

UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN

Facultad de Ingeniería

Escuela Profesional de Ingeniería en Informática y Sistemas

**ANÁLISIS COMPARATIVO DE LOS PARADIGMAS
DE LA PROGRAMACIÓN, ORIENTADA A OBJETOS
Y PROGRAMACIÓN ORIENTADA A ASPECTOS
EN LA CONSTRUCCIÓN DEL MÓDULO DE
AUTENTICACIÓN DEL SOFTWARE
DE LA BIBLIOTECA DE LA LIGA
DEPARTAMENTAL DE FÚTBOL
DE TACNA - 2015**

TESIS

Presentada por:

Bach. Ciro Marcel Cohaila Lara

Para optar el Título Profesional de:

INGENIERO EN INFORMÁTICA Y SISTEMAS

TACNA – PERÚ

2020

UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN
FACULTAD DE INGENIERÍA
JURADO CALIFICADOR Y CALIFICACIÓN DE LA SUSTENTACIÓN DE TESIS

TESIS N° _____

TÍTULO PROFESIONAL DE
Ingeniero en Informática y Sistemas

La Secretaría Académica Administrativa de la Facultad de Ingeniería, por resolución de Facultad N° 03237-2015-FAIN/UNJBG, designó jurado para la sustentación oral de la tesis titulada ANÁLISIS COMPARATIVO DE LOS PARADIGMAS DE LA PROGRAMACIÓN, ORIENTADA A OBJETOS Y LA PROGRAMACIÓN ORIENTADA A ASPECTOS EN LA CONSTRUCCIÓN DEL MODULO DE AUTENTICACIÓN DEL SOFTWARE DE LA BIBLIOTECA DE LA LIGA DEPARTAMENTAL DE FUTBOL DE TACNA – 2015, año 2015.

El mismo está conformado por:

Presidente: Dr. ERBERT FRANCISCO OSCO MAMANI

Secretario: Dr. EDWIN ANTONIO HINOJOSA RAMOS

Vocal: Mgtr. GIANFRANCO ALEXEY MÁLAGA TEJADA

Para calificar la sustentación de la Tesis en acto público el día 10 de diciembre del 2015.

Presentado por el bachiller Ciro Marcel Cohaila Lara de la Escuela Académico Profesional de Ingeniería en Informática y Sistemas.

El Jurado Calificador en forma secreta e individual emitió un calificativo sobre el tema de la tesis expuesta y procedió a obtener el promedio que arrojó el calificativo aprobado con la nota de 16 (Dieciséis).

Para ratificar lo detallado firman:



Dr. Erbert Francisco Osco Mamani
Presidente



Dr. Edwin Antonio Hinojosa Ramos
Secretario



MSC. Gianfranco Alexey Málaga Tejada
Vocal

UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN
FACULTAD DE INGENIERÍA
ESCUELA PROFESIONAL DE INGENIERÍA EN INFORMÁTICA Y SISTEMAS

**"ANÁLISIS COMPARATIVO DE LOS PARADIGMAS DE LA
PROGRAMACIÓN ORIENTADA A OBJETOS Y LA PROGRAMACIÓN
ORIENTADA A ASPECTOS EN LA CONSTRUCCIÓN DEL MÓDULO DE
AUTENTICACIÓN DEL SOFTWARE DE LA BIBLIOTECA DE LA LIGA
DEPARTAMENTAL DE FUTBOL DE TACNA - 2015".**

TESIS PRESENTADA A LA COMISIÓN REVISADORA Y APROBADA POR EL
JURADO CALIFICADOR, INTEGRADA POR:

Presidente



Dr. Erbert Francisco Osco Mamani

Secretario



Dr. Edwin Antonio Hinojosa Ramos

Vocal



MSC. Gianfranco Alexey Málaga Tejada

Asesor



Dra. Karín Yanet Supo Gavancho

DEDICATORIA

Un día te veré, tan solo estas durmiendo yo lo sé.

Un día te veré y no más despedidas esta vez.

Te reconoceré, te abrazaré,

y al lado de tu cuello sé que llorare.

Me mirarás, y reirás y me dirás, ¡no muerte ya!

Disculpa por haberme demorado tanto Aldo.

De tu hermano que nunca te olvidará: Ciro Marcel.

AGRADECIMIENTO

A la Universidad Nacional Jorge Basadre Grohmann, a mis docentes por todos los conocimientos impartidos, a las grandes amistades que hice a lo largo de mis años de estudio y a las cuatro mujeres más importantes de mi vida, porque no me equivoco al decir que son las mejores madres del mundo y marcaron mi vida de una manera muy especial, cuando las palabras sobran solo puedo mirarlas en fotos y pensar en un “**Gracias**”, por darme la vida, por haberse quitado el pan de la boca para darme a mí, por darme el milagro de traer una nueva vida a este mundo y lo más importante es por haber sacrificado parte de su vida para darme vida a mí, desde el cielo me ven dos de ustedes y aun no me acostumbro al dolor, pero no dejo de agradecer a Dios por ser tan afortunado al haberlas visto sonreír.

Gracias María, Zoila, Dora y Yeny.

ÍNDICE DE CONTENIDO

	Pág.
DEDICATORIA	iv
AGRADECIMIENTO	v
RESUMEN	xii
SUMMARY	1
INTRODUCCIÓN	2
CAPÍTULO I PLANTEAMIENTO DE LA INVESTIGACIÓN	4
1.1 Descripción del problema	4
1.1.1 Antecedentes del problema	4
1.1.2 Problemática de la investigación	8
1.2 Formulación del problema	10
1.2.1 Problema general	10
1.2.2 Problemas específicos	10
1.3 Justificación	11
1.4 Alcances y limitaciones	12
1.5 Objetivos	12
1.5.1 Objetivo general	12
1.5.2 Objetivos específicos	13
1.6 Hipótesis	13
1.6.1 Hipótesis global	13
1.6.2 Sub hipótesis	13
1.7 Variables	14

1.7.1	Identificación de variables	14
1.7.2	Definición de variables	14
1.7.3	Operacionalización de variables	15
1.8	Diseño de la investigación	16
1.8.1	Diseño experimental o no experimental	16
1.8.2	Población y muestra	18
1.8.3	Técnicas e instrumentos para la recolección de datos	20
1.8.4	Análisis de datos	24
1.8.5	Selección de pruebas estadísticas	25
CAPÍTULO II MARCO TEÓRICO		26
2.1	Marco referencial	26
2.2	Bases teóricas respecto al problema	27
2.2.1	Programación orientada a objetos	27
2.2.2	Programación orientada a aspectos	37
2.2.3	Módulo de autenticación y diseño de software	54
CAPÍTULO III DESARROLLO		58
3.1	Descripción del caso de estudio	58
3.2	Diseño de un software orientado a objetos	58
3.2.1	Requerimientos	59
3.2.2	Análisis	63
3.2.3	Diseño	68
3.2.4	Implementación	73
3.3	Diseño de un software orientado a aspectos	75

3.3.1	Requerimientos	75
3.3.2	Análisis	77
3.3.3	Diseño	80
3.3.4	Implementación	84
CAPÍTULO IV RESULTADOS		89
4.1	Resultados obtenidos de la prueba del tamaño (LCD).	89
4.2	Resultados obtenidos de la prueba complejidad ciclomática	93
4.3	Resultados obtenidos de la eficiencia computacional (uso de memoria RAM)	98
DISCUSIONES		104
CONCLUSIONES		106
RECOMENDACIONES		108
REFERENCIAS BIBLIOGRÁFICAS		109
ANEXOS		114
ANEXO 01: Interfaz gráfica del módulo de autenticación y mensajes de error.		114
ANEXO 02: Realización de las pruebas estadísticas		117
ANEXO 03: Instrumento utilizado para la recolección de datos. Ficha de registro de los indicadores		120
ANEXO 04: Matriz de consistencia.		122

ÍNDICE DE FIGURAS

	Pág.
Figura 01: Conceptos que involucra una aplicación orientada a aspectos	38
Figura 02: Incumbencias o intereses en un sistema	39
Figura 03: Interacción entre aspectos y funcionalidad básica	41
Figura 04: Separación de intereses	43
Figura 05: Implementación de incumbencias transversales sin y con aspectos	44
Figura 06: Función del tejedor de aspectos	48
Figura 07: Proceso de tejido	49
Figura 08: Transformación del modelo de análisis al modelo de diseño	55
Figura 09: Caso de uso - autenticar usuario	61
Figura 10: Modelo de análisis - autenticar usuario	64
Figura 11: Diagrama de actividades - autenticar usuario	66
Figura 12: Diagrama de secuencia - autenticar usuario	67
Figura 13: Diagrama de componentes modelo vista controlador	71
Figura 14: Diagrama de despliegue del sistema	72
Figura 15: Diagrama de clases - autenticar usuario	73
Figura 16: Caso de uso - validar usuario con POA	77
Figura 17: Validar usuario	79
Figura 18: Ilustración de la ejecución de los puntos de corte	81

Figura 19: Estructura de elementos del diseño	82
Figura 20: Diagrama de componentes MVC con la inclusión de aspectos	83
Figura 21: Diagrama de despliegue del módulo	83
Figura 22: Tejido de aspectos con las clases base	85
Figura 23: Implementación del módulo de autenticación	86
Figura 24: Aspecto de autenticación	87
Figura 25: Pantalla de errores	88
Figura 26: Cuadro comparativo cantidad de LCD	90
Figura 27: Tamaño de LCD aspectos	91
Figura 28: Resumen comparativo	93
Figura 29: Grafo de flujo del módulo de autenticación	95
Figura 30: Cuadro comparativo complejidad ciclomática	96

ÍNDICE DE TABLAS

	Pág.
Tabla 1: Identificación de variables e indicadores	15
Tabla 2: Población de módulos del software	19
Tabla 3: Muestra de módulos del software	20
Tabla 4: Caso de uso autenticar usuario	62
Tabla 5: Resultados de cuantificar líneas de código	89
Tabla 6: Tamaño de líneas de código aspectos	91
Tabla 7: Cuadro resumen comparativo	92
Tabla 8: Cuadro comparativo de la complejidad ciclomática	96
Tabla 9: Tabla de escalas de la complejidad ciclomática	97
Tabla 10: Hardware para las pruebas de uso de memoria	98
Tabla 11: Software para las pruebas de uso de memoria	98
Tabla 12: Resultados de uso de memoria en diferentes test del modulo	99
Tabla 13: Pruebas de normalidad con SPSS	101
Tabla 14: Prueba de normalidad	102
Tabla 15: Prueba de igualdad de varianza	102
Tabla 16: Prueba T-Student	103

RESUMEN

El presente trabajo tiene por objetivo establecer un análisis comparativo del efecto de la programación orientada a aspectos y la programación orientada a objetos en el diseño e implementación del módulo de autenticación del software de la biblioteca de la liga departamental de futbol de Tacna. En su realización se incluyeron las etapas de desarrollo del software para así de esta forma determinar el efecto de cada tipo de programación en el producto. Se realizaron dos grupos de estudio uno orientado a aspectos y otro grupo orientado a objetos, en los que se desarrollaron actividades de especificación de requerimientos, análisis para llegar al diseño y en última instancia comparar ambos productos finales con referencia a los indicadores, que son la cuantificación de Líneas de Código (LCD), la complejidad ciclomática y los promedios de uso de memoria. Los resultados muestran que el uso de la programación orientada a aspectos es favorable al diseño de un software en comparación con la orientación a objetos.

Palabras claves: Programación orientada a aspectos, Programación orientada a objetos, diseño de un módulo de autenticación.

SUMMARY

The present work aims to establish a comparative analysis of the effect of aspect-oriented programming and object-oriented programming in the design and implementation of the software authentication module of the Tacna departmental soccer league library. In its implementation, the software development stages were included in order to determine the effect of each type of programming on the product. Two study groups were carried out, one oriented to aspects and another group oriented to objects, in which requirements specification activities were developed, analysis to arrive at the design and ultimately comparing both final products with reference to the indicators, which are the Quantification of Lines of Code (LCD), cyclomatic complexity and memory usage averages. The results show that the use of aspect-oriented programming is favorable to software design compared to object-oriented.

Keywords: Aspect Oriented Programming, Object Oriented Programming, Authentication Module Design.

INTRODUCCIÓN

La evolución del diseño de software desde comienzos del desarrollo de software ha ido cambiando según la evolución de la introducción de nuevos retos de organizar la estructura de un sistema, los trabajos sobre diseño se centran en criterios de modularidad, de separación de tareas, de acoplamiento que permita un buen grado de mantenibilidad y evolución para llevar a una buena arquitectura del software de una manera decente.

El motivo que llevó a desarrollar el presente trabajo de investigación fue buscar una mejor forma de acercarse al concepto de diseño de software en la separación de unidades de software, aumento de la realización de tareas específicas en cada subsistema siguiendo el principio de divide y vencerás, brindado de una buena mantenibilidad y baja complejidad, ya que al realizar esto el producto final ofrecerá mejores características en diferentes ámbitos de calidad. Se piensa obtener resultados que generen un punto de partida para futuras investigaciones y reflexión sobre la forma tradicional que se ha venido pensando en la creación de un software, como también ofrecer una alternativa para el desarrollo de software.

Se pretende demostrar que haciendo uso de la programación orientada a aspectos se lograra un mejor diseño comparado con la programación orientada a objetos.

Finalmente se presenta la estructura del trabajo de investigación realizado: En el capítulo uno se describe el problema que enfrenta la etapa del diseño del software, viéndolo como un punto de equilibrio de calidad, así como la búsqueda de la modularización de un sistema en diferentes niveles de competencias, haciendo una descripción en la incidencia de usar ambos enfoques en el diseño del software. En el capítulo dos se realiza un estudio teórico y caracterización de los enfoques de programación, así como la descripción del diseño y las características que debe de tener un buen diseño. En el capítulo tres se realiza el desarrollo de dos casos, uno orientado a aspectos y el otro orientado a objetos. En cada uno se siguen los mismos pasos, difiriendo en la forma de cómo se produce un software desde los requerimientos, análisis, pasando por el diseño hasta el producto final. En el capítulo cuatro se realizan pruebas al módulo para finalmente realizar un análisis comparativo de los datos obtenidos y su respectiva interpretación, identificando los beneficios que puedan traer cada enfoque de programación aplicado. Y por último, en el capítulo final se enuncian las conclusiones obtenidas, así como futuros trabajos que podrían partir sobre este, o dar pasos para futuras discusiones.

CAPÍTULO I

PLANTEAMIENTO DE LA INVESTIGACIÓN

1.1 Descripción del problema

1.1.1 Antecedentes del problema

Haciendo mención de los **antecedentes internacionales** se debe mencionar que la programación orientada a objetos (POO) supuso un gran paso en la ingeniería del software, ya que presentaba un modelo de objetos que parecía encajar de manera adecuada con los problemas reales (Bahit, 2010). La cuestión era saber descomponer de la mejor manera el dominio del problema al que nos enfrentáramos, encapsulando cada concepto en lo que se dicen llamar objetos y haciéndoles interactuar entre ellos, habiéndoles dotado de una serie de propiedades. Surgieron así numerosas metodologías para ayudar en tal proceso de descomposición y aparecieron herramientas que incluso automatizaban parte del proceso. Sin embargo, frecuentemente la relación entre la complejidad de la solución y el problema resuelto hace pensar en la necesidad de un nuevo cambio (Keuler, 2011). Así pues, nos encontramos con muchos problemas donde la POO no es suficiente para capturar de una manera clara todas las propiedades y comportamientos de lo que queremos dotar nuestra aplicación.

El desarrollo de software en todos sus niveles continuamente enfrenta el incremento de la complejidad y tamaño de los sistemas que se desarrollan, consecuentemente los métodos que se aplican en el diseño del software deben ser escalables en tamaño y complejidad con el fin de entregar productos de calidad en el tiempo requerido. Es decir, los métodos de programación que se aplican deberán proporcionar apoyo práctico de manera eficiente y eficaz para sus requerimientos y componentes (Keuler, 2011).

El diseño del software cumple una función elemental en el desarrollo del software ya que es un punto de quiebre en donde se pone en peso a la calidad (Sommerville, 2008). El diseño además de tener la labor de implementar el análisis en relación con la especificación de requerimientos sienta las bases para la implementación. Entonces en este punto es donde la implementación hereda la estructura y el modo en que ha de implementarse cada componente de software. Sin embargo, una buena especificación de requerimientos y un buen análisis sin una buena separación de competencias al nivel de requerimientos, hacen el efecto que, en el diseño del software exista una pobre identificación o una sobrecarga en la tarea de identificar atributos del sistema y su división en subsistemas por ende la arquitectura a nivel del software se verá afectada y posteriormente en la implementación se dará el efecto de atributos del sistema implementados en otros componentes y a su vez diseminados por el resto del sistema, a este efecto se le denomina la tiranía de la descomposición dominante (Sampieri y otros, 2006).

Como consecuencia se recae en una poca mantenibilidad, y la complejidad de sistema aumenta, y a su vez existiendo una diferencia entre diseño e implementación, lo cual en la práctica no es deseable como se indica en (Bosch, 2000), un indicador de un buen sistema es aquel que guarda relación entre las etapas de su desarrollo.

Por otro lado, según como menciona en (Sommerville, 2008) al aumentar el tamaño del sistema las relaciones entre requerimientos y componentes son complejas, un requerimiento puede implementarse en un componente, y cada uno incluir partes, elementos de otros requerimientos. Es decir, de una u otra manera si se extiende un cambio de requerimiento o lógica se afectará a varios componentes, lo que compromete seriamente al diseño del software y la mantenibilidad del sistema.

Los conceptos y tecnologías reunidos bajo el nombre "Programación Orientada a Aspectos" (AOP, por las siglas de aspect oriented programming; o AOSD, por aspect oriented software development) buscan resolver un problema identificado hace tiempo en el desarrollo de software. Se trata del problema de la separación de incumbencias (separation of concerns). AOP no es el único intento por solucionar este problema, del que se habla a continuación: hay varias propuestas, muchas de las cuales se agrupan (junto con AOP) en el campo de estudio denominado ASoC (advanced separation of concerns). (Kendall, 2005).

La programación orientada a aspectos (POA) es una nueva metodología de programación que aspira a soportar la separación de competencias para los aspectos antes mencionados. Es decir, que intenta separar los componentes y los aspectos unos de otros, proporcionando mecanismos que hagan posible abstraerlos y componerlos para formar todo el sistema. En definitiva, lo que se persigue es implementar una aplicación de forma eficiente y fácil de entender. (Acosta, 2012).

La programación orientada a aspectos (POA), es un desarrollo que sigue al paradigma de la orientación a objetos, y como tal, soporta la descomposición orientada a objetos, además de la procedimental y la descomposición funcional. Pero, a pesar de esto, POA no se puede considerar como una extensión de la programación orientada a objetos (POO), ya que puede utilizarse con los diferentes estilos de programación ya mencionados (Bahit, 2011).

Haciendo mención de los **antecedentes nacionales** se realizaron dos grupos de estudio uno orientado a aspectos y otro grupo orientado a objetos, en los que se desarrollaron actividades de especificación de requerimientos, análisis para de esta manera llegar al diseño y posteriormente pasar a la codificación, y en última instancia comparar ambos como productos finales. Los resultados muestran que el uso de la programación orientada a aspectos mejora el diseño de un software en comparación con la orientación a objetos. En su realización se incluyeron las diferentes etapas de desarrollo del software para así de esta forma determinar el efecto de cada tipo de programación en el software (Gonzales, 2014).

1.1.2 Problemática de la investigación

En el desarrollo de software en todos sus niveles continuamente enfrenta el incremento de la complejidad y tamaño de los sistemas que se desarrollan, consecuentemente los métodos que se aplican en el diseño del software deben ser escalables en tamaño y complejidad con el fin de entregar productos de buena calidad en el tiempo requerido. Es decir, los métodos de programación que se aplican deberán proporcionar apoyo práctico de manera eficiente y eficaz para sus requerimientos y componentes (Keuler, 2011).

El diseño del software cumple una función elemental en el desarrollo del software ya que es un punto de quiebre en donde se pone en peso a la calidad (Sommerville, 2011). El diseño además de tener la labor de implementar el análisis en relación a la especificación de requerimientos sienta las bases para la implementación. Entonces en este punto es donde la implementación hereda la estructura y el modo en que ha de implementarse cada componente de software.

Sin embargo, una buena especificación de requerimientos y un buen análisis sin una buena separación de competencias al nivel de requerimientos, hacen el efecto que, en el diseño del software exista una pobre identificación o una sobrecarga en la tarea de identificar atributos del sistema y su división en subsistemas por ende la arquitectura a nivel del software se verá afectada y posteriormente en la implementación se dará el efecto de atributos del sistema implementados en otros componentes y a su vez diseminados por el resto del

sistema, a este efecto se le denomina la tiranía de la descomposición dominante (Sampieri y otros, 2006).

Como consecuencia se recae en una poca mantenibilidad, y la complejidad de sistema aumenta, y a su vez existiendo una diferencia entre diseño e implementación, lo cual en la práctica no es deseable como se indica en (Bosch, 2000), un indicador de un buen sistema es aquel que guarda relación entre las etapas de su desarrollo.

La programación orientada a aspectos ofrece soporte a este problema, sin embargo, en muchos casos, su aplicación se da en la etapa de implementación vale decir la codificación, esto quiere decir que en la práctica, en la etapa de diseño se hace mención a atributos del sistema pero no muestra la manera de cómo interactuarán con los requerimientos funcionales, o bien se realizan tablas de correspondencia entre atributos del sistema y requerimientos funcionales, con esto se piensa que el efecto en el diseño del software aplicando aspectos se hace de manera tardía provocando una no correspondencia entre fases de desarrollo, por lo tanto se debería aplicar desde los inicios del proceso de desarrollo del software.

Pero conlleva a retos como su identificación y estructuración a través de los requerimientos, siguiendo el análisis hasta el diseño y llegando a la implementación, una vez que se realice se podrá comprender un efecto más cercano en el diseño del software.

Finalmente, ante esta situación se evidencia que diseño de software forma una de las principales partes del desarrollo de software. Así mismo, se ve evidente comparar el efecto en el diseño de un software de estos enfoques en todo el proceso de desarrollo, por ello es que se realiza el presente trabajo de investigación.

1.2 Formulación del problema

1.2.1 Problema general

¿Mediante un análisis comparativo se puede determinar la eficiencia de la programación orientada a aspectos y la programación orientada a objetos en la construcción de un módulo de autenticación?

1.2.2 Problemas específicos

- a) ¿Se podrá establecer y cuantificar la cantidad de líneas de código, complejidad ciclomática y uso de memoria como características que definen la eficiencia en el desarrollo con programación orientada a aspectos del módulo de autenticación del software de la biblioteca de la liga departamental de futbol de Tacna?
- b) ¿Se podrá establecer y cuantificar la cantidad de líneas de código, complejidad ciclomática y uso de memoria como características que definen la eficiencia en el desarrollo con programación orientada a objetos del módulo de autenticación del software de la biblioteca de la liga departamental de futbol de Tacna?

- c) ¿Se podrá realizar un análisis comparativo veraz de los resultados obtenidos de cuantificar nuestros índices de eficiencia?

1.3 Justificación

La necesidad de mejorar la tarea del diseño de un software, elevar la abstracción, eficiencia y desarrollo, ha llevado al surgimiento de nuevos paradigmas de programación. Y de la manera de cómo abordamos el diseño de un software desde sus inicios hasta la entrega del producto final.

Esta investigación generará reflexión y discusión tanto sobre el conocimiento existente en la programación orientada a aspectos, como dentro de la programación orientada a objetos, ya que, de alguna manera u otra, se confrontarán teorías sobre el efecto en nuestro caso sobre la construcción de un módulo de autenticación, lo cual necesariamente conlleva hacer epistemología del conocimiento existente.

En este estudio, por otra parte, se comparará el efecto de los tipos de programación aplicados en el diseño del módulo de autenticación. Por tal razón investigaciones futuras encontrarán en este trabajo la profundización del conocimiento acerca de los enfoques de programación presentados cuya relevancia en la construcción del software, en particular, y de los programadores, analistas, en general, es indiscutible.

1.4 Alcances y limitaciones

El alcance de nuestro estudio abarca a la programación orientada a aspectos (POA) y La programación orientada a objetos (POO) únicamente en el software de la biblioteca de la liga departamental de futbol de Tacna, siendo este software distribuido a nivel nacional, se tomó la muestra de nuestro departamento.

La primera limitación de este estudio es la falta de un servidor propio en donde se ejecutarán los sistemas, se menciona porque a través del estudio y en la implementación existen frameworks, librerías que, para el uso de funciones especiales, requieren la instalación de dependencias en el lado del servidor, lo cual limita un poco la implementación. Y la segunda limitante es que el proceso de desarrollo del software en algunos casos una actividad que incluye varias disciplinas a nivel de etapas, para el presente trabajo no se realizará con múltiples equipos de desarrollo sino se realizará de manera individual.

1.5 Objetivos

1.5.1 Objetivo general

Comparar la programación orientada a objetos y la programación orientada a aspectos teniendo como base la eficiencia en el desarrollo del módulo de autenticación del software de la biblioteca de la liga departamental de futbol de Tacna.

1.5.2 Objetivos específicos

- a) Cuantificar y evaluar la cantidad de líneas de código, complejidad ciclomática y uso de memoria con programación orientada a aspectos en el módulo de autenticación del software de la biblioteca de la liga departamental de futbol de Tacna.
- b) Cuantificar y evaluar la cantidad de líneas de código, complejidad ciclomática y uso de memoria con programación orientada a objetos en el módulo de autenticación del software de la biblioteca de la liga departamental de futbol de Tacna.
- c) Hacer un análisis comparativo de los resultados de las evaluaciones de las características.

1.6 Hipótesis

1.6.1 Hipótesis global

En la construcción de un módulo de autenticación el diseño e implementación con la programación orientada a aspectos, la eficiencia es favorable comparado con la programación orientada a objetos en el software de la biblioteca de la liga departamental de futbol de Tacna.

1.6.2 Sub hipótesis

- a) En la construcción de un módulo de autenticación con la programación orientada a aspectos la eficiencia es favorable.

- b) En la construcción de un módulo de autenticación con la programación orientada a objetos la eficiencia es desfavorable.
- c) En la comparación de módulos la diferencia de rendimientos es significativa.

1.7 Variables

1.7.1 Identificación de variables

1.7.1.1 Variables:

V1: Módulo de autenticación con programación orientada a aspectos.

V2: Módulo de autenticación con programación orientada a objetos.

1.7.2 Definición de variables

a) Módulo de autenticación con programación orientada a aspectos

Definición: La Programación orientada a aspectos es un paradigma de programación cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de incumbencias. Para crear un módulo de autenticación se basará en que se pueden encapsular los diferentes conceptos que componen esta aplicación en entidades bien definidas, asimismo se tendrá que separar las incumbencias, eliminando las dependencias entre cada uno de los módulos. De esta forma se consigue razonar mejor sobre los conceptos, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables.

b) Módulo de autenticación con programación orientada a objetos

Definición: La programación orientada a objetos es un paradigma de programación que usa los objetos en sus interacciones, para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento (David Watt, 2010). Se creará un módulo de autenticación basado en el más común de los análisis para la implementación de un login.

1.7.3 Operacionalización de variables

Tabla 1:

Identificación de variables e indicadores

Variables	Indicador
V1: Módulo de autenticación con programación orientada a objetos	• Cantidad de líneas de código con POO • Coeficiente de complejidad ciclomática con POO • Promedios de tiempo de ejecución y el consumo de memoria con POO
V2: Módulo de autenticación con programación orientada a aspectos	• Cantidad de líneas de código con POA • Coeficiente de complejidad ciclomática con POA • Promedios de tiempo de ejecución y el consumo de memoria con POA

Fuente: *Elaboración propia*

1.8 Diseño de la investigación

1.8.1 Diseño experimental o no experimental

La investigación no experimental es también conocida como investigación Ex Post Facto, término que proviene del latín y significa después de ocurridos los hechos. Según Kerlinger la investigación ex post facto es un tipo de “investigación sistemática en la que el investigador no tiene control sobre las variables independientes porque ya ocurrieron los hechos o porque son intrínsecamente manipulables” (Kerlinger, 1983 p.269). En la investigación ex post facto los cambios en la variable independiente ya ocurrieron y el investigador tiene que limitarse a la observación de situaciones ya existentes dada la incapacidad de influir sobre las variables y sus efectos (Sampieri Hernández, Fernández y Baptista, 2010).

Si consideran que la variación de las variables se logra no por manipulación directa sino por medio de la selección de las unidades de análisis en las que la variable estudiada tiene presencia (Sampieri Hernández, Fernández y Baptista, 2010).

Es muy importante destacar que en una investigación experimental la variable independiente se manipula y por eso se le llama variable activa mientras que en la investigación ex post facto la variable independiente no es susceptible de manipulación y por eso se le llama variable atributiva. Existen al menos tres aspectos en los que la investigación experimental es semejante a la investigación ex post facto:

- Por medio de estos tipos de investigación se pueden comprobar hipótesis.
- Se utilizan grupos semejantes excepto en algún aspecto o característica específica.
- Se utilizan métodos estadísticos para el tratamiento y análisis de datos.

Las diferencias principales entre ambos tipos de investigación radican en los siguientes aspectos:

- La investigación experimental tiene un control estricto de las variables extrañas, no así en la investigación ex post facto.
- La investigación experimental parte de grupos similares para encontrar una diferencia y establecer la relación causa-efecto. La investigación ex post facto estudia dos grupos diferentes y busca qué es lo que hace la diferencia para establecer la relación causa-efecto.

Con los resultados que arroja una investigación ex post facto no es posible afirmar con seguridad una relación causal entre dos o más variables, como ocurre en la investigación experimental. Lo anterior debido a la posibilidad de que no se hayan encontrado otros factores que sí están afectando la variable dependiente. Si esto ocurre entonces se tienen datos espurios o falsos, es decir, existen serias dudas acerca de su origen.

La investigación experimental implica establecer mecanismos de control como condición del método experimental. D'Ary, Jacobs y Razavieh, consideran que la variación de las variables se logra no por manipulación directa sino por medio de la selección de las unidades de análisis en las que la variable estudiada tiene presencia, por ejemplo, se puede analizar cómo influyó el movimiento del primero de enero de 1994 en Chiapas sobre la economía nacional, también se puede analizar la percepción de personas con síndrome de down y personas que no lo tienen. Si los cambios en la variable independiente ya ocurrieron y están fuera de la capacidad de manipulación y control del investigador, por esta razón en la investigación ex post facto se estudia de manera retrospectiva el fenómeno en cuestión. Ambas variables están fuera del control del investigador. El investigador empieza con la observación de hechos que ya se han presentado y que se han manifestado en una serie de eventos. A partir de las observaciones se procede a diseñar tanto los objetivos como las hipótesis dando inicio a la investigación en sentido opuesto a una investigación experimental.

1.8.2 Población y muestra

- **Población:** El software de la biblioteca de la liga departamental de futbol de Tacna tiene veinticinco (25) módulos, de los cuales 07 de dichos módulos tienen más de 500 LCD en su implementación con programación orientada a objetos (POO), para un mejor análisis del presente estudio se escogió uno de ellos.

Tabla 2:*Población de módulos del software.*

SOFTWARE DE LA BIBLIOTECA DE LA LIGA DEPARTAMENTAL DE FUTBOL DE TACNA	
TIPO DE PROGRAMACIÓN	POO
MÓDULO	LCD
Autenticación	503
administrador	998
autor	355
categoría	362
ciclo	375
configuración	901
cuenta	532
editorial	393
ejemplar	946
ejemplar_estado	242
ejemplar_ubicacion	261
escuela	384
escuela_usuario	203
estadoejemplar	357
feriado	339
país	345
perfildeusuario	411
perfildeusuario_has_usuario	224
préstamo	751
publicación	10
reserva	201
sanción	179
tipopublicacion	379
ubicación	346
usuario	906

***Fuente:** Elaboración propia*

- **Muestra:** El módulo de autenticación del software de la biblioteca de la liga departamental de futbol de Tacna, cuenta con 503 LCD, para el presente caso de estudio es el más adecuado para su análisis y posterior comparación con programación orientada a aspectos (POA), ya que se puede notar bien la abstracción en dicho módulo.

Tabla 3:

Muestra de módulos del software.

SOFTWARE DE LA BIBLIOTECA DE LA LIGA DEPARTAMENTAL DE FUTBOL DE TACNA	
TIPO DE PROGRAMACIÓN	POO
MÓDULO	LCD
Autenticación	503

Fuente: *Elaboración propia*

1.8.3 Técnicas e instrumentos para la recolección de datos

Son procedimientos utilizados por el investigador para recabar información (Pelekais, Finol, Neuman y Parada, 2005).

De igual forma, Balestrini señala que “estas técnicas son diversas según el objeto que se aplique y no se excluyan entre sí. Todavía es preciso, por una parte, saber elegir la más adecuada y, por la otra, utilizarla convenientemente” (Grawitz, 2006, pág. 145).

Se utilizó como instrumento las tablas comparativas de los resultados obtenidos en el desarrollo.

1.8.3.1 Técnicas

Dada la naturaleza del estudio se requirió obtener datos reales y actuales a partir de la ejecución del software viendo rendimiento computacional y sus cambios a partir de la ejecución del mismo en ambos casos de estudio; para lo cual se ha establecido como técnicas de investigación a la “**observación científica**”.

La observación, consiste en saber seleccionar aquello que se quiere analizar. Se suele decir que "Saber observar es saber seleccionar". Para la observación lo primero es plantear previamente qué es lo que interesa observar.

La observación científica "tiene la capacidad de describir y explicar el comportamiento, al haber obtenido datos adecuados y fiables correspondientes a conductas, eventos y /o situaciones perfectamente identificadas e insertas en un contexto teórico.

En la observación se puede considerar tres niveles sistematización:

- **La observación asistemática:** ha sido la que tradicionalmente sea realizado en el deporte. El conocido "ojímetro" para dar información sobre una actuación determinada o las observaciones realizadas sin ninguna planificación previa. Esta observación consiste en una constatación directa, sin ningún objetivo explicitado y sin estar perfectamente definido, sin un orden o planificación determinado. Son narraciones escritas en forma directa o datos recogidos de una manera azarosa.

- **La observación semisistemática:** en ésta los objetivos del registro se fijan a priori; precisan planificar el orden y el tiempo, se necesita especificar la conducta o evento antecedente, el núcleo observado y la conducta o evento posterior, pero no están categorizadas las conductas a observar.
- **La observación sistemática:** intentan agrupar la información a partir de ciertos criterios fijados previamente o partiendo de estos registros. Requiere categorizar hechos, conductas y / o eventos que se han de observar.

Estos tres niveles conducen a dos tipos claros de observación: la observación asistemática y la observación científica.

"El estudio del acto de enseñanza utiliza como principal instrumento de investigación, sistemas o planos de observación de acontecimientos o comportamientos". Como toda ciencia la enseñanza y el entrenamiento necesitan de grabaciones objetivas. La observación constituye un método de toma de datos destinados a representar lo más fielmente posible lo que ocurre, la realidad. Piéron (1986).

1.8.3.2 Instrumentos

Los instrumentos de medición son los recursos que utiliza el investigador para registrar la información o datos sobre las variables que tiene en mente (Sampieri Hernández, 2010, pág. 276).

Es de particular importancia otorgar y no olvidar el valor que tienen los instrumentos que se emplearán en una investigación. Muchas veces se inicia un trabajo sin identificar qué tipo de información se necesita o las fuentes en las cuales puede obtenerse; esto ocasiona pérdidas de tiempo, e incluso, a veces, el inicio de una nueva investigación. Por tal razón, se considera esencial definir las técnicas a emplearse en la recolección de la información, al igual que las fuentes en las que puede adquirir tal información.

Las técnicas son de hecho, recursos o procedimientos de los que se vale el investigador para acercarse a los hechos y acceder a su conocimiento y se apoyan en instrumentos para guardar la información tales como: el cuaderno de notas para el registro de observación y hechos, el diario de campo, los mapas, la cámara fotográfica, la grabadora, la filmadora, el software de apoyo; elementos estrictamente indispensables para registrar lo observado durante el proceso de investigación.

Instrumentos que pautan la observación:

- Cuestionario de observación
- Guion de observación
- Lista de preguntas
- Lista de control
- Escala de estimación

- Escala de calificación
- Lista de cotejo

Instrumentos que organizan y sistematizan la información:

- Registro anecdótico
- Cuaderno de observación /diario
- Ficha de seguimiento individual
- Informe de observación
- Ficha de registro de observación

El instrumento utilizado para organizar y sistematizar la información es la “*ficha de registro*”, donde se almacenaron los datos obtenidos de nuestras 02 variables, por cada compilación del módulo en observación se registraron datos de los 03 indicadores (LCD, complejidad ciclomática y uso de memoria) de cada una de las variables (**Anexo 03**).

1.8.4 Análisis de datos

El procesamiento de datos, cualquiera que se la técnica empleada para ello no es otra cosa que el registro de los datos obtenidos por los instrumentos empleados, mediante una técnica analítica en la cual se comprueba la hipótesis y se obtienen las conclusiones (Tamayo, 2004, pág. 187).

Según Arias, no basta con recolectar los datos ni cuantificarlos adecuadamente. Es necesario analizarlos, compararlos y presentarlos de manera que realmente lleven a la confirmación o al rechazo de la hipótesis (Tamayo, 2004, p. 187).

Los datos recopilados fueron procesados y analizados con el paquete estadístico SPSS (Statistical Package for Social Sciences) en su versión 20 con el apoyo de MS Excel versión 2013; presentando los resultados en tablas y/o cuadros estadísticos a fin de realizar el análisis e interpretación.

1.8.5 Selección de pruebas estadísticas

La prueba “t” es una prueba estadística para evaluar si dos grupos difieren entre sí de manera significativa respecto a sus medias (Sampieri Hernández, 2010, pág. 460).

Para comprobar las hipótesis se aplicó la prueba estadística de T de Student, a fin de determinar si hay diferencia significativa entre las medias de los resultados de uso de memoria.

CAPÍTULO II

MARCO TEÓRICO

2.1 Marco referencial

En el año de 2012, Lester Rodríguez, en su libro “Caracterización de la Programación Orientada a Aspectos (POA)”, implementó una aplicación utilizando el paradigma orientado a objetos y orientado a aspectos con el objetivo de establecer una comparación entre ellos identificando las ventajas y desventajas de este último (Acosta, 2012).

En el año 2013, Boticki Boticki I. Katic, M. y Martin S., en su artículo “Exploring the educational Benefits of Introducing Aspect - Oriented Programming into a Programming Course”, realizó un experimento enseñando en forma separada a dos grupos de estudiantes, uno con la programación orientada a aspectos (POA) y programación orientada a objetos (POO) con el objeto de evaluar el efecto educacional de cada tipo de programación. Llegando a la conclusión que la programación orientada a aspecto logra mejores resultados académicos, Los estudiantes que utilizan POA escribieron código más cortos y concisos, y sus programas se caracterizaron por una mejor modularidad (Boticki y otros, 2013).

En el año de 2008, Tabares Marta Silvia, Alférez Salinas Germán y Alferez Salinas Edward en su artículo denominado “El desarrollo de software orientado a aspectos: un caso práctico para un sistema de ayuda en línea”, explica la teoría de la POA, además de ello desarrolla un caso práctico en la cual se enfrenta a un sistema desarrollado orientado a objetos, re-diseñando la lógica del sistema comparando y obteniendo características positivas en cuanto a un código modularidad y acoplamiento (Tabares y otros, 2008).

En el año de 1998, William Harrison y Harold Ossher en su artículo titulado “Subject - Oriented Programming (A Critique of Pure Objects)”, explica en forma teórica que los objetos están ligados con el concepto de identidad (separación de estado y comportamiento), así mismo, muchos puntos de vista con abstracciones subjetivas del objeto distorsionan el concepto de programación orientada a objetos, por lo tanto se debe poner más atención en la identidad para unirlos.

2.2 Bases teóricas respecto al problema

2.2.1 Programación orientada a objetos

2.2.1.1 Definición

Los programas que permiten el acceso del usuario a un sistema utilizan una autenticación para verificar la identidad de cada uno (es decir, para establecer que el usuario es quien dice ser) y la programación orientada a objetos es un paradigma de programación que usa los objetos en sus interacciones, para diseñar aplicaciones

y programas informáticos. Está basado en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento (David Watt, 2010).

Es un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas miembros de una jerarquía de clases unidas mediante relaciones de herencia. (Booch, 2009).

La programación orientada a objetos se puede definir como una técnica o estilo de programación que utiliza objetos como bloques esenciales de construcción. Los elementos básicos de la programación orientada a objetos (POO) son: objetos, mensajes, métodos y clases (Barraza y otros, 2006).

2.2.1.2 Características de la programación orientada a objetos

La potencia real de los objetos reside en sus propiedades: abstracción, herencia y polimorfismo (Barraza y otros, 2006).

a) Abstracción

Una abstracción denota las características esenciales de un objeto que lo distinguen de todos los demás tipos de objeto y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador. (Booch, 2009).

Es la capacidad de crear tipos de datos definidos por el usuario. Es una definición de tipo que incluye datos y funciones (métodos) en una sola estructura definida llamada clase. La abstracción de datos es un concepto fundamental en el diseño de un programa, pues permite definir el dominio y la estructura de los datos (atributos), junto con una colección o conjunto de operaciones que tienen acceso a los mismos (métodos). La abstracción permite no preocuparse de los detalles accesorios, y generalizar y centrarse en los aspectos que permiten tener una visión global del problema (Barraza y otros, 2006).

La abstracción es un nuevo tipo de dato definido por el programador; está compuesto por atributos (identificadores o datos) y métodos o acciones que se realizan sobre los atributos, definidos en una sola estructura llamada clase.

CLASE = atributos + métodos

Definir una abstracción significa describir una entidad del mundo real, con características, propiedades y métodos seleccionados con la posibilidad de emular una interacción entre ellas para lograr un fin.

b) Herencia

La herencia es la jerarquía de clases más importante (elemento esencial en los sistemas orientados a objetos). Básicamente la herencia define una relación entre clases, en la que una clase comparte la estructura de comportamiento definida

en una o más clases (lo denominado herencia simple o múltiple). **“La herencia es la capacidad para crear nuevas clases de objetos que se construyen basados en clases existentes”** (Booch, 2009). Además, a estos nuevos objetos creados es posible asignarles nuevos atributos y métodos. (Aguilar, 1996).

Es la relación existente entre dos o más clases, donde una es la principal (madre) y otras son secundarias y dependen (heredan) de ellas (clases hijas), donde a la vez los objetos heredan las características de los objetos de los cuales heredan (Bahit, 2011).

La herencia permite definir una nueva clase extendiendo una clase ya existente. Esta nueva clase, clase derivada, posee todas las características de la clase más general y, además, la extiende en algún sentido (Echevarría y otros, 2004).

La herencia en términos más simples es una propiedad que permite obtener características adicionales de una clase “base” de las que ya tienen.

c) Polimorfismo

La palabra polimorfismo proviene de dos raíces griegas: “poli”, múltiples y “morfismo”, formas. Esta característica se describe como la capacidad que objetos similares tienen para responder de diferentes formas al mismo mensaje, y permite al programador implementar múltiples formas de un mismo método, dependiendo cada una de ellas de la clase sobre la que se realice la implementación (Barraza y

otros, 2006, pág. 12). Esto permite acceder a varios métodos distintos utilizando el mismo medio de acceso (el mismo nombre). El polimorfismo está muy relacionado con la herencia.

Con respecto al polimorfismo nos dice que “es la propiedad que indica, literalmente, la posibilidad de que una entidad tome muchas formas. En términos prácticos, el polimorfismo permite referirse a objetos de clases diferentes mediante el mismo elemento de programa y realizar la misma operación de diferentes formas, según sea el objeto que se referencia en ese momento” (Aguilar, 1996, pág. 19).

Se puede concluir diciendo que es la capacidad de distintos objetos contar con métodos, propiedades y atributos del mismo nombre sin que se interfieran. Adicionalmente se tiene:

d) Encapsulamiento

La encapsulación o encapsulamiento es la propiedad que permite asegurar que el contenido de la información de un objeto está oculta al mundo exterior: el objeto A no conoce lo que hace el objeto B, y viceversa. La encapsulación (también se conoce como la ocultación de la información), en esencia, es el proceso de ocultar todos los secretos de un objeto que no contribuyen a sus características esenciales (Aguilar, 1996).

El acceso a los datos de una clase debe realizarse de forma controlada, protegiéndolos de accesos no deseados. Cuando se desarrolla una aplicación, a veces es necesario ocultar los tipos de datos usados para que el usuario permanezca independiente de los detalles de los mismos. De esta manera, el usuario no es sensible a los cambios que se puedan producir en los tipos de datos elegidos dentro de una clase (Echevarría y otros, 2004).

e) Jerarquía

La jerarquía es una propiedad que permite la ordenación de las abstracciones. Las dos jerarquías más importantes de un sistema complejo son: estructura de clases (jerarquía «es-un» (is-a) generalización/especialización) y estructura de objetos (jerarquía «parte-de» (part-of): agregación) (Aguilar, 1996).

f) Modularidad

La modularidad es la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes (Aguilar, 1996).

La modularidad, permite poder modificar las características de la clase que definen a un objeto, de forma independiente de las demás clases en la aplicación. En otras palabras, si nuestra aplicación puede dividirse en módulos separados,

normalmente clases, y a su vez estas en sub clases, y estos módulos pueden compilarse y modificarse sin afectar a los demás de su entorno de trabajo o módulos contiguos, entonces dicha aplicación ha sido implementada en un lenguaje de programación que soporta la modularidad.

g) Tipos

Un tipo es una caracterización precisa de las propiedades estructurales o de comportamiento que comparten una serie de entidades (Las clases implementan a los tipos) no son exactamente lo mismo. Los tipos son la puesta en vigor de la clase de los objetos, de modo que los objetos de tipos distintos no pueden intercambiarse o, como mucho, pueden intercambiarse sólo de formas muy restringidas (Booch, 2009).

h) Concurrencia

La concurrencia es la propiedad que distingue un objeto activo de uno que no está activo.

Una vez que se introduce la concurrencia a un sistema, hay que tener en cuenta como los objetos activos sincronizan sus actividades con otros, así como con objetos puramente secuenciales.

Mientras que la programación orientada a objetos se centra en la abstracción de datos, encapsulamiento y herencia, la concurrencia se centra en la abstracción de

procesos y la sincronización. El objeto es un concepto que unifica estos dos puntos de vista distintos: cada objeto (extraído de una abstracción del mundo real) puede representar un hilo separado de control (una abstracción de un proceso). Tales objetos se llaman activos.

En un sistema orientado a objetos se puede conceptualizar el mundo como un conjunto de objetos cooperativos, algunos de los cuales son activos y sirven así como centros de la actividad independiente (Booch, 2009).

i) Persistencia

Un objeto de software ocupa una cierta cantidad de espacio, y existe durante una cierta cantidad de tiempo.

Hay un espacio de existencia que va desde los objetos transitorios que surgen en la evaluación de una expresión hasta los objetos de una base de datos. Este espectro de persistencia abarca lo siguiente:

- Resultados transitorios en la evaluación de expresiones.
- Variables locales en la activación de procedimientos.
- Variables propias, globales y elementos del montículo cuya duración difiere de su ámbito.
- Datos que existen entre ejecuciones de un programa.
- Datos que existen entre varias versiones de un programa.

- Datos que sobreviven al programa.

La persistencia abarca algo más que la mera duración de los datos. En las bases de datos orientadas a objetos, no sólo persiste el estado del objeto, sino que su clase debe trascender también a cualquier programa individual, de forma que todos los programas interpreten de la misma manera el estado.

La persistencia es la propiedad de un objeto por la que su existencia trasciende el tiempo, es decir, el objeto continúa existiendo después de que su creador deja de existir y/o el espacio (Booch, 2009). Es decir, la posición del objeto varía con respecto al espacio de direcciones en el que fue creado.

2.2.1.3 Ventajas de la programación orientada a objetos

Las principales ventajas de la programación orientada a objetos son (Barraza y otros, 2006):

- Las herramientas POO ayudan a mejorar la complejidad.
- Mejora la productividad debido a la reutilización de código (herencia).
- Mejora la seguridad y calidad.
- Permite generar programas modulares mejor estructurados.

2.2.1.4 Inconvenientes de la programación orientada a objetos

Se cita algunos inconvenientes respecto a la programación orientada a objetos (Martinez, 2014):

- Son más grandes que otros programas, consume más memoria y espacio en disco.
- Requiere mucho trabajo para crear, incluyendo la creación de diagramas en la fase de planificación y la codificación en la fase de ejecución.

Finalmente se puede concluir:

- La programación orientada a objetos no es la solución definitiva, pero permite construir programas complejos a partir de entidades de software más simples llamadas objetos, que son instancias reales o muestras de clases, lo cual permite heredar datos y códigos de aplicaciones existentes. Esto mejora la productividad del programador y facilita la extensión y la reutilización de clases en otras aplicaciones con mínima modificación al código original (Barraza y otros, 2006).
- Uno de los principales problemas de la programación orientada a objetos, es la incapacidad de abstraer eficientemente la funcionalidad que se repite en distintos módulos que no estén relacionados en la jerarquía de clases. Uno de los ejemplos más claros de este inconveniente son las instrucciones de bitácora y de traza de ejecución. Si en un sistema se quisiera mostrar un mensaje cada vez que comienza la ejecución de un método, se tendría que agregar dicha instrucción en todos los métodos del sistema. Supóngase, que se incorpora una

instrucción en cada uno de los métodos del sistema que muestre el nombre del método al cual se está invocando. Si en el futuro, se decide que es necesario ver además, los valores de los argumentos con los cuales fue llamado cada método, se necesitaría hacer una modificación en cada uno de los lugares donde fue aplicada la instrucción originalmente. Dicha modificación implica un esfuerzo lineal en comparación con el tamaño del sistema. Todos estos problemas surgen de la falta de mecanismos del paradigma para la abstracción de comportamiento transversal (Erdody, 2006).

2.2.2 Programación orientada a aspectos

2.2.2.1 Definición

La programación orientada a aspectos es un estilo de programación cuyo principal objetivo es lograr una adecuada modularización de los conceptos involucrados en una aplicación, esto se traduce en lograr la separación entre los requerimientos funcionales de los no funcionales para obtener un mejor entendimiento de los conceptos, eliminando la dispersión del código y haciendo que las implementaciones resulten más comprensibles, adaptables y reutilizables (Arismendi y otros, 2008).

La programación orientada a aspectos (POA) no es un estilo de programación distinto a las técnicas de programación anteriores, sino que es un

estilo de programación que agrega nuevos elementos para permitir la modularización de los asuntos transversales. Desde la perspectiva de la POA una aplicación consiste en un conjunto de requerimientos funcionales modularizables (componentes base o funcionalidad básica), más un conjunto de asuntos transversales de difícil modularización (aspectos).

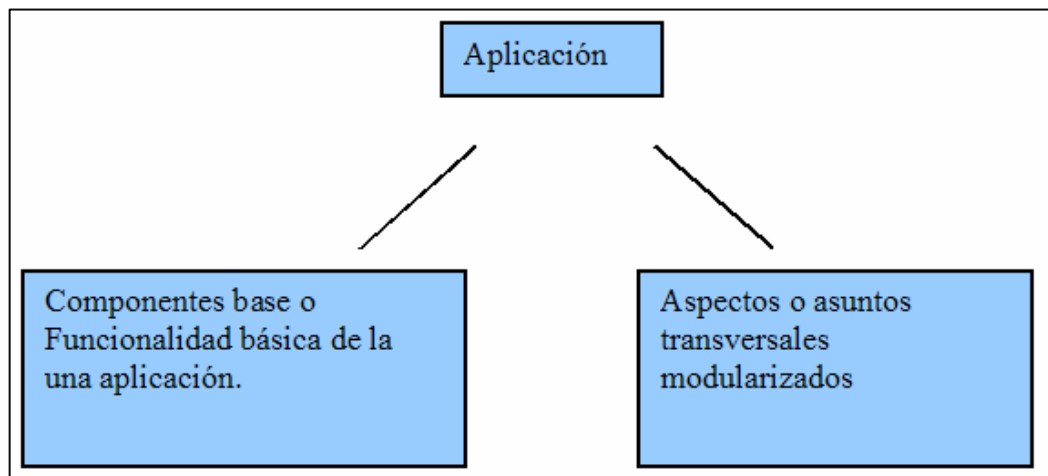


Figura 01: Conceptos que involucra una aplicación orientada a aspectos.

Fuente: Arismendi y otros, 2008

La POA es una tecnología que se basa principalmente en la programación orientada a objetos (POO) para la implementación de los componentes base, esto no quiere decir bajo ningún caso que no pueda utilizarse con otros estilos de programación, dado que la POA también soporta programación estructurada o funcional. Es por esta razón que la programación orientada a aspectos no puede ser vista como una extensión de la POO.

2.2.2.2 Comprendiendo la separación de intereses

En la Ingeniería del Software se ha utilizado durante años el principio de la separación de incumbencias (Separation of Concerns) para gestionar la complejidad del desarrollo de software mediante la separación de las funcionalidades principales de la aplicación de otras partes con un propósito específico, ortogonales a la funcionalidad principal (autenticación, administración, rendimiento, login, etc.).

Es posible ver un sistema como la combinación de múltiples intereses. Se define interés como cualquier aspecto que le concierne a un programa, ya sea relacionado a la infraestructura, al negocio, a los requerimientos o a las estructuras de diseño.

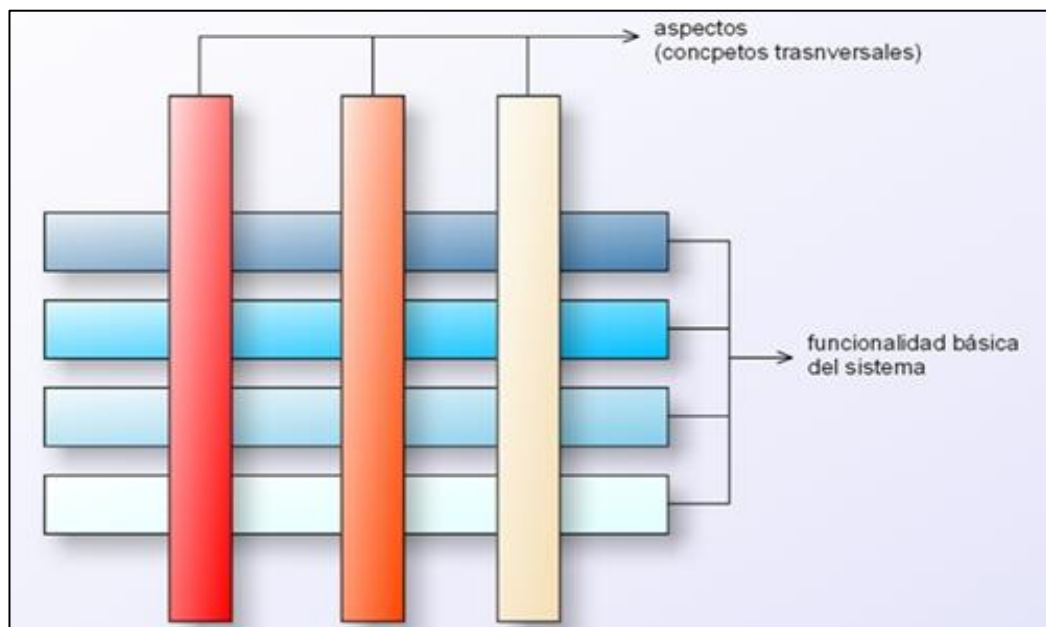


Figura 02: Incumbencias o intereses en un sistema

Fuente: Brichau Johan y D'Hondt Theo Aspect - Oriented Software Development – 2005, Pag. 128

El principio de separación de las competencias o intereses promueve el buen diseño e implementaciones reutilizables. Es un factor determinante en la evolución continua de los paradigmas de desarrollo porque los ingenieros de software necesitan mecanismos de desarrollo adecuados para lograrlo. Aunque el paradigma orientado a objetos ya apoya este principio bastante bien, “aspecto” implica ofrecerles enfoques y técnicas mejoradas que permiten una mejor separación de las competencias en todo el ciclo de vida de desarrollo (Brichau y otros, 2005).

2.2.2.3 Aspecto

Un aspecto es una unidad modular que se disemina por la estructura de otras unidades funcionales. Los aspectos existen tanto en la etapa de diseño como en la etapa de implementación. Un aspecto de diseño es una unidad modular que se entremezcla en la estructura de otras partes del diseño. Un aspecto de programa o de código es una unidad modular del programa que aparece en otras unidades del programa (Kiczales y otros, 2007).

Gregor Kiczales, creador del paradigma de programación orientada a aspectos (POA) define a un aspecto como “Una unidad modular que se disemina por la estructura de otras unidades funcionales. Los Aspectos existen tanto en la etapa de diseño como en la de implementación. Un Aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del diseño.

Un Aspecto de implementación es una unidad modular del programa que aparece de otras unidades modulares del programa”.

Lo que propone es agrupar los lenguajes orientados a objetos, los procedurales y funcionales como lenguajes de procedimiento generalizado (LPG), ya que sus mecanismos claves de abstracción y composición pueden verse como agrupados bajo una misma raíz. Los métodos de diseño que han surgido para los LPG tienden a dividir el sistema en unidades de comportamiento o funciones. En palabras simples un aspecto puede ser definido como: “La encapsulación y modularización de un asunto transversal”.

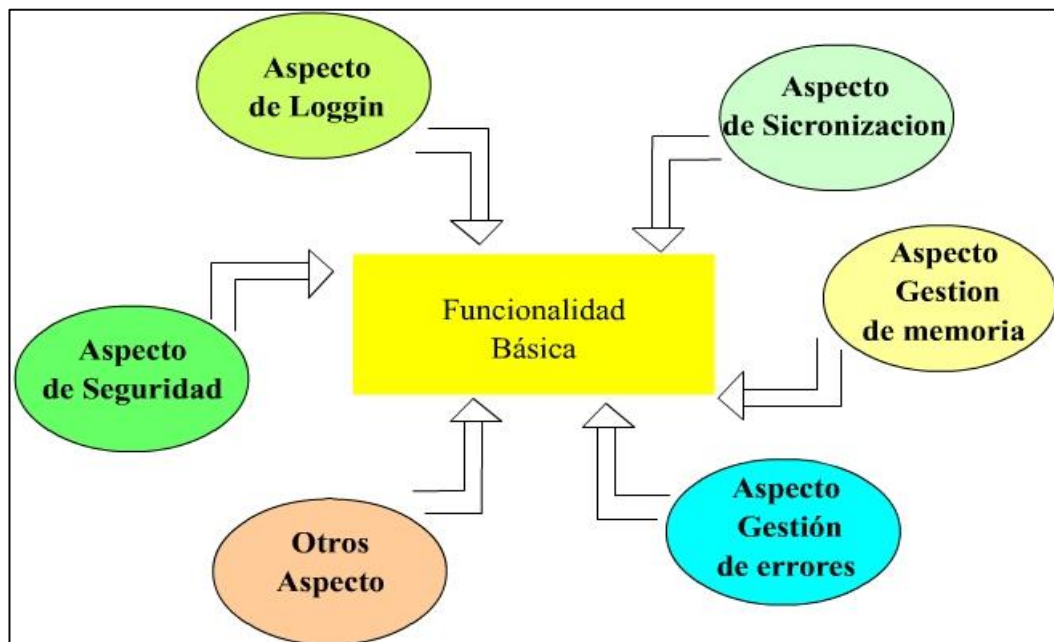


Figura 03: Interacción entre aspectos y funcionalidad básica.

Fuente: Barraza Olinda Velarde - Introducción a la programación orientada a objetos - México: Pearson Educación, 2006, Pag. 98

Se puede ver gráficamente cómo se interrelaciona los componentes base (requerimientos funcionales o funcionalidad básica) de un programa, con los diversos tipos de aspectos que lo afectan. Se observa que en el centro se encuentra un rectángulo representando la funcionalidad básica en donde coexisten los requerimientos funcionales y a su alrededor diferentes burbujas que representan aspectos, éstos se comunican generalmente con el sistema en forma de requerimientos no funcionales.

El enfoque orientado a aspectos define un mecanismo que ayuda a resolver problemas de codificación en los requisitos, los cuales son un código disperso y diseminado, que no se resuelven fácilmente usando el enfoque orientado a objetos. Este mecanismo se enfoca principalmente en la separación de intereses de un sistema para obtener una mejor modularización.

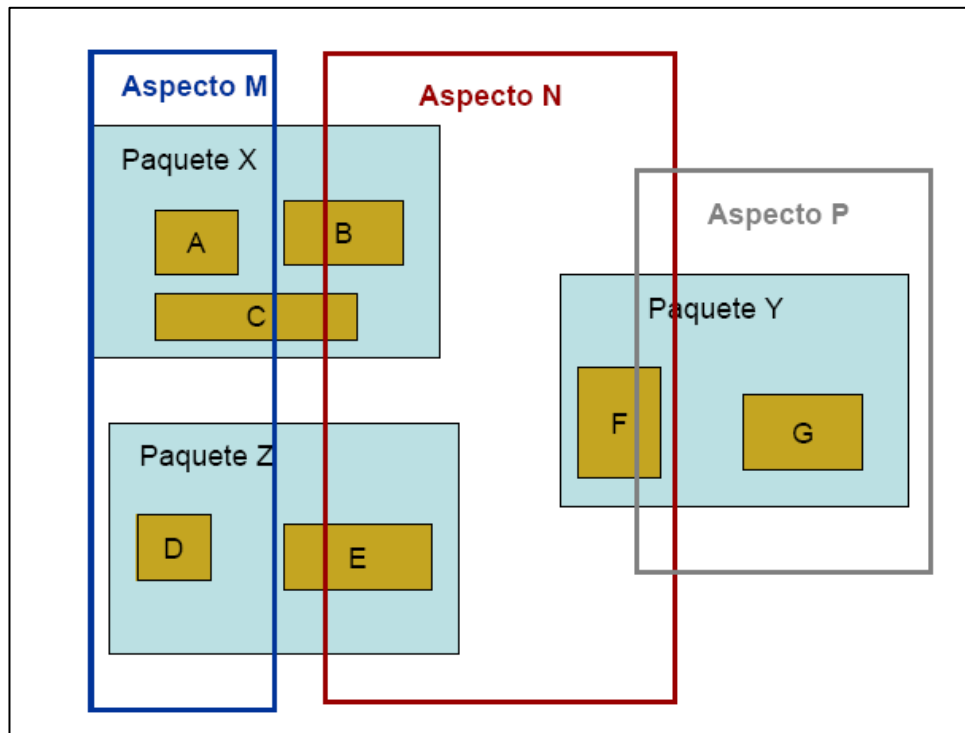


Figura 04: Separación de intereses

Fuente: Sommerville Ian, Software Engineering 9th Libro. – 2011, pág. 211

En general, se puede decir que dos propiedades se entrecruzan si al implementarse deben componerse de manera diferente, y aún deban ser coordinadas. Esto se debe a que tienen un comportamiento en común. Al proveer los LPG solamente un único medio de composición, es el programador quien debe realizar la tarea extra descomponer manualmente las propiedades que se entrecruzan, lo cual lleva a un oscurecimiento y a un aumento de la complejidad del código.

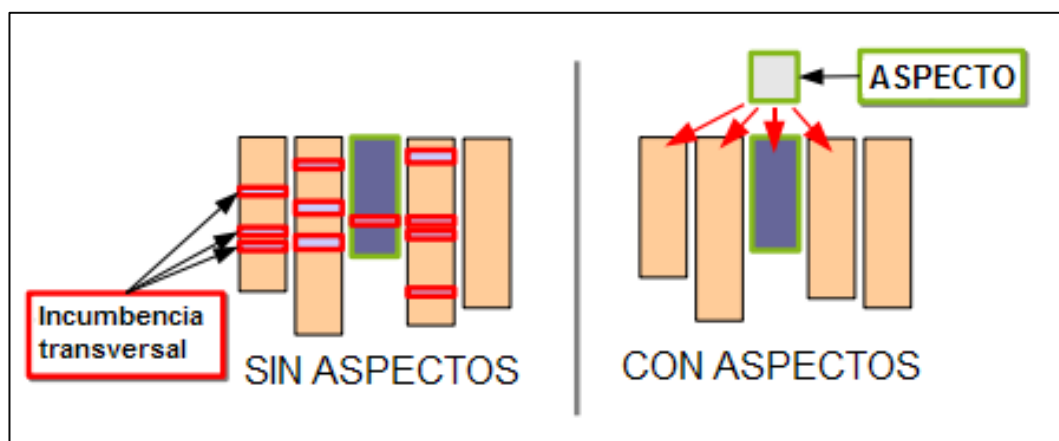


Figura 05: Implementación de incumbencias transversales sin y con aspectos

Fuente: Sommerville Ian, Software Engineering 9th Libro. – 2011, pág. 242

Gracias a todo esto se puede diferenciar los aspectos de los demás componentes del sistema. Al momento de implementar una propiedad, se tiene que la misma tomará una de las dos siguientes formas:

- a) Un componente:** Si puede claramente encapsularse dentro de un procedimiento. Un elemento es claramente encapsulado si está bien localizado, es fácilmente accesible y resulta sencillo componerlo.
- b) Un aspecto:** Si no puede claramente encapsularse en un procedimiento. (Ejemplo: autenticación, Accesibilidad, sincronización, seguridad entre otros.).

Teniendo estas definiciones es posible definir un aspecto como una propiedad de un sistema que no puede ser encapsulada claramente y envuelve diversos componentes funcionales.

Las características más relevantes de los aspectos para ser considerados como tales son (Fernando Asteasuain, 2002):

- Un aspecto no debe interferir con otro.
- Cada aspecto debe ser claramente identificable.
- Los aspectos deben ser fácilmente intercambiables.

2.2.2.4 Fundamentos de la programación orientada a aspectos

El desarrollo de software orientado a aspectos se enfoca en crear una mejor abstracción modular del sistema. Incluye las siguientes fases:

1. Requerimientos
2. Análisis
3. Diseño
4. Implementación

La primera fase trata la separación de intereses tanto los funcionales como los no funcionales; los requisitos funcionales son modelados con casos de uso que representan la función básica del sistema y los requisitos no funcionales se representan con casos de uso de infraestructura. En el análisis y el diseño los casos de uso se representan en una estructura de composición y agrupa elementos de modelo que colaboran para lograr los requisitos del sistema tanto funcional como no funcional. En la implementación se genera el código de las clases y aspectos.

La idea de la solución de la orientación aspecto tiene por objeto de eliminar la transversalidad de incumbencias en un único módulo bien definido (llamado aspecto) que puede estar compuesto de forma automática con el resto del programa (la denominada "núcleo" o "base"). Cada técnica orientada a aspectos se define por un modelo Punto de corte. De acorde a esto se caracteriza por tres elementos: Join Points, Join point selection mechanism y un medio para especificar la semántica en puntos de unión (Keuler, 2011):

- **Join Points** (Punto de unión): Un punto de unión es un punto en un programa donde el comportamiento puede ser alterado, aumentado o eliminado. En otras palabras, los puntos de unión están considerando el tipo de interfaz entre los aspectos y elementos básicos. Ejemplo: unir puntos en el nivel de programación como son llamadas a métodos, acceso a campos, o llamadas de constructor.

Dicho de otra manera, un evento en un programa en ejecución en que puede ejecutarse el consejo asociado con un aspecto.

- **Join point selection mechanism:** Modelo de punto de unión (Punto de corte): Un punto de corte puede ser definida como una consulta en el propio programa. La manera más simple y trivial para capturar puntos de unión es enumerarlas ya sea utilizando sus nombres, ID's o cualquier propiedad pueden ser utilizados para su identificación exacta. Otra

técnica que puede ser visto como una extensión de la enumeración es capturar puntos de unión deseadas, por un patrón (en su mayoría un nombre o patrón de la firma). Dicho de otra forma el conjunto de eventos que se pueden referenciar en un punto de corte.

- **Adaptation (advice):** En términos orientados a aspectos sería “avisos”, se consideran como acciones que se ejecutan en cada punto de enlace dentro de un mismo punto de corte, estas acciones se traducen en rutinas o fragmentos de código.

En general, hay tres maneras de cómo el aviso puede ser introducido como son: antes, después, y alrededor.

- **Weaving (Tejido):** El proceso de transformación que pone elementos de base y aspectos juntos se llama "tejer", la incorporación de código del consejo (advice) en los puntos de enlace (join point) especificados por un tejedor (weaving) de aspectos.

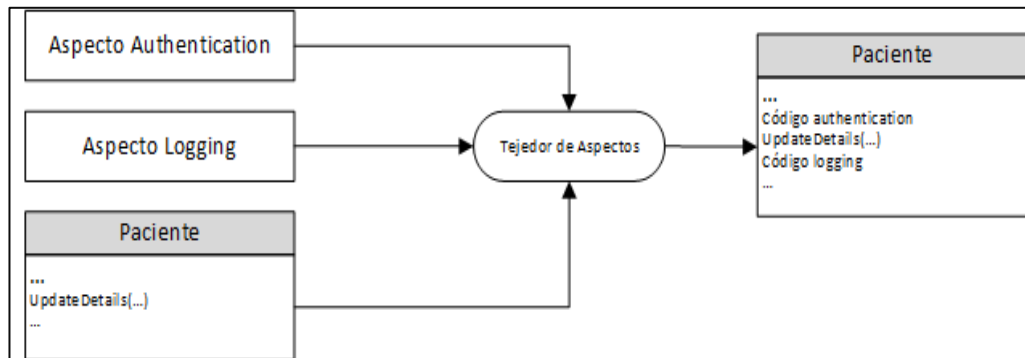


Figura 06: Función del tejedor de aspectos

Fuente: Quintero Antonia Maria Reina Visión general de la programación orientada a aspectos. - Sevilla, 2000. Pág. 143

Los elementos básicos de la programación orientada a aspectos, la funcionalidad básica (requerimientos funcionales, lenguaje base) es capturada por el tejedor de aspectos, y luego cada aspecto (incumbencia transversal) es tejido con la funcionalidad básica siendo los puntos de corte quienes indican en qué sección del código del lenguaje base deben escribirse.

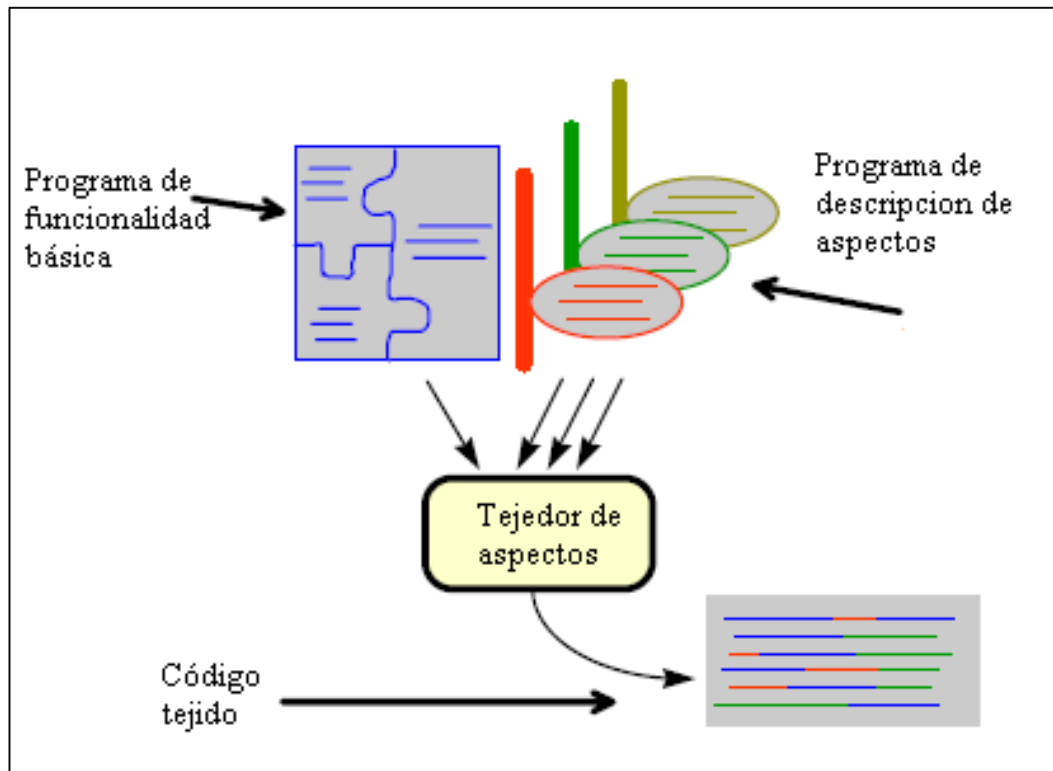


Figura 07: Proceso de tejido

Fuente: Quintero Antonia Maria Reina Visión general de la programación orientada a aspectos. - Sevilla, 2000. Pág. 145

Existen enfoques diferentes al tejido de aspectos con el código base:

Pre procesamiento de código fuente, en el que un tejedor toma entrada de código fuente y genera nuevo código fuente en un lenguaje como Java o C++, que luego pueden compilarse usando el compilador de lenguaje estándar. Este enfoque se adoptó para el lenguaje AspectX con su asociado XWeaver (Birrerr y otros, 2005).

Tejido de tiempo de vinculación, en el que el compilador se modifica para incluir un tejedor de aspectos. Un lenguaje orientado a aspectos, como AspectJ, se

procesa y genera bytecode Java estándar. Entonces éste puede ejecutarse directamente mediante un intérprete Java o procesarse aún más para generar código de máquina nativo.

Tejido dinámico en tiempo de ejecución. En este caso, se monitorizan los puntos de enlace y, cuando ocurre un evento referenciado en un punto de corte, se integra el consejo correspondiente con el programa en ejecución.

Respecto a esto Asteasuain señala que, el tejido estático (tejido de tiempo de vinculación) evita que el nivel de abstracción introducido por la programación orientada a aspectos (POA) derive en un impacto negativo en la eficiencia de la aplicación, ya que todo el trabajo se realiza en tiempo de compilación, y no existe sobrecarga en ejecución. Si bien esto es deseable el costo es una menor flexibilidad: los aspectos quedan fijos, no pueden ser modificados en tiempo de ejecución, ni existe la posibilidad de agregar o remover nuevos aspectos (Fernando Asteasuain, 2002).

Y sobre el tejido dinámico señala que el tejido dinámico implica que el proceso de composición se realiza en tiempo de ejecución, decrementando la eficiencia de la aplicación. El postergar la composición brinda mayor flexibilidad y libertad al programador, ya que cuenta con la posibilidad de modificar un aspecto según información generada en ejecución, como también introducir o remover

dinámicamente aspectos. La característica dinámica de los aspectos pone en riesgo la seguridad de la aplicación.

Se afirma que el enfoque usado más comúnmente para el tejido de aspectos es el tejido de tiempo de vinculación, pues esto permite la implementación eficiente de los aspectos sin una gran carga de tiempo de ejecución. El tejido dinámico es el enfoque más flexible, pero puede incurrir en significativas penalizaciones de rendimiento durante la ejecución del programa. Actualmente el pre procesamiento de código fuente se usa poco (Sommerville, 2011).

2.2.2.5 Beneficios e inconvenientes de la programación orientada a aspectos

La ingeniería de software como cualquier disciplina de la ingeniería está sujeta a costos y beneficios, la programación orientada a aspectos no está libre, los críticos de la programación orientada a aspectos (POA) a menudo hablan de lo difícil que es entender y de hecho, la POA necesita tiempo, paciencia y práctica, POA exige que se piense en el diseño e implementación del sistema de una manera nueva.

a) Beneficios:

- Algunas de las principales ventajas del paradigma de Programación Orientado a Aspectos son (Laddad, 2002):
- Ayuda a superar los problemas de codificación como son código mezclado y código diseminado.

- Implementación modularizada, logra separar cada concepto con mínimo acoplamiento.
- Mayor posibilidad de evolución, La separación de conceptos permite agregar nuevos aspectos, modificar y/o remover aspectos existentes fácilmente.
- Mayor reusabilidad: Al ser implementados separadamente, tienen mayores probabilidades de ser reusados en otros sistemas con requerimientos similares
- Permite N-dimensiones.
- Principio de mínimo acoplamiento y máxima cohesión.

La implementación de cada interés en forma independiente facilita el mantenimiento del sistema y la introducción de modificaciones o nuevas funcionalidades.

b) Inconvenientes:

- Para los sistemas orientados a aspectos, los procesos de las pruebas de validación no son diferentes a los de cualquier otro sistema. El programa final ejecutable se trata como una **caja negra** y se diseñan pruebas para demostrar si el sistema cumple o no con los requerimientos. Sin embargo, el uso de aspectos causa problemas reales con las inspecciones de programa y las pruebas de **caja blanca**, mientras que el código fuente

del programa se usa para identificar potenciales pruebas de defecto (Sommerville, 2011).

- En un sistema orientado a aspectos, es imposible la lectura de código secuencial. El lector debe examinar cada aspecto, comprender sus puntos de corte (que pueden ser patrones) y el modelo de punto de enlace del lenguaje orientado a aspectos.

(Cugola y otros, 1999) Refieren:

- Posibles conflictos entre código funcional (código base) y otros aspectos. En general tales enfrentamientos resultan de la necesidad de romper la encapsulación de funcionales unidades para implementar un aspecto diferente.
- Posibles enfrentamientos entre aspectos. Los dos aspectos funcionan perfectamente cuando se ejecutan individualmente, pero al aplicarlos en conjunto resulta en un comportamiento fallido.
- Posibles choques entre el código de aspecto y mecanismos del lenguaje de especificaciones. Uno de los ejemplos más conocidos de problemas que cae dentro de esta categoría es la anomalía de herencia.

2.2.3 Módulo de autenticación y diseño de software

2.2.3.1 Definición

En la seguridad de ordenador, la autenticación es el proceso de intento de verificar la identidad digital del remitente de una comunicación como una petición para conectarse. El remitente siendo autenticado puede ser una persona que usa un ordenador, un ordenador por sí mismo o un programa del ordenador. En un web de confianza, "autenticación" es un modo de asegurar que los usuarios son quien ellos "dicen" ser - que el usuario que intenta realizar funciones en un sistema es de hecho el usuario que tiene la autorización para hacer así.

El diseño crea una representación o modelo del software, pero a diferencia del análisis (que se enfoca en la descripción de los datos, las funciones y el comportamiento requerido), el modelo de diseño proporciona detalles acerca de las estructuras de datos, las arquitecturas, las interfaces, y los componentes del software que son necesarios para implementar el sistema. El diseño permite al ingeniero de software modelar el sistema o producto en relación con su calidad y mejorarse antes de generar código, de realizar pruebas y de que los usuarios finales se vean involucrados a gran escala. El diseño es el sitio donde se establece la calidad del software (Pressman, 2009).

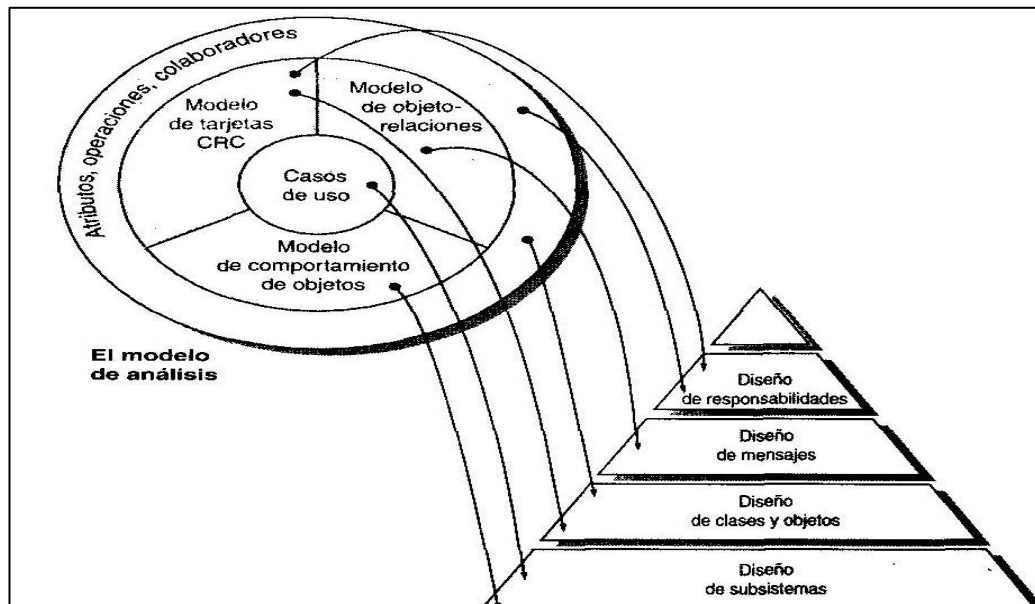


Figura 08: Transformación del modelo de análisis al modelo de diseño

Fuente: Fernando Asteasuain Bernardo Ezequiel Contreras Programación orientada a aspectos - Análisis del paradigma. - Buenos aires, 2002. Pág. 78

El diseño arquitectónico define la relación entre los elementos estructurales más importantes del software, los estilos arquitectónicos y patrones de diseño que pueden usarse para satisfacer los requisitos definidos por el sistema, y las restricciones que afectan la manera en que se pueden implementar los patrones arquitectónicos. La representación del diseño arquitectónico puede derivarse de la especificación del sistema del modelo de análisis y de la interacción de los subsistemas definidos dentro del modelo del análisis (Pressman, 2009).

El diseño de la interfaz describe la forma en que el software se comunica con los sistemas que interactúan con él y con los humanos que los utilizan. Una interfaz implica un flujo de información (por ejemplo, datos o control) y un tipo de

comportamiento específico. Por lo tanto, los escenarios de uso y los modelos de comportamiento proporcionan mucha de la información que se requiere en el diseño de la interfaz.

2.2.3.2 Importancia del diseño

La importancia del diseño del software puede describirse con una sola palabra: Calidad. El diseño es la etapa en la que se fomentará la calidad en la ingeniería del software. El diseño proporciona las representaciones del software susceptibles de evaluar de la calidad. El diseño es la única forma en que, de manera exacta, un requisito del cliente se puede convertir en un sistema o producto de software terminado. El diseño del software sirve como fundamento para todas las actividades subsecuentes de la ingeniería del software y del soporte de éste. Sin diseño se corre el riesgo de construir un sistema inestable, el cual fallará cuando se realicen cambios pequeños que será difícil de probar cuya finalidad no podrá evaluarse sino hasta etapas tardías del proceso del software, cuando queda poco tiempo y ya se ha gastado mucho tiempo en él (Pressman, 2009). El Diseño es la única manera de materializar con precisión los requerimientos del cliente.

2.2.3.3 Proceso y calidad del diseño

Se pueden considerar tres características que sirven como guía en la evaluación de un buen diseño:

- El diseño debe implementar todos los requisitos explícitos contenidos en el modelo de análisis, y debe ajustarse a todos los requisitos implícitos que desea el cliente.
- El diseño debe de ser una guía legible y comprensible para quienes generan código y quienes realizan pruebas y, en consecuencia dan soporte al software.
- El diseño debe proporcionar una imagen completa del software, dando dirección a los dominios de datos, funcionales y de comportamiento, desde una perspectiva de implementación.

2.2.3.4 Criterios técnicos para un buen diseño

- Un diseño debe de ser modular, y a su vez dividirse en subsistemas con características funcionales independientes.
- Debe presentar una estructura arquitectónica que se haya creado mediante patrones de diseño reconocibles, la integren componentes que exhiban buenas características de diseño, y puedan implementarse de manera evolutiva.
- Un diseño debe conducir a interfaces que reduzcan la complejidad de las conexiones entre los componentes y el ambiente externo.

CAPÍTULO III

DESARROLLO

3.1 Descripción del caso de estudio

Como implementación del caso de estudio se presenta el diseño de un módulo de autenticación para el software de la biblioteca de la liga departamental de futbol de Tacna, en el primer caso con la programación orientada a objetos y el segundo caso con la programación orientada a aspectos. Posteriormente se procedió a cuantificar las líneas de código, la complejidad ciclomática y el rendimiento computacional (uso de memoria RAM) para cada uno de los dos casos que se están comparando. Para poder concluir cuál de estos paradigmas tiene mejor eficiencia, se contabilizó la cantidad de líneas de Código (LCD), se usó la prueba estadística “t de Student” para comparar las medias estadísticas de los resultados obtenidos del rendimiento computacional.

3.2 Diseño de un software orientado a objetos

Para el desarrollo del siguiente caso, se optó por seguir la metodología rational unified process, más conocido como RUP, porque permite seleccionar el conjunto de componentes de proceso para que se ajusten a las necesidades específicas del proyecto, además ofrece una guía en cada etapa del desarrollo del software.

Se necesita un enfoque integral para el desarrollo de sistemas de software con los aspectos de los requisitos, el análisis y el diseño, hasta la implementación y prueba (Jacobson y otros, 2004).

3.2.1 Requerimientos

Los requerimientos para un sistema son descripciones de lo que el sistema debe hacer: el servicio que ofrece y las restricciones en su operación. Tales requerimientos reflejan las necesidades de los clientes por un sistema que atienda cierto propósito, como sería controlar un dispositivo, colocar un pedido o buscar información (Sommerville, 2011).

Roger S. Pressman, conocido ingeniero de software y autor de varios libros de Ingeniería del Software, expresa que, para que un esfuerzo de desarrollo de software tenga éxito, es esencial comprender perfectamente los requisitos del software. Independientemente de lo bien diseñado o codificado que esté un programa, si se ha analizado y especificado pobremente, decepcionará al usuario y desprestigiará al que lo ha desarrollado.

Según este autor en esta fase se identifican dos categorías de casos de uso: uno de aplicación y otro de infraestructura. Los casos de uso de aplicación representan los requerimientos funcionales del sistema y los casos de uso de infraestructura representan las cualidades del sistema que dan soporte para cada caso de uso de aplicación.

Partiendo de estas definiciones queda claro que el objetivo del requerimiento es describir lo que debería hacer el módulo y permitir al desarrollador y al cliente ponerse de acuerdo con esa descripción.

Los casos de uso ayudan a capturar los requerimientos funcionales de sistemas desde la perspectiva del usuario, de esta forma da una visión temprana del sistema antes de su implementación. Aunque existen diversas aproximaciones para realizar los requerimientos de los sistemas, los diagramas de casos de uso brindan una perspectiva gráfica entendible entre usuario y analista para el modelado inicial del sistema, con esto se minimizan incongruencias, incompatibilidades, redundancias entre los requerimientos en lenguaje natural del usuario.

Un modelo de caso de uso describe lo que hace un sistema sin describir cómo lo hace, es decir, es un modelo lógico del sistema (Kendall, 2005). Kendall sugiere algunos lineamientos para la identificación:

- Revise las especificaciones del negocio e identifique los actores en el dominio del problema.
- Identifique los eventos de alto nivel y desarrolle los casos de uso principales que describen dichos eventos y cómo los inician los actores. Examine cuidadosamente los papeles jugados por los actores para identificar todos los posibles casos de uso principales iniciados por cada

actor. No se necesita mostrar los casos de uso con poca o ninguna interacción del usuario.

- Revise cada caso de uso principal para determinar las posibles variaciones de flujo a través del caso de uso. Con este análisis, establezca las rutas alternativas. Debido a que el flujo de eventos normalmente es diferente en cada caso, busque actividades que podrían tener éxito o fallar. También busque cualesquier ramas en la lógica de caso de uso en que son posibles resultados diferentes.

Habiendo visto los detalles de la etapa de requerimientos, se procedió a capturar e identificar el escenario, al actor y su interacción en relación con el módulo.

Diagrama de Casos de uso - Autenticación de usuarios

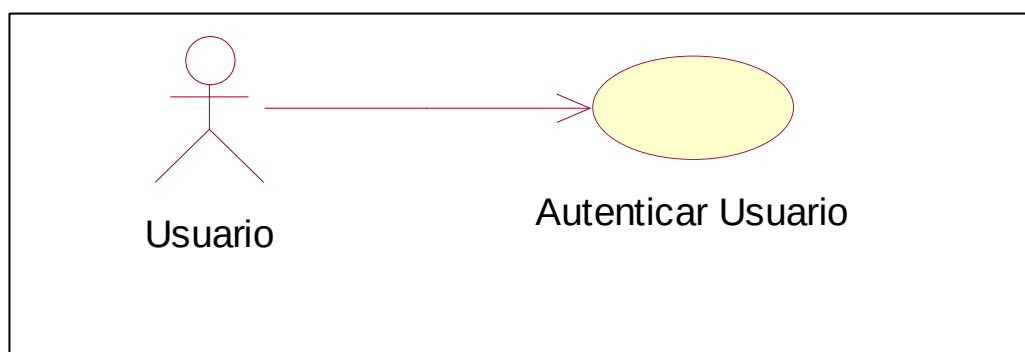


Figura 09: Caso de uso - autenticar usuario

Fuente: Elaboración propia

Tabla 4:*Caso de uso autenticar usuario*

Caso de uso	Autenticar Usuario	
Actor Principal	Usuario del Sistema	
Resumen	Permite a un usuario acceder al sistema, introduciendo su nombre de usuario y su contraseña.	
Pre-condiciones	El usuario debe estar registrado en el sistema	
Post-condiciones	El usuario ha accedido al sistema.	
Flujo principal	Acción del Actor	Responsabilidad del Sistema
	1. Introduce su nombre de usuario y su contraseña.	
		2. Valida los datos que el usuario ha introducido.
		3. Permite al usuario acceder a la aplicación.
El usuario no introduce los datos	4. Accede al sistema.	
	Acción del Actor	Responsabilidad del Sistema
	2. Introduce los datos y vuelve al paso 2 del flujo	1. Pide de nuevo los datos.
		3. No se introducen los datos y vuelve al paso 1.
El usuario o la contraseña no son correctas	Acción del Actor	Responsabilidad del Sistema
		1. Notifica al usuario el error en los datos.
	3. Introduce de nuevo los datos.	2. Pide que se introduzcan los datos de nuevo.
		4. Vuelve al paso 2 del flujo principal.

Fuente: Elaboración propia

Hasta esta etapa se han identificado los principales escenarios del proceso de autenticación, además se identificó al principal actor del sistema que es el usuario, quien desarrolla la dinámica del proceso.

3.2.2 Análisis

El análisis se enfoca en la producción de un modelo del sistema, llamado el modelo de análisis, que es correcto, completo, consistente y verificable. El análisis se diferencia de la obtención de requerimientos en que los desarrolladores se enfocan en la estructuración y formalización de los requerimientos obtenidos de los usuarios (Bruegge y otros, 2002).

El análisis es un refinamiento y estructuración de los requisitos capturados en la actividad anterior (requerimientos). El objetivo del análisis es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de entender (Jacobson, 2010).

El análisis orientado a objetos es un método de análisis que examina los requisitos desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del problema (Booch, 2009).

Los objetos de entidad representan la información persistente rastreada por el sistema. Los objetos de frontera representan la interacción entre los actores y el sistema. Los objetos de control representan las tareas realizadas por el usuario y soportadas por el sistema (Bruegge y otros, 2002).

Tiene ventajas como proporcionar a los desarrolladores una heurística simple para distinguir conceptos diferentes pero relacionados, el enfoque de tres tipos de objetos da como resultado objetos más pequeños y más especializados, conduce a modelos que son más adaptables al cambio.

Las clases del análisis luego evolucionan hacia otras clases más detalladas en el Modelo del Diseño. A continuación, se procede a la realización de los casos de uso hacia el modelo de análisis en un contexto estático.

Modelo de análisis de autenticación de usuarios

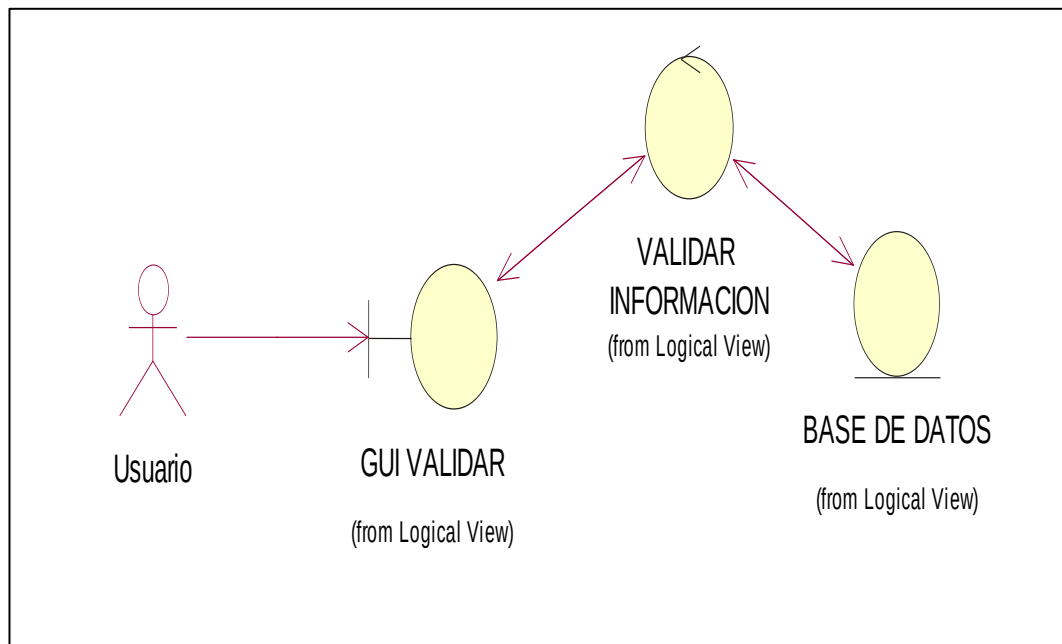


Figura 10: Modelo de análisis - autenticar usuario

Fuente: Elaboración propia

Hasta el momento se ha identificado la interacción de las clases del análisis que deben tener en los casos de uso en un contexto estático, pero no se ha identificado la interacción que deben tener dentro de los casos de uso, como se lleva o se ejecuta, por ejemplo se sabe que el usuario realiza una búsqueda y también se sabe que el administrador realiza un registro de un usuario, se conoce sus responsabilidades, pero, no se sabe cómo interactúan (los objetos) para llevar la realización del caso de uso, lo que nos lleva a desenvolver un modelo dinámico del sistema con las clases identificadas con diagramas de secuencias.

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. Muestra cómo se distribuye el comportamiento de un caso de uso (o escenario) entre sus objetos participantes. El diagrama de secuencia es un tipo de diagrama usado para modelar interacción entre objetos en un sistema según UML. Los diagramas de secuencia no son, por lo general, un buen medio de comunicación con el usuario. Sin embargo, representan otro cambio de perspectiva y permiten que los desarrolladores encuentren objetos faltantes o áreas grises en la especificación del sistema (Bruegge y otros, 2002).

Entonces basados en los modelos del análisis y en los casos de uso se hizo la descripción de la interacción de los objetos en un diagrama de secuencias.

Diagrama de actividades

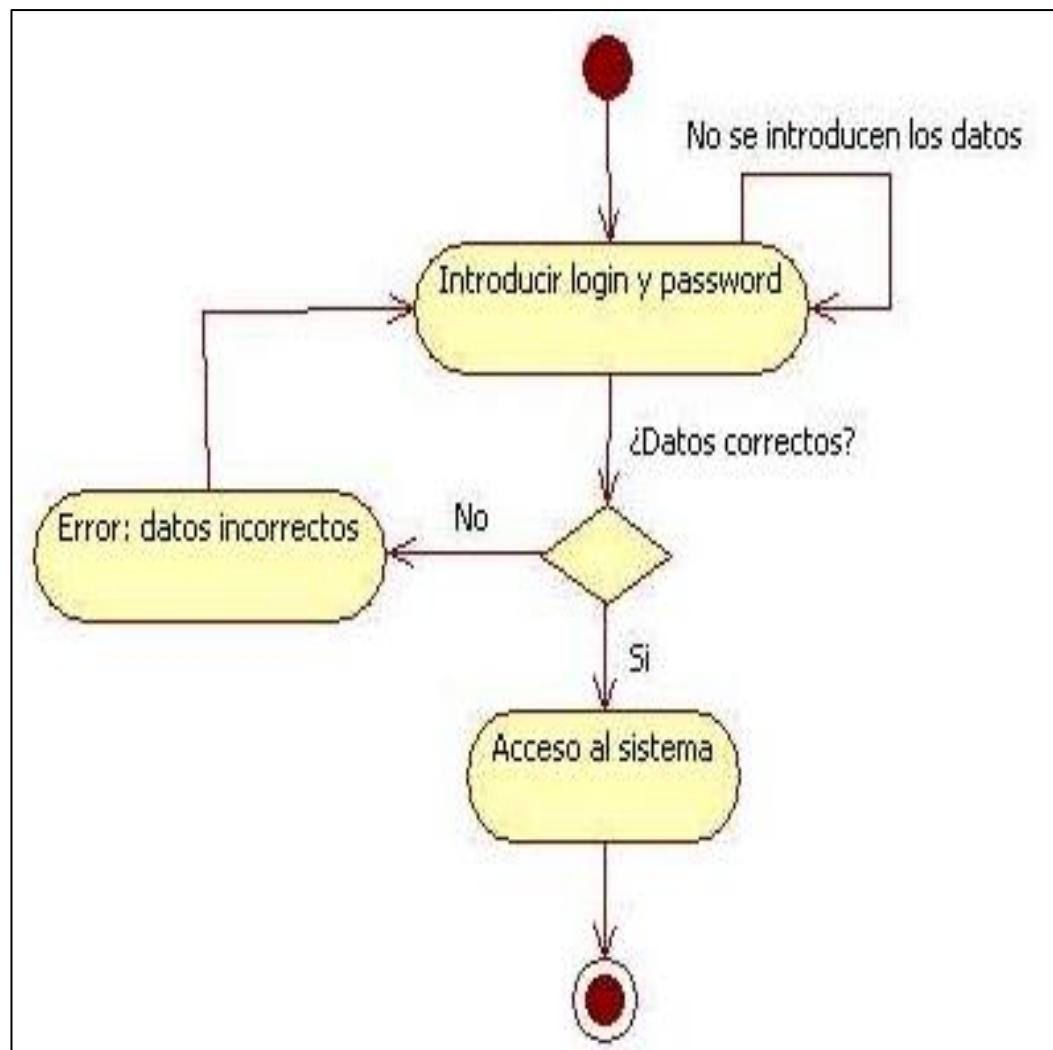


Figura 11: Diagrama de actividades - autenticar usuario

Fuente: Elaboración propia

Diagrama de secuencia de autenticación de usuarios

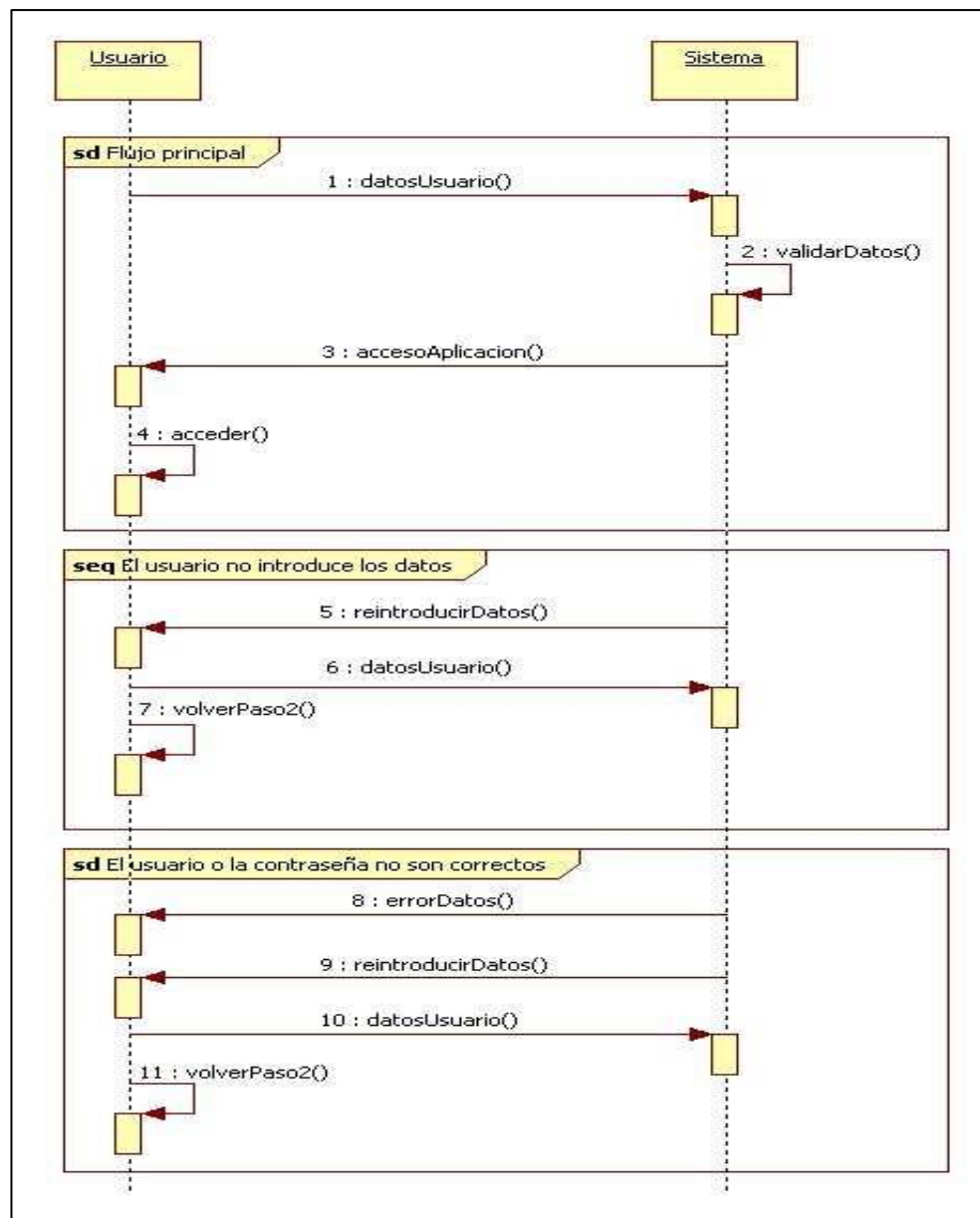


Figura 12: Diagrama de secuencia - autenticar usuario

Fuente: Elaboración propia

3.2.3 Diseño

El diseño de sistemas es la transformación del modelo de análisis en un modelo de diseño del sistema. Durante el diseño del sistema los desarrolladores definen los objetivos de diseño del proyecto y descomponen el sistema en subsistemas más pequeños que pueden ser realizados por equipos individuales. Los desarrolladores también seleccionan estrategias para la construcción del sistema como la plataforma de hardware y software en la que se ejecutará el sistema, la estrategia de almacenamiento de datos persistentes, el flujo de control global, la política de control de acceso y el manejo de condiciones de frontera (Bruegge y otros, 2002).

El diseño de sistemas consiste en la transformación del modelo de análisis en el modelo de diseño, que toma en cuenta los requerimientos no funcionales y las restricciones descritas en el enunciado del problema y el documento de análisis de requerimientos.

La obtención de requerimientos con los casos de uso y el modelo de análisis se enfocó en la funcionalidad del sistema, ahora en el diseño se enfoca en las estructuras de datos, componentes de software y hardware necesarios para implementarlo. Así mismo, se llenó el vacío entre la especificación, el análisis y el sistema que se entregó a los usuarios.

El diseño del sistema es el primer paso en esta dirección. El diseño del sistema da como resultado los siguientes productos:

- Una lista de objetivos de diseño que describe las cualidades del sistema que deben optimizar los desarrolladores.
- Una arquitectura de software que describe la descomposición en subsistemas desde el punto de vista de responsabilidades del subsistema, dependencias entre subsistemas, correspondencia de los subsistemas con el hardware y decisiones de política principales, como el flujo de control, control de acceso y almacenamiento de datos.

a) Identificación de los objetivos de diseño

Esta sección está centrada básicamente en las cualidades que debe tener el sistema, y se hizo uso de las especificaciones de pre y post condiciones de los casos de uso, así como las notas donde se señala algunas restricciones. Muchos de los objetivos de diseño pueden inferirse a partir de los requerimientos no funcionales, o inferirse de los requerimientos funcionales como forma de respaldar a las funciones realizadas.

b) Arquitectura de software

La arquitectura de software es importante porque afecta el desempeño y la potencia, así como la capacidad de distribución y mantenimiento de un sistema (Bosch, 2000). Como afirma Bosch, los componentes individuales implementan los requerimientos funcionales del sistema. Los requerimientos no funcionales dependen de la arquitectura del sistema, es decir, la forma en que dichos componentes se organizan y comunican. En muchos sistemas, los requerimientos no funcionales están también influidos por componentes individuales, pero no hay duda de que la arquitectura del sistema es la influencia dominante (Sommerville, 2011).

Entonces se ve necesario establecer la arquitectura del software para contar con una estructura con la cual se podrá desarrollar la implementación. Se elige la arquitectura del modelo vista controlador (citado en el marco teórico) debido a la facilidad de ver e interactuar con múltiples datos, además al no conocerse requerimientos futuros permite que se implementen fácilmente para la interacción y presentación. Brinda la ventaja de separar la lógica de la aplicación de la presentación con este mínimo código, ya que las vistas están separadas del código.

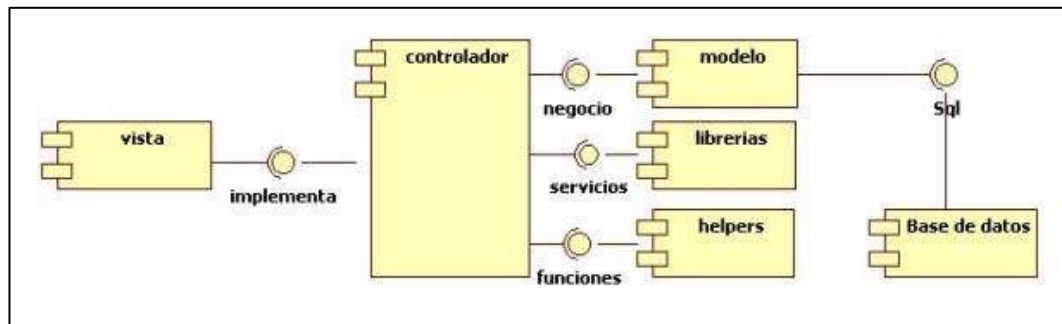


Figura 13: Diagrama de componentes modelo vista controlador

Fuente: Booch Grady Análisis y diseño de orientado a Objetos - 2009. Pág. 188

En el diagrama de componentes se puede observar la interacción de los componentes modelo vista controlador, además de la colaboración de paquetes de librerías y helpers con el objeto de facilitar la elaboración del módulo de autenticación.

Habiendo detallado la arquitectura del software en un entorno conceptual, queda por definir el soporte para el diseño del software, concentrarnos en el hardware, un diseño físico que dé soporte a la arquