Interpretación: en la tabla 4.6 se observa que si el valor de la métrica LDD disminuye entonces la analizabilidad, cambiabilidad y capacidad de prueba (relación indirecta fuerte --) y la estabilidad (relación indirecta débil -) aumentan, como la métrica LOD en la tabla 4.5 ha aumentado de 0.8396 (valor con POO) a 0.8427 (valor con POA), por tanto podemos decir que la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba (mantenibilidad) no ha mejorado respecto a la métrica LOD.	

Ficha Observación N° 11	
Objetivo: verificar la variación de la métrica MPC antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica: MPC	**Métrica:** MPC
Valor de la métrica: 3.1287	**Valor de la métrica:** 3.0680
Variación de la métrica: -0.0608	
Interpretación en la tabla 4.6 se observa que si el valor de la métrica MPC disminuye entonces la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba aumentan (relación indirecta fuerte --) y como la métrica MPC en la tabla 4.5 ha disminuido de 3.1287 (valor con POO) a 3.0680 (valor con POA), por tanto podemos decir que la analizabilidad,	

cambiabilidad, estabilidad y capacidad de prueba (mantenibilidad) ha mejorado respecto a la métrica MPC.

Ficha Observación N° 12	
Objetivo: verificar la variación de la métrica NAM antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica:NAM	**Métrica:** NAM
Valor de la métrica: 6.7030	**Valor de la métrica:** 6.6019
Variación de la métrica: -0.1010	
Interpretación: en la tabla 4.6 se observa que si el valor de la métrica NAM disminuye entonces la analizabilidad, cambiabilidad y capacidad de prueba (relación indirecta fuerte --) y la estabilidad (relación indirecta débil-) aumentan, como la métrica NAM en la tabla 4.5 ha disminuido de 6.7030 (valor con POO) a 6.6019 (valor con POA), por tanto podemos decir que la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba (mantenibilidad) ha mejorado respecto a la métrica NAM.	

Ficha Observación N° 13	
Objetivo: verificar la variación de la métrica NOC antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica: NOC	**Métrica:** NOC
Valor de la métrica: 0.2673	**Valor de la métrica:** 0.2621
Variación de la métrica: -0.0052	
Interpretación: en la tabla 4.6 se observa que si el valor de la métrica NOC disminuye entonces la analizabilidad, capacidad de prueba, la estabilidad (relación indirecta débil -) y cambiabilidad (relación indirecta fuerte--) aumentan, como la métrica NOC en la tabla 4.5 ha disminuido de 0.2673 (valor con POO) a 0.2621 (valor con POA), por tanto podemos decir que la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba (mantenibilidad) ha mejorado respecto a la métrica NOC.	

Ficha Observación N° 14	
Objetivo: verificar la variación de la métrica NOM antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**

Métrica: NOM	**Métrica:** NOM
Valor de la métrica: 5.7327	**Valor de la métrica:** 5.6505
Variación de la métrica: -0.0822	
Interpretación: en la tabla 4.6 la analizabilidad, cambiabilidad y capacidad de prueba están muy inversamente relacionadas a esta métrica, mientras que la estabilidad esta inversamente relacionada y como la variación de esta métrica es negativa, por tanto podemos decir que la mantenibilidad ha mejorado al aplicar POA respecto a esta métrica.	

Ficha Observación N° 15	
Objetivo: verificar la variación de la métrica RFC antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica: RFC	**Métrica:** RFC
Valor de la métrica: 8.3861	**Valor de la métrica:** 8.2524
Variación de la métrica: -0.1337	
Interpretación: en la tabla 4.6 la analizabilidad, cambiabilidad y capacidad de prueba están muy inversamente relacionadas a esta métrica, mientras que la estabilidad esta inversamente relacionada y como la variación de esta métrica es negativa, por tanto podemos decir que la mantenibilidad ha mejorado respecto a esta métrica.	

Ficha Observación N° 16	
Objetivo: verificar la variación de la métrica TCC antes y después de aplicar la programación orientada a aspectos.	
Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica: TCC	**Métrica:** TCC
Valor de la métrica: 0.0352	**Valor de la métrica:** 0.0345
Variación de la métrica: -0.0007	
Interpretación: en la tabla 4.6 la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba están muy directamente relacionadas a esta métrica y como la variación de esta métrica se aproxima a cero, por tanto podemos despreciar este valor negativo y afirmar que la mantenibilidad no han mejorado respecto a esta métrica.	

Ficha Observación N° 17
Objetivo: verificar la variación de la métrica WMC antes y después de aplicar la programación orientada a aspectos.

Producto software: Contabilidad	**Producto software:** ContabilidadPoa
Técnica programación: POO	**Técnica programación: POA**
Métrica: WMC	**Métrica:** WMC
Valor de la métrica: 7.2475	**Valor de la métrica:** 7.3010
Variación de la métrica: 0.0534	
Interpretación: en la tabla 4.6 se observa que si el valor de la métrica WMC disminuye entonces la analizabilidad, cambiabilidad y capacidad de prueba (relación indirecta fuerte --) y la estabilidad (relación indirecta débil-) aumentan, como la métrica WMC en la tabla 4.5 ha aumentado de 7.2475 (valor con POO) a 7.3010 (valor con POA), por tanto podemos decir que la analizabilidad, cambiabilidad, estabilidad y capacidad de prueba (mantenibilidad) no ha mejorado respecto a la métrica WMC.	

4.4 DISCUSIÓN DE RESULTADOS

a) Para evaluar la mantenibilidad del producto software se tuvo muchas dificultades, no se tenía un modelo adecuado que permitiera obtener la mantenibilidad del producto software a partir del código fuente. Existían varios modelos y finalmente se escogió el modelo propuesto por la empresa Arisa (desarrolladora de la herramienta VizzMaintenance, el cual fue utilizado para medir la calidad del producto software), este modelo de evaluación fue validado y verificado por investigadores de dicha empresa, además existen investigaciones y publicaciones referentes al modelo de evaluación, en la página web de la empresa.

b) Respecto a la investigación de Almonacid y Hernández realizados en el 2008, estoy de acuerdo que la cantidad líneas de código disminuye al aplicar POA sobre POO, ya en la ficha de observación n° 9 se nota que la cantidad de líneas de código por clase con POO es 70.3267 y con POA ese valor es 64.9903.

c) Respecto a la investigación de Ferreira realizado en el 2006, coincido en que la POA mejora la separación de código en el uso de las excepciones, ya que en el desarrollo de la aplicación contable se aplicó POA en el manejo de excepciones y se nota esa mejora en el manejo de excepciones.

CAPITULO V

CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

a) El valor de la mantenibilidad para el software contable evaluado, antes de aplicar la Programación Orientada a Aspectos es de 19.0300, esto se puede verificar específicamente en la Tabla 4.5.

b) Según la tabla 4.5 el valor de la mantenibilidad después de aplicar la programación orientada a aspectos al software contable es de 19.4000.

c) Según la tabla 4.5 la diferencia de la mantenibilidad del software antes y después de aplicarse la Programación Orientada a Aspectos al software contable es de 0.3700.

d) Según la ficha N° 1 hasta la ficha de N° 16 las métricas indican que la mantenibilidad del producto software ha mejorado, pero en la ficha de observación N° 8 la métrica LEN influye de manera negativa en la mantenibilidad del producto software cuya variación es 0.0937, en la ficha de observación N° 10 la métrica LOD influye de manera negativa en la mantenibilidad del producto software cuya variación es 0.0031 y en la ficha de observación N° 17 la métrica WMC influye también de manera negativa en la mantenibilidad del producto software cuya variación es 0.0534 despreciando estos valores (ya que se aproximan a cero) podemos afirmar que la POA influye de manera positiva en la mantenibilidad del producto software.

5.2 RECOMENDACIONES

a) Se sugiere a la comunidad de desarrollo de software utilizar la Programación Orientada a Aspectos como parte del desarrollo de sus soluciones software.

b) Se sugiere considerar la Programación Orientada a Aspectos dentro del currículo

de estudios de la carrera de ingeniería de sistemas, computación, sistemas de información y otros relacionados en la producción de software.

c) El campo de la Programación Orientada a Aspectos es muy amplio, se recomienda realizar estudios como por ejemplo la influencia de la POA en la gestión de tareas realizadas en un Sistema Operativo, influencia en la seguridad de una aplicación, etc.

d) El campo de investigación en calidad de software es muy amplio, se recomienda realizar investigaciones como por ejemplo desarrollo de Software para la medición automatizada de métricas de calidad de software.

BIBLIOGRAFÍA

1 Almonacid y Hernández (2008). ***Programación Orientada a Aspectos.*** Recuperado el 03 de noviembre de 2012, de http://cybertesis.ubiobio.cl/tesis/2008/almonacid_r/doc/almonacid_r.pdf

2 APA (2006). ***NTP ISO/IEC 9126.*** Lima, Perú: Indecopi.

3 Bai, Y. (2010). ***Practical Database Programming With Visual C#.NET.*** New Jersey, Estados Unidos: John Wiley & Sons, Inc., Hoboken.

4 Baigorria L., Montejano G., Riesco D. (2006).***MÉTRICAS C&K APLICADAS AL DISEÑO ORIENTADO A ASPECTOS.*** Recuperado el 07 de noviembre del 2012, de http://sedici.unlp.edu.ar/bitstream/handle/10915/22133/Documento_completo.PDF?sequence=1

5 Cáceres, Gonzáles A. (2004). ***Programación orientada a aspectos.*** Recuperado el 03 de octubre del 2014, de http://computacion.cs.cinvestav.mx/~acaceres/courses/udo/poo/files/slides/POA-01.pdf.

6 Carrasco, S. (2006). ***Metodología de la investigación científica.*** Lima, editorial San Marcos.

7 Ebert, C. et al. (2005). ***Best Practices in Software Measurement.*** Berlin, Germany: Springer.

8 Ferreira (2006).***Modularização de Tratamento de Exceções Usando Programação Orientada a Aspectos.*** Recuperado el 3 de octubre del 2012, de http://cutter.unicamp.br/document/?down=vtls000414124.

9 Irrazábal E. y Garzás J. (2010). Análisis de métricas básicas y herramientas de código libre para medir la mantenibilidad. ***Revista Española de Innovación, Calidad e Ingeniería de Software,*** 6(3), 56-65.

10 Joyanes Aguilar, L. (2003). ***Fundamentos de programación*** (3ª Ed.). Aravaca, España: McGraw-Hill/Interamericana de España, S.A.U.

11 Kiczales, G et al. (1997). ***Aspect-Oriented Programming.*** Recuperado el 22 de octubre del 2012, de http://www2.parc.com/csl/groups/sda/publications/papers/.../for-web.pdf.

12 Poo D., Kiong D. y Ashock S. (2007). ***Object-Oriented Programming and Java*** (2ª Ed.). Verlag, London: Springer

13 Pressman, R. (2002). ***Ingeniería de software*** (5ª Ed.). Aravaca, España: McGraw-Hill/Interamericana de España, S.A.U.

14 Sánchez, H. y Reyes, C. (2009). ***Metodología y diseños en la investigación científica*** (4ta Ed.) .Lima, Perú: Visión universitaria.

15 Reina, A. (2000). ***Visión General de la Programación Orientada a Aspectos.*** Recuperado el 20 de octubre del 2012, de http://www.lsi.us.es/docs/informes/aopv3.pdf

16 Rosenberg, D. y Stephens, M. (2007). ***Casos de Uso Guiados Por Modelado de Objetos Con UML.*** New York, Estados unidos.

17 Ruiz F. y Polo M. (2001). ***Mantenimiento del Software.*** Recuperado el 25 de octubre, de http://alarcos.inf-cr.uclm.es/doc/mso/

18 Stephen H. Kan (2003). ***Metrics and Models in Software Quality Engineering*** (2nd Edition). Addison-Wesley.

19 Stroustrup, B. (1992). ***El lenguaje de programación c++*** (2ª Ed.). Aravaca, España: McGraw-Hill/Interamericana de España, S.A.U.

20 Teory T., Lightstone S. y Nadeau T. (2006).**Database Modeling and Design** (4ª Ed.). United States of America: Elsevier Inc.

21 Walls, G (2008). ***Spring.*** Madrid, Ediciones Anaya Multimedia.

ANEXO A

A continuación se presenta las actividades, tareas, artefacto, técnica y participante en el desarrollo de software con el proceso Iconix.

A. Resumen de las tareas y actividades de cada etapa del proceso Iconix:

A.1 Análisis de requisitos

Tabla N° A.1 Análisis de requisitos (Porras, 2011)

Tarea	Artefacto	Técnica	Responsables
Identificar requisitos	Requisitos funcionales y no funcionales Casos de prueba	• Entrevistas • Definir lo que el sistema debe hacer • Escribir al menos un caso de prueba para cada requisito	Analista Cliente Usuario
Identificar objetos del mundo real y dibujar modelo de dominio.	Modelo de dominio	• Identificar clases clave del negocio • Identificar sustantivos y depurar • Identificar objetos en requisitos funcionales y asignar al modelo de dominio • Utilizar agregación y generalización	Analista
Realizar prototipo de interfaz grafica	Prototipo GUI	• Utilizar historias de eventos del usuario(storyboards) • Utilizar los requisitos funcionales	Programador Analista

		• Diseñar interfaz gráfica básica	
Descubrir casos de uso	• Lista de casos de uso	• Utilizar requisitos funcionales • Entrevistas	Usuario Cliente Analista
Dibujar y empaquetar casos de uso	• Diagrama de casos de uso • Paquete de casos de uso	• Identificar roles y responsabilidades de actores • Asociar actores con casos de uso • Relacionar casos de uso • Agrupar lógicamente casos de uso	Analista
Asignar requisitos funcionales a los casos de uso	Relación entre requisitos funcionales y casos de uso	• Asignar requisitos funcionales a los casos de uso	
Escribir el primer borrador de casos de uso	Primer borrador de casos de uso	• Utilizar glosario de objetos del modelo de dominio • Utilizar la regla de dos párrafos • Escribir el caso de uso como flujos de evento/respuesta • Escribir el caso de uso con estructura sustantivo-verbo-	

		sustantivo • Escribir caso de uso en voz activa • Referenciar por sus nombres las pantallas	

A.2 Revisión de Requisitos

Tabla Nº A.2 Revisión de requisitos (Porras, 2011)

Tarea	Artefacto	Técnica	Responsables
Revisar el modelo de dominio	Modelo de dominio	• Identificar al menos 80% de clases clave de dominio del problema	Analista Cliente Usuario Programador
Revisar el prototipo GUI	Prototipo GUI	• Diseñar precisión la GUI relacionada al caso de uso	
Revisar modelo de casos de uso	Casos de uso revisado	• Eliminar clases fuera del dominio del problema • Cambiar descripción de voz pasiva activa • Describir todos los curos alternos • Asociar todos los requisitos a los casos de uso • Describir que intenta hacer el	

		usuario para cada caso de uso	

A.3 Diseño Preliminar

Tabla Nº A.3 Diseño preliminar (Porras, 2011)

Tarea	Artefacto	Técnica	Responsables
Rescribir el primer borrador para cada caso de uso	Casos de uso desambiguado	• Rescribir el caso de uso durante el análisis de robustez	
Identificar el primer corte de objetos que completan escenarios para cada caso de uso	Diagrama de robustez completo.	• Copiar la descripción del caso de uso en el diagrama de robustez • Usar las clases del modelo de dominio • Crear un objeto interfaz por cada GUI y nombrarlo • Transformar verbos del caso de uso en controladores • Relacionar un caso de uso al diagrama de robustez cuando es invocado • Utilizar las reglas para construir el diagrama de robustez	Analista

Actualizar el modelo de dominio	Modelo de dominio actualizado	• Actualizar el modelo de dominio con nuevas clases y atributos durante el análisis de robustez	
Actualizar el diagrama de clases de análisis	Modelo de dominio actualizado	• Actualizar el diagrama de clases de análisis al finalizar el análisis de robustez • Asignar atributos a las clases entidad	

A.4 Revisión de diseño preliminar

Tabla N° A.4 Revisión de diseño preliminar (Porras, 2011)

Tarea	Artefacto	Técnica	Responsables
Revisar *descripción* de casos de uso	Caso de uso	• Coincidir la descripción del caso de uso con el diagrama de secuencia.	Usuario Cliente Analista Programador
Revisar diagrama de robustez	Diagrama de robustez	• Coincidir el diagrama de robustez con descripción del caso de uso • Verificar que el diagrama de robustez cumple las reglas. • Verificar que el diagrama de robustez tiene todos los cursos alternos	

Revisar modelo de dominio actualizado	Modelo de dominio actualizado	• Coincidir los objetos entidad del diagrama de robustez con el modelo de dominio actualizado	

A.5 Arquitectura técnica

Tabla Nº A.5 Arquitectura (Porras, 2011)

Tarea	Artefacto	Técnica	Responsables
Diseñar diagrama de componentes	Diagrama de componentes	• Entrevistas • Características del negocio • Utilizar la arquitectura MVC • Integrar frameworks	Cliente Analista Programador Arquitecto de software
Diseñar diagrama de despliegue	Diagrama de despliegue	• Entrevistas • Características del negocio • Utilizar modelo cliente servidor	

A.6 Diseño Detallado

Tabla Nº A.6 Diseño detallado (Porras, 2011)

Tarea	Artefacto	Técnica	Responsables
Dividir modelo de dominio actualizado para cada caso de uso	Parte de modelo de dominio actualizado	• Coincidir las clases entidad del diagrama de robustez con parte del modelo de dominio actualizado y dibujado	Diseñador
Dibujar un diagrama de secuencia para cada caso de uso	Diagrama de secuencia	• Copiar la descripción del caso de uso • Copiar objetos entidad, interfaz y actores del diagrama de robustez • Verificar que un mensaje del diagrama de secuencia es verbo en el caso de uso • Hacer refactoring al diagrama de secuencia antes de codificar	Programador Diseñador
Actualizar el diagrama de clases de un caso de uso	Diagrama de clases	• Asignar operaciones a las clases a partir de mensajes del diagrama de secuencia • Establecer multiplicidad en las clases	

		• Depurar las clases, operaciones y atributos del diagrama de clases	
Extraer controladores *para pruebas* unitarias	Lista de controladores	• Identificar controladores para la lógica del negocio desde un diagrama de robustez	

A.7 Revisión Crítica de Diseño

Tabla Nº A.7 Revisión crítica de diseño (Porras, 2011)

Tarea	Artefacto	Técnica	Responsables
Revisar diagrama de secuencia	Diagrama de secuencia	• Verificar que el diagrama de secuencia coincide con la descripción del casos de uso • Verificar que el diagrama de secuencia representa los cursos básicos y alterno • Verificar en los mensajes que los atributos y valores de retorno son correctos	Programador Diseñador

Revisar diagrama de clases	Diagrama de clases	• Verificar que el nombre, atributos y operaciones se asignaron correctamente a las clases • Asignar requisitos no funcionales a los casos de uso y clases	
Revisar modelo de dominio actualizado	Modelo de dominio actualizado	• Verificar nombres y atributos del modelo dominio actualizado	
Revisar lista de pruebas unitarias	Lista de controladores	• Actualizar la lista de controladores	

ANEXO B

DEFINICIÓN DE TÉRMINOS BÁSICOS.

Ficha Bibliográfica	
Autor(a): Gregor Kiczales	**Editorial:** Springer-Verlag
Título: Programación orientada a aspectos	**Ciudad, país:** Alemania
Año: 1997	
Resumen del contenido: La programación orientada a aspectos es una nueva técnica de programación que permite expresar programas que involucran aspectos, incluyendo un aislamiento apropiado, composición y reutilización de cada aspecto.	
Número de edición o Impresión: Primera edición	

Ficha Bibliográfica	
Autor(a): Luis Joyanes Aguilar	**Editorial:** McGraw-Hill

Título: Programación orientada a objetos	**Ciudad, país:** España
Año: 2003	
Resumen del contenido: La programación orientada a objetos es un método de implementación donde los programas se organizan como colecciones cooperativas de objeto el cual representa una instancia de una clase, las clases son miembros de una jerarquía de clases unidas por una relación de herencia.	
Número de edición o Impresión: 3ª Edición	

Ficha Bibliográfica	
Autor(a): APA	**Editorial:** Indecopi
Título: Mantenibilidad del producto software	**Ciudad, país:** Lima, Perú
Año: 2006	
Resumen del contenido: La mantenibilidad es el conjunto de atributos que soporta el esfuerzo necesario para realizar modificaciones especificadas de una aplicación.	
Número de edición o Impresión:	

Ficha Bibliográfica	
Autor(a): Baigorria L., Montejano G. y Riesco D.	**Editorial:**
Título: Puntos de enlace	**Ciudad, país:** Argentina
Año: 2006	
Resumen del contenido: Un punto de enlace es la interfaz entre los aspectos y los módulos del lenguaje de componentes, es decir, es un lugar del código en el que se puede aumentar comportamientos adicionales agregados en un aspecto.	
Número de edición o Impresión:	

Ficha Bibliográfica	
Autor(a): Baigorria L., Montejano G. y Riesco D.	**Editorial:**
Título: Relación entre aspectos y clases	**Ciudad, país:** Argentina
Año: 2006	
Resumen del contenido: Se refiere a que un aspecto puede relacionarse con una o varias clases. Esta relación es la que muestra que una clase se ve afectada por dicho aspecto.	
Número de edición o Impresión:	

Ficha Bibliográfica	
Autor(a): Pressman, R.	**Editorial:** McGraw-Hill
Título: Cohesión	**Ciudad, país:** Aravaca, España
Año: 2002	
Resumen del contenido: La cohesión es una indicación cualitativa del grado que tiene un módulo para centrarse en una sola cosa.	
Número de edición o Impresión: 5ª Ed.	

ANEXOC

DESARROLLO USANDO ICONIX

C.1 REQUERIMIENTOS DEL SOFTWARE PARA LLEVAR LA CONTABILIDAD DE LA INSTITUCIÓN NUEVA ACRÓPOLIS.

C.1.1 Requerimientos

a. Requerimientos Funcionales

Tabla Nº C.1 Requisitos funcionales

N°	REQUISITOS FUNCIONALES
01	El **usuario** debe acceder al software mediante un nombre de usuario y clave.
02	El administradpr debe ser capaz de crear una cuenta para usuarios previa solicitud de acceso y también podrá modificarla.
03	El software debe permitir registrar **facturas, boletas** y otros **documentos** contables de acuerdo a la **pperación contable** como: **Compras. Ventas, Notas de Crédito, Notas de Débito,** etc.
04	El software debe permitir seleccionar una operación contable como: compras, ventas, caja ingreso, caja egreso, ingreso a almacén, etc.

Ficha Bibliográfica	
Autor(a): Pressman, R.	**Editorial:** McGraw-Hill
Título: Cohesión	**Ciudad, país:** Aravaca, España
Año: 2002	
Resumen del contenido: La cohesión es una indicación cualitativa del grado que tiene un módulo para centrarse en una sola cosa.	
Número de edición o Impresión: 5ª Ed.	

ANEXO C

DESARROLLO USANDO ICONIX

C.1 REQUERIMIENTOS DEL SOFTWARE PARA LLEVAR LA CONTABILIDAD DE LA INSTITUCIÓN NUEVA ACRÓPOLIS.

C.1.1 Requerimientos

a. Requerimientos Funcionales

Tabla N° C.1 Requisitos funcionales

N°	REQUISITOS FUNCIONALES
01	*El* ***usuario*** debe acceder al software mediante un nombre de usuario y clave.
02	El administrador debe ser capaz de crear una cuenta para usuarios previa solicitud de acceso y también podrá modificarla.
03	El software debe permitir registrar **facturas, boletas** y otros **documentos** contables de acuerdo a la **operación contable** como: **Compras. Ventas, Notas de Crédito, Notas de Débito**, etc.
04	El software debe permitir seleccionar una operación contable como: compras, ventas, caja ingreso, caja egreso, ingreso a almacén, etc.

05	El software debe permitir seleccionar una **empresa**, **ejercicio** y **periodo** de la operación contable.
06	El software debe permitir registrar las **cuentas contables**.
07	El software debe permitir registrar el **Beneficiario** (Proveedor).
08	El software debe permitir registrar una **Sede** ubicado en algún punto del país.
09	El software debe permitir registrar los tipos de documentos como: Factura, Boleta de venta, Nota de Crédito, Nota de Débito, Carta de porte aéreo, póliza de adjudicación, etc.
10	El software debe permitir al usuario emitir registro de compras.
11	El software debe permitir al usuario emitir registro de ventas.

b. Requisitos no funcionales

Tabla N° C.2 Requisitos no funcionales

N°	REQUISITOS NO FUNCIONALES
01	El software debe ser una aplicación web.
02	El sistema debe mostrar una interfaz amigable y de fácil uso.
0 3	El sistema debe ser seguro y confiable, tener buen soporte de información y de acceso rápido.

04	El sistema debe tener la capacidad de crecer y evolucionar con el avance tecnológico.
05	El software debe ser fácil de mantener, ante un error.
06	El sistema debe restringir el acceso de algunos usuarios a ciertas funcionalidades internas, que son manejadas por los usuarios autorizados.

C.1.2 Listado de candidatos a clases de dominio a partir de los requerimientos del software

1. Usuario
2. Facturas
3. Boletas
4. Documentos
5. Operación contable
6. Compras
7. Ventas
8. Notas de Crédito
9. Notas de Débito
10. Empresa
11. Ejercicio
12. Periodo
13. Cuentas contables
14. Beneficiario
15. Sede

C.1.3 Glosario de términos depurado

1. Usuario
2. Factura
3. Boleta
4. Documento
5. Operación contable
6. Compra
7. Venta
8. Notas de Crédito
9. Notas de Débito
10. Empresa

11.Periodo
12. Cuentas contables
13. Beneficiario
14.Sede

C.1.4 Modelo de dominio inicial

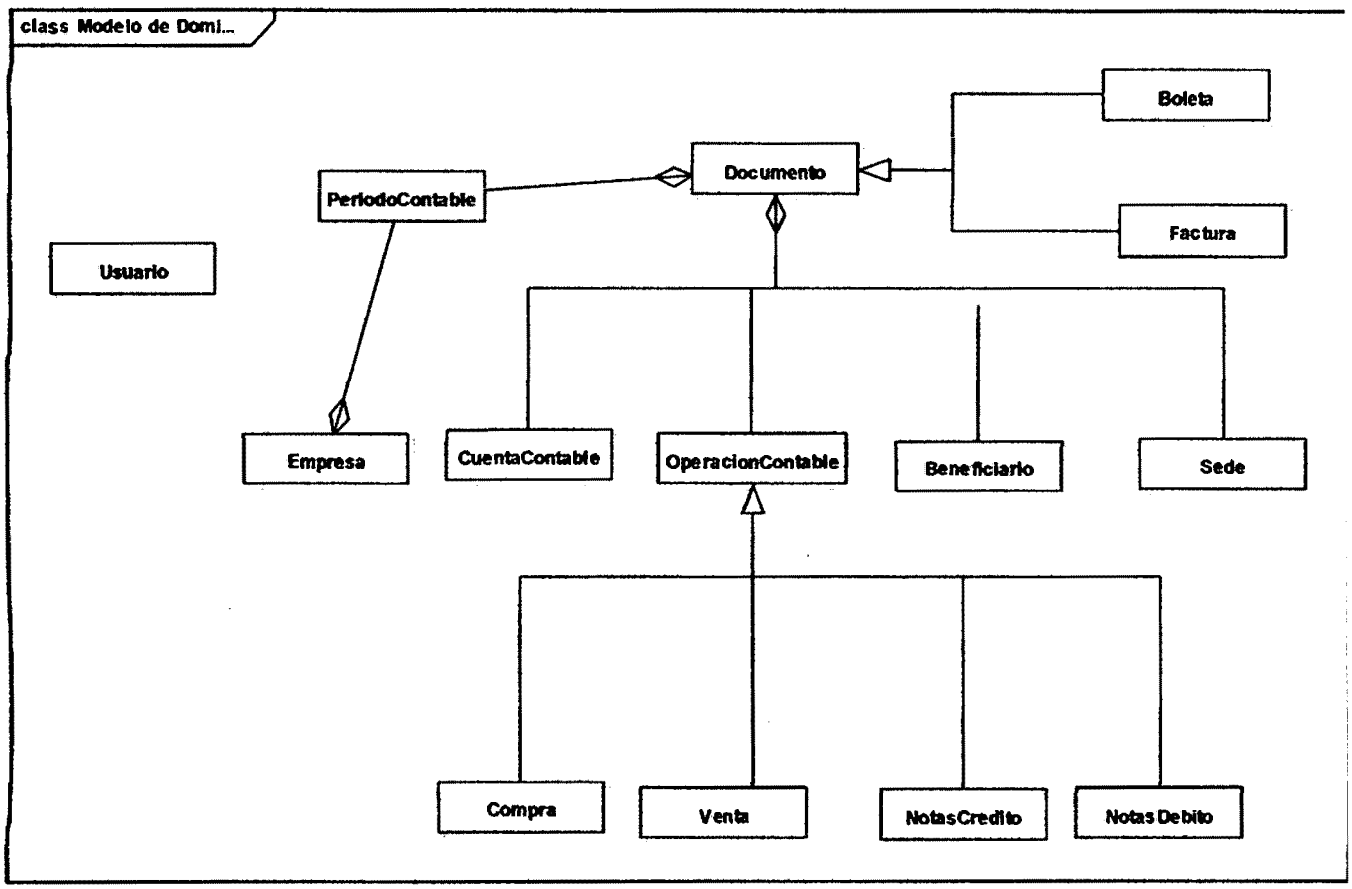

Figura N° C.1 Modelo de dominio inicial, software Contabilidad Nueva Acrópolis

:.2 MODELO DE CASOS DE USO

:.2.1 Identificación de actores a partir de los requerimientos funcionales

a. Administrador
b. Contador
c. Auxiliar contable

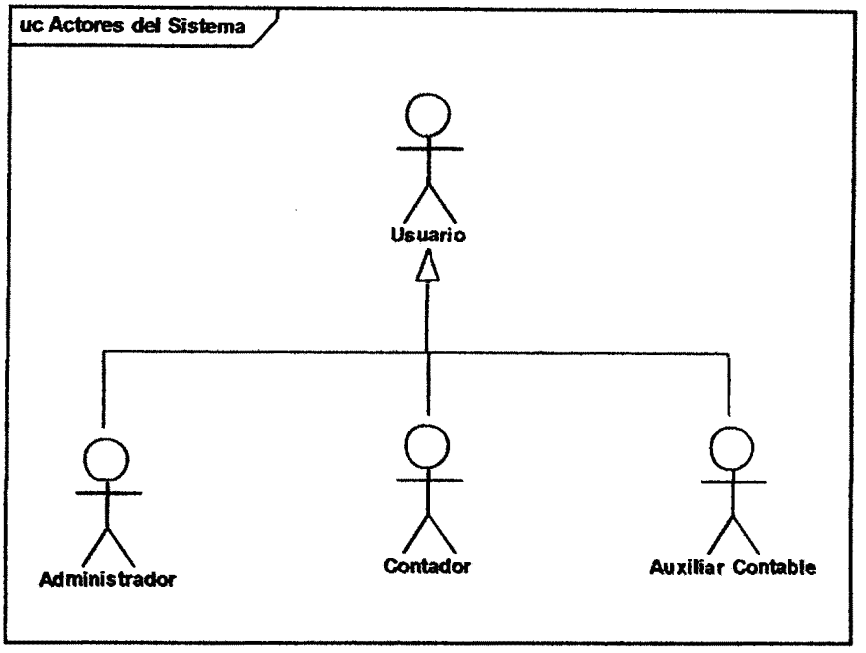

Figura N° C.2 Actores o usuarios del sistema, software Contabilidad Nueva Acrópolis

C.2.2 Identificación de casos de uso a partir de los requerimientos funcionales:

Tabla N° C.3 Casos de uso por requerimientos funcionales, software Contabilidad Nueva Acrópolis

N° Req	REQUISTOS FUNCIONALES	CASOS DE USO
1.	El usuario debe acceder al software mediante un nombre de usuario y clave.	CU. 1 Iniciar sesión
2.	El administrador debe ser capaz de crear una cuenta para usuarios previa solicitud de acceso y también podrá modificarla	CU. 2 Registrar cuenta CU. 3 Actualizar cuenta
3.	El software debe permitir registrar facturas, boletas y	CU. 4 Registrar Documento CU. 5 Actualizar Documento

	otros documentos contables de acuerdo a la operación contable como: Compras. Ventas, Notas de Crédito, Notas de Débito, etc.cursos de reforzamiento	
4.	El software debe permitir seleccionar una operación contable como: compras, ventas, caja ingreso, caja egreso, ingreso a almacén, etc.	CU. 6 Seleccionar Operación contable
5.	El software debe permitir seleccionar una empresa, ejercicio y periodo de la operación contable.	CU. 7 Seleccionar Parámetros iniciales del sistema
6.	El software debe permitir registrar las cuentas contables.	CU. 8 Registrar Cuenta Contable CU. 9 Actualizar Cuenta Contable
7.	El software debe permitir registrar el Beneficiario (Proveedor).	CU. 10 Registrar Beneficiario CU. 11 Actualizar Beneficiario
8.	El software debe permitir registrar una Sede ubicado en algún punto del país.	CU. 12 Registrar Sede CU. 13 Actualizar Sede
9.	El software debe permitir registrar los tipos de	CU. 14 Registrar Tipo documento CU. 15 Actualizar Tipo documento

	documentos como: Factura, Boleta de venta, Nota de Crédito, Nota de Débito, Carta de porte aéreo, póliza de adjudicación, etc.	
10.	El software debe permitir al usuario emitir registro de compras.	CU. 16 Emitir Reporte de Compras
11.	El software debe permitir al usuario emitir registro de ventas.	CU. 17 Emitir Reporte de Ventas

2.2.3 Listado de casos de uso

Tabla N° C.4 Listado de casos de uso, software Contabilidad Nueva Acrópolis

CU. 1 Iniciar sesión
CU. 2 Registrar cuenta CU. 3 Actualizar cuenta
CU. 4 Registrar Documento CU. 5 Actualizar Documento
CU. 6 Seleccionar Operación contable
CU. 7 Seleccionar Parámetros iniciales del sistema
CU. 8 Registrar Cuenta Contable CU. 9 Actualizar Cuenta Contable
CU. 10 Registrar Beneficiario CU. 11 Actualizar Beneficiario
CU. 12 Registrar Sede CU. 13 Actualizar Sede
CU. 14 Registrar Tipo documento

CU. 15 Actualizar Tipo documento
CU. 16 Emitir Reporte de Compras
CU. 17 Emitir Reporte de Ventas

C.2.4 Empaquetamiento de casos de uso

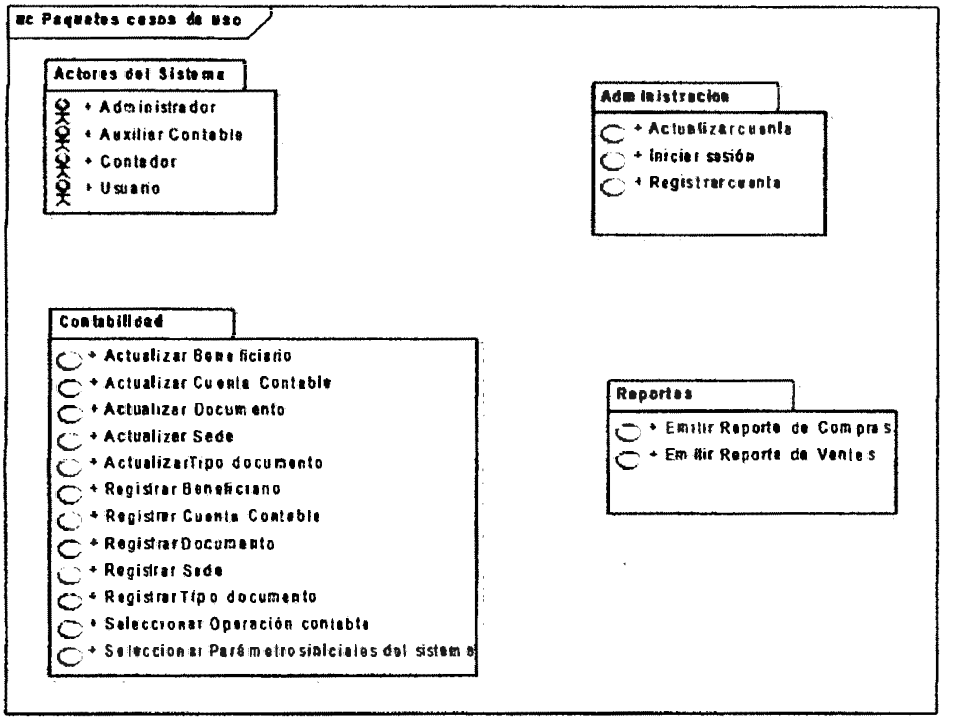

Figura Nº C.3 Casos de uso empaquetados, software Contabilidad Nueva Acrópolis

a. Paquete Administración

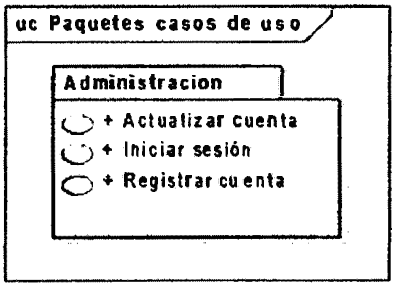

Figura Nº C.4 Paquete administración, software Contabilidad Nueva Acrópolis

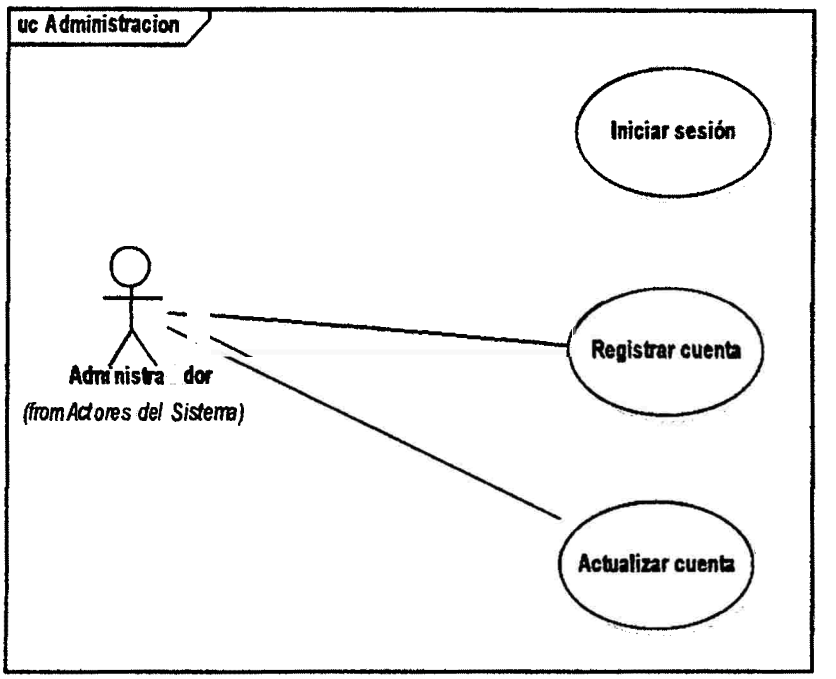

Figura N° C.5 Casos de uso del paquete administración, software Contabilidad Nueva Acrópolis

b. Paquete Contabilidad

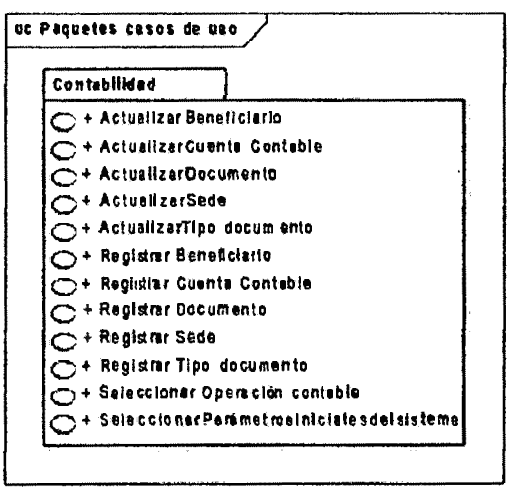

Figura N° C.6 Paquete contabilidad, software Contabilidad Nueva Acrópolis

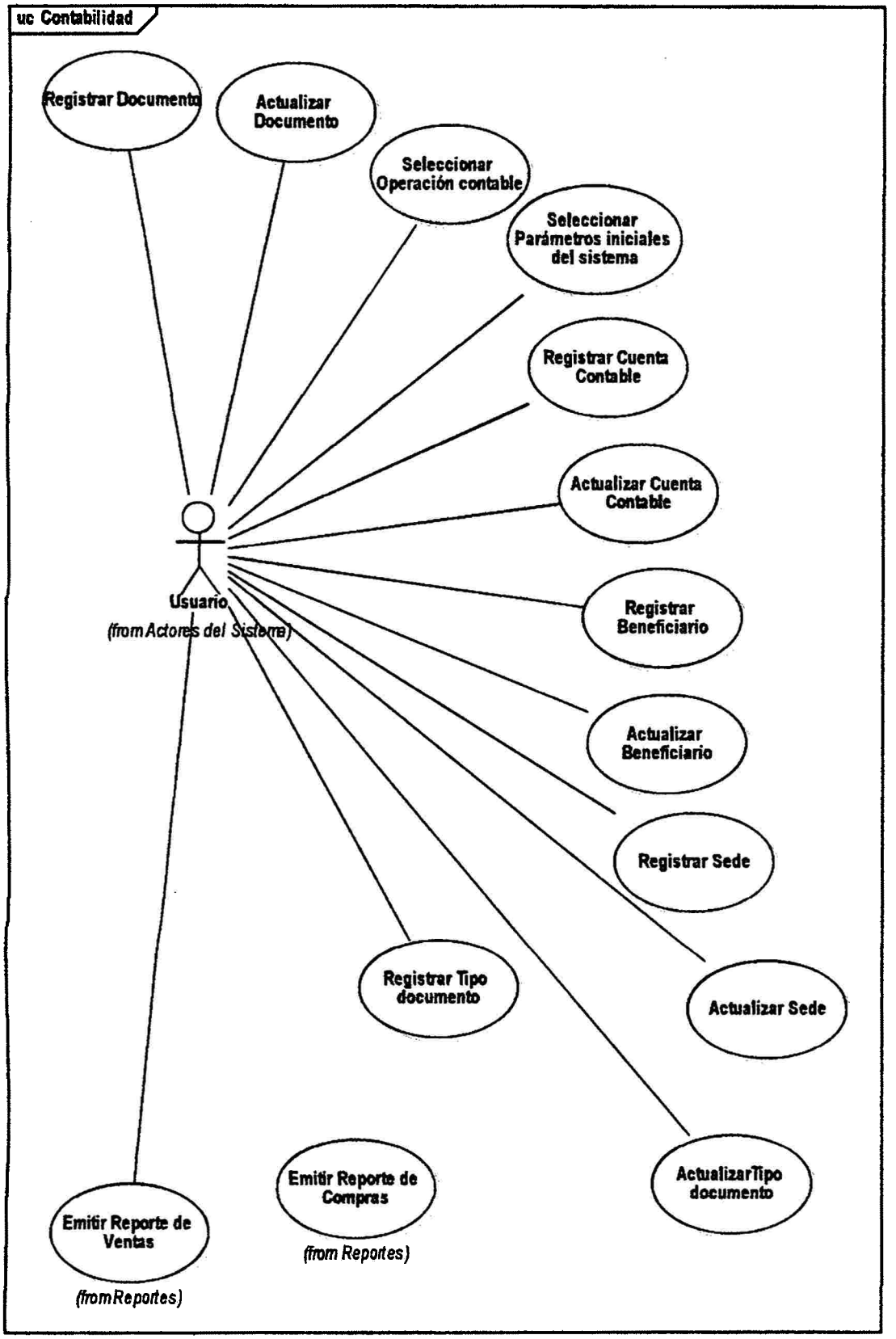

Figura Nº C.7 Casos de uso del paquete contabilidad, software Contabilidad Nueva Acrópolis

c. Paquete Reportes

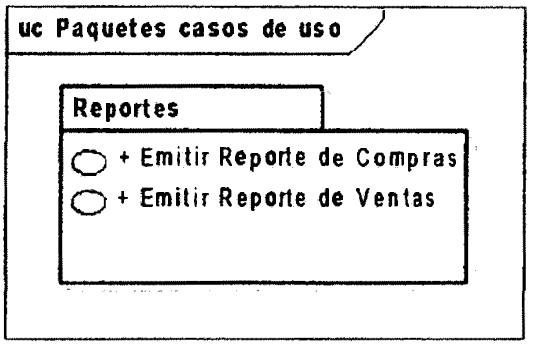

Figura Nº C.8 Paquete reportes, software Contabilidad Nueva Acrópolis

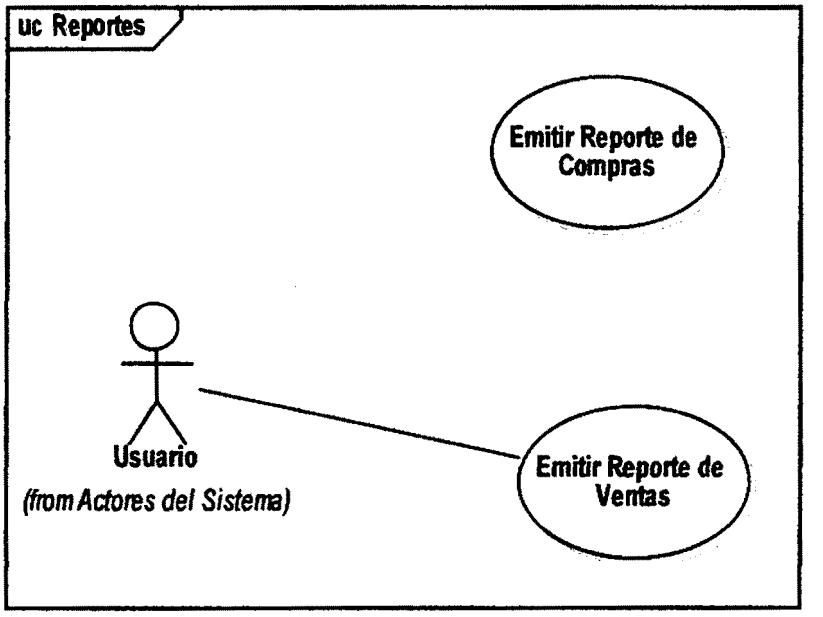

Figura Nº C.9 Casos de uso del paquete reportes, software Contabilidad Nueva Acrópolis

C.2.5 Primer Borrador de Casos de Uso

CU.4 Registrar Documento

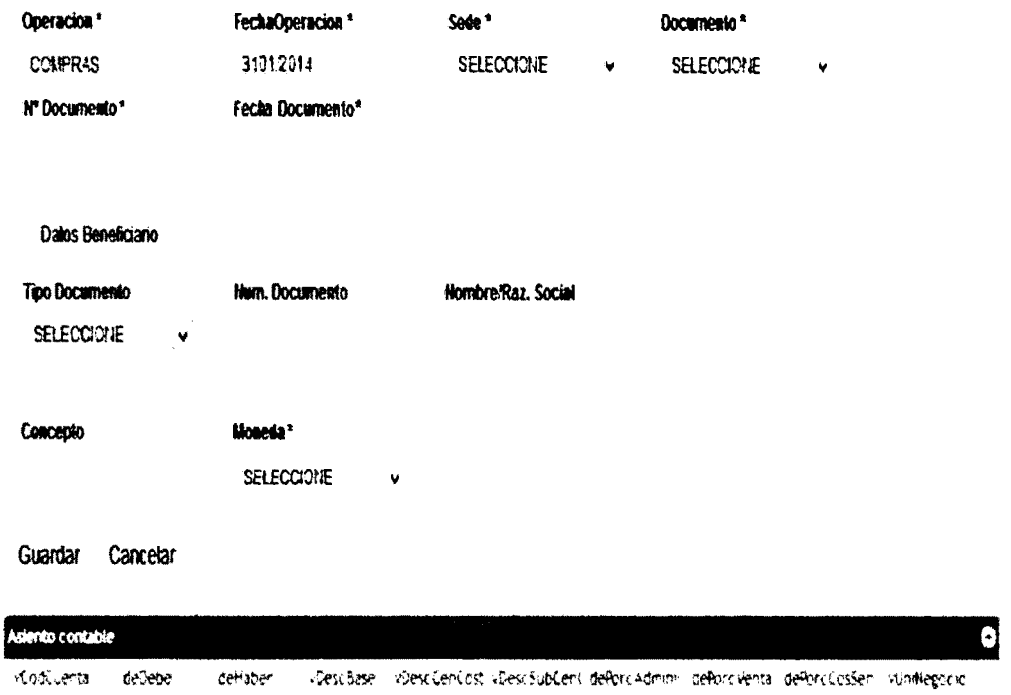

Figura N° C.10 Interfaz del caso de uso Registrar Documento, software Contabilidad Nueva Acrópolis

CURSO BÁSICO:

1. El usuario cliquea dos veces sobre una operación contable en la interfaz SeleccionarOperacion, el sistema carga la fecha de operación, una lista de sedes y una lista de tipos de documentos y muestra la interfaz MantenerDocumento.
2. El usuario introduce datos del documento como: sede, tipo documento, N° documento, fecha documento, datos de beneficiario, concepto, tipo de moneda y el importe total, el sistema precarga los datos de asientos

contables con el importe total.

3. El usuario realiza el balance correspondiente con los asientos contables y hace clic en el botón Guardar, el sistema guarda los datos ingresados en la base de datos y muestra el mensaje "Datos Registrados Correctamente".

CURSO ALTERNO:

1. No se ingresaron campos obligatorios, el sistema muestra un mensaje error.
2. La fecha de documento es mayor que la fecha de operación, el sistema muestra un mensaje de error.
3. El usuario desea cancelar el registro del documento, el usuario cliquea en el botón Cancelar, el sistema limpia los campos de texto.

C.3 REVISIÓN DE REQUISITOS

Se revisará el modelo de dominio inicial, se diseñará las GUI'S final y se reescribirán los borradores de los casos de uso.

C.3.1 Revisión de la interfaz gráfica de los casos de uso críticos

Rediseñando las interfaces gráficas y reescribiendo el borrador de casos de uso.

CU.4 Registrar Documento

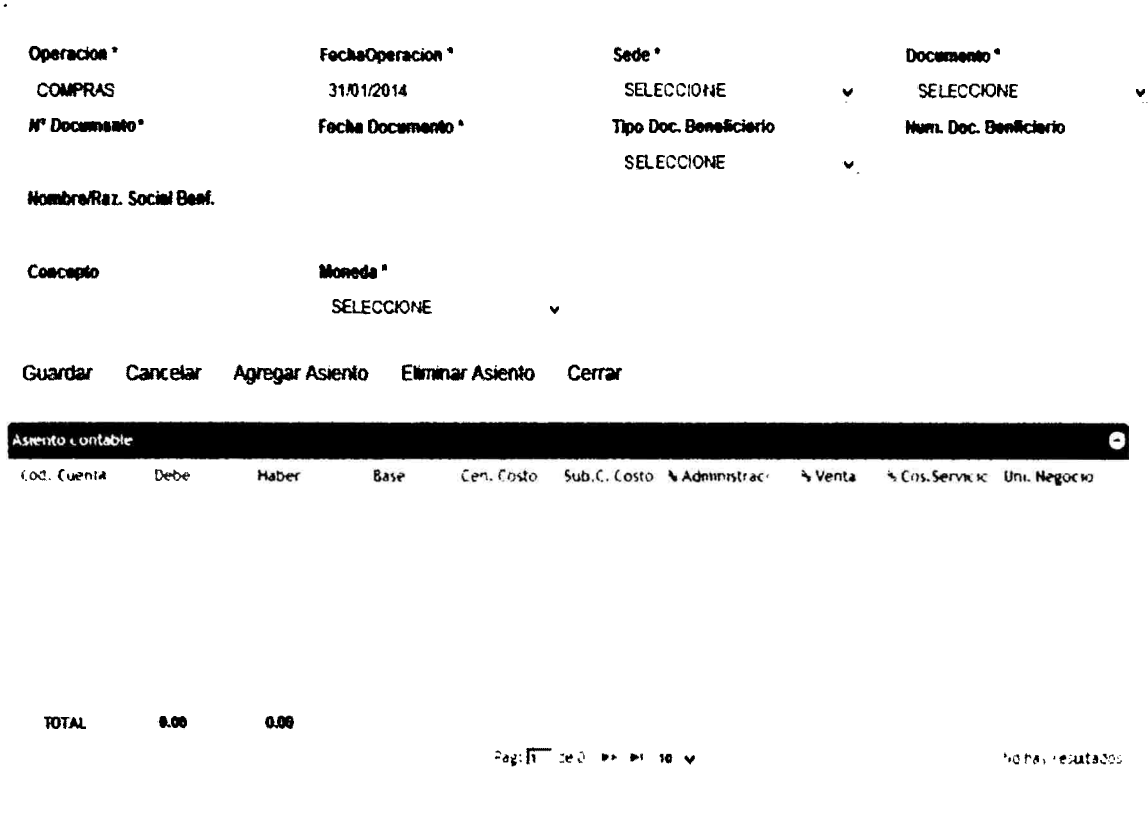

Figura N° C.11 Interfaz final del caso de uso Registrar Documento, software Contabilidad Nueva Acrópolis

CURSO BÁSICO:

1. El usuario hace clic en el botón "Agregar Documento" el sistema carga la fecha de operación, una lista de sedes y una lista de tipos de documentos y muestra la interfaz MantenerDocumento.
2. El usuario introduce datos del documento como: sede, tipo documento, N° documento, fecha documento, datos de beneficiario, concepto, tipo de moneda y el importe total, el sistema precarga los datos de asientos

contables con el importe total.

3. El usuario realiza el balance correspondiente con los asientos contables y hace clic en el botón Guardar, el sistema guarda los datos ingresados en la base de datos y muestra el mensaje "Datos Registrados Correctamente".

CURSO ALTERNO:

1. **Campos obligatorios no ingresados:**
   El sistema muestra una pantalla con el mensaje "Debe ingresar los campos obligatorios".
2. **Fecha documento mayor que la fecha de operación:** El sistema muestra una pantalla con el mensaje "La fecha de Documento no puede ser mayor que la fecha de operación".
3. El usuario desea cancelar el registro del documento, el usuario cliquea en el botón Cancelar, el sistema limpia los campos de texto.

C.3.2 Relacionando requisitos funcionales con casos de uso críticos

Tabla N° C.5 Caso de uso escogido relacionado a su requisito funcional, software Contabilidad Nueva Acrópolis

N° Req	REQUISTOS FUNCIONALES	CASOS DE USO
3	El software debe permitir registrar facturas, boletas y otros documentos contables de acuerdo a la operación contable como: Compras. Ventas, Notas de Crédito, Notas de Débito, etc.cursos de reforzamiento	CU. 4 Registrar Documento

C.6 ARQUITECTURA TÉCNICA

A. El Patrón Arquitectónico MVC (Modelo Vista Controlador)

Se usara este patrón en la implementación del software, debido a que permite una implementación de manera ordenada, es decir orden y sencillez en la codificación. El mantenimiento del mismo también se hace más sencillo.

En el patrón MVC se diferencian bien tres capas (modelo-vista-controlador), el controlador es el intermediario entre la vista y el modelo, el modelo representa la lógica del negocio e intermediario con la base de datos y la vista (ventanas) es el encargado de mostrar la información al usuario de manera gráfica.

B. Arquitectura por capas

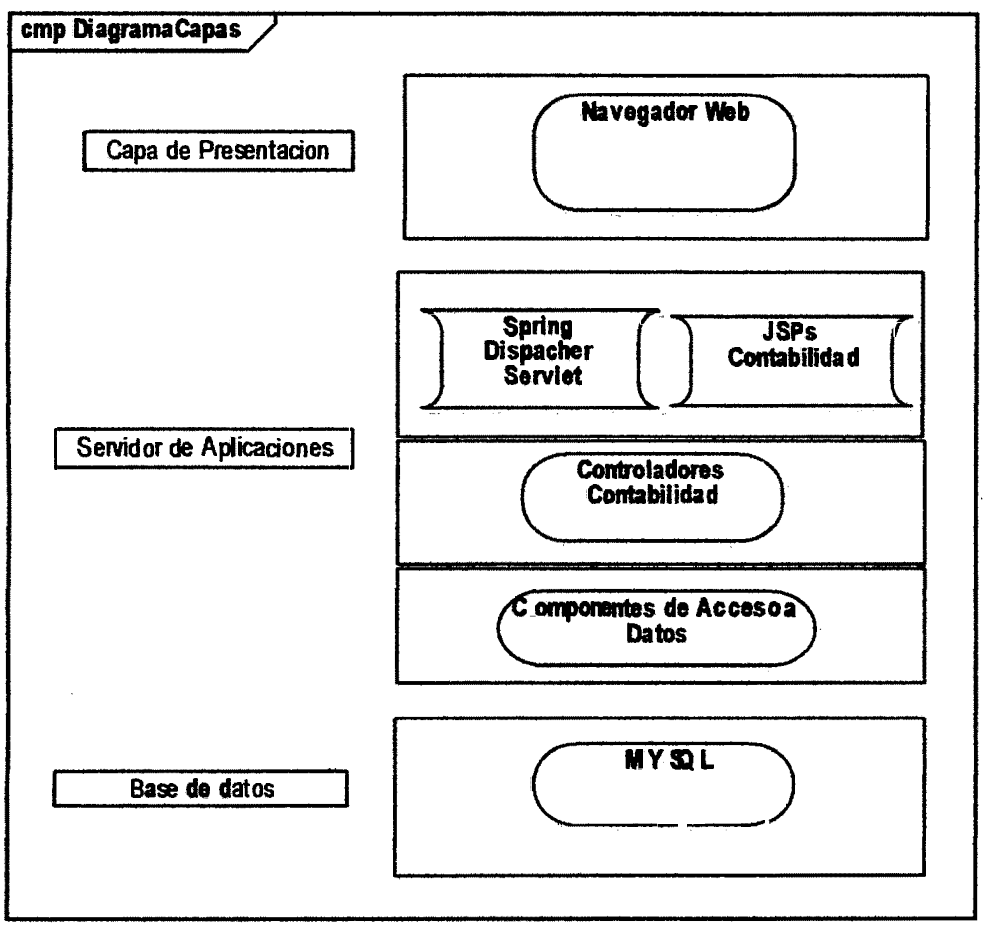

Figura N° C.17 Arquitectura técnica inicial, software Contabilidad Nueva Acrópolis

C. Diagrama de Componentes

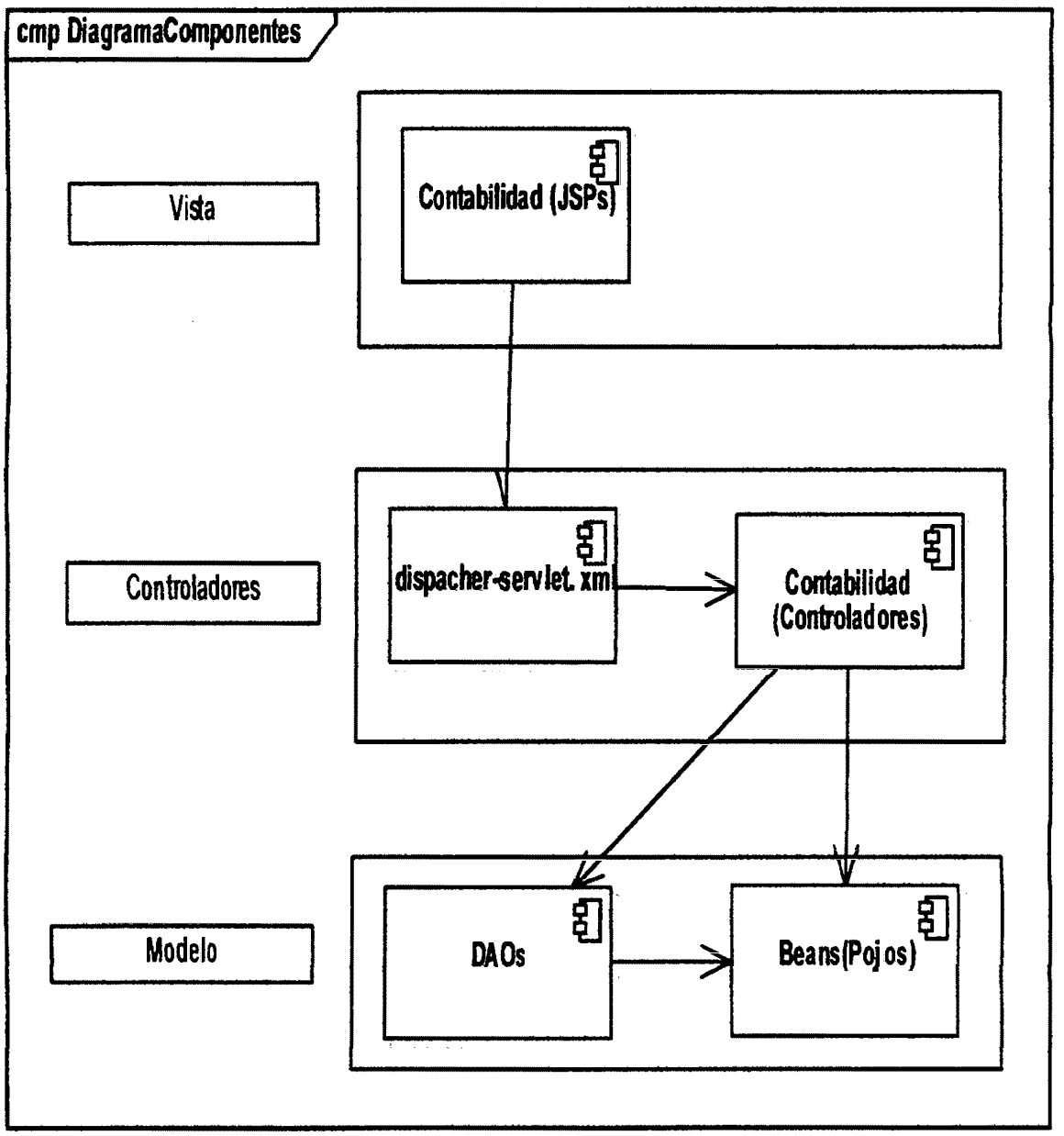

Figura N° C.18 Diagrama de componentes, software Contabilidad Nueva Acrópolis

D. Diagrama de Despliegue

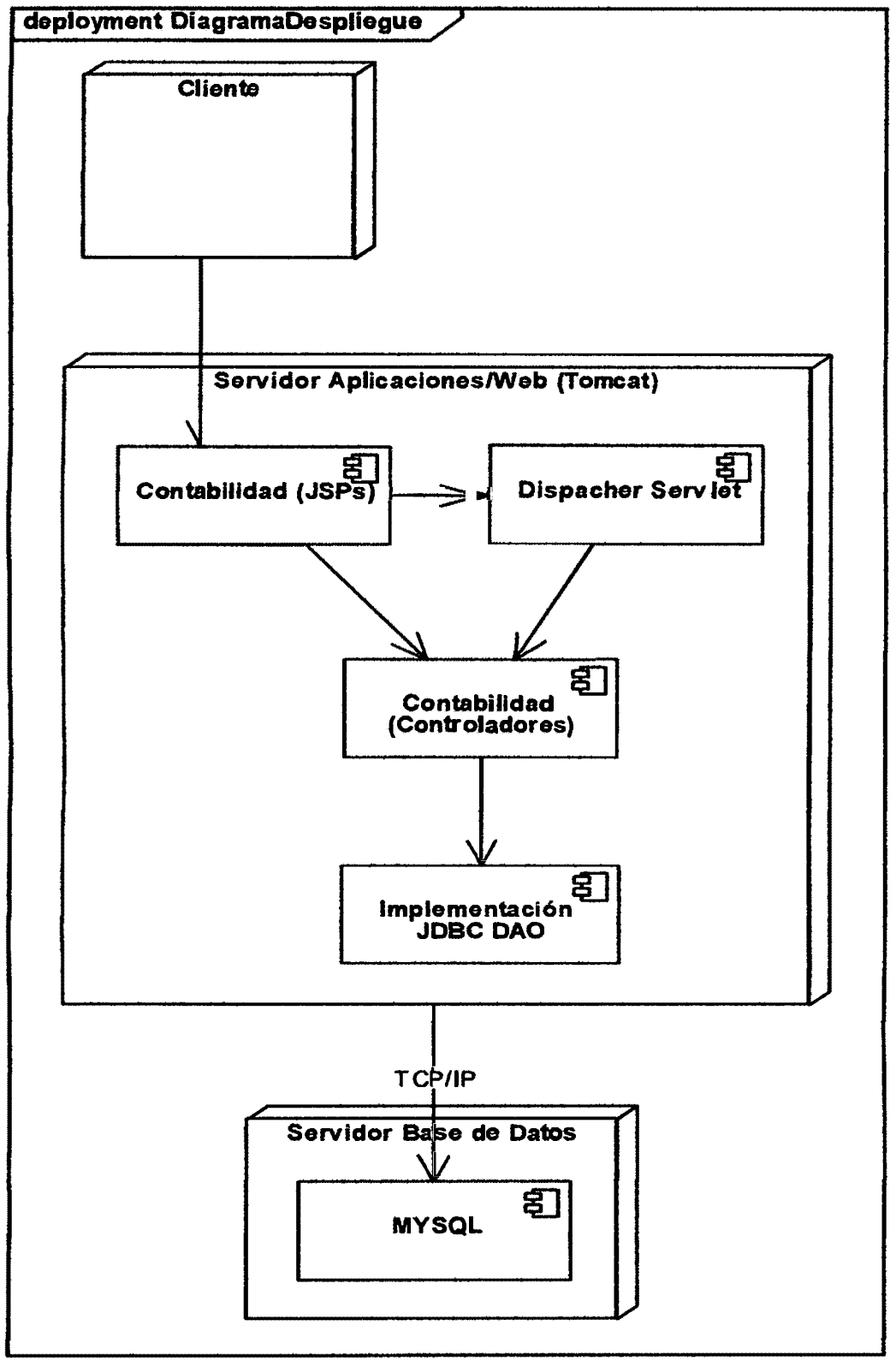

Figura Nº C.19 Diagrama de despliegue, software Contabilidad Nueva Acrópolis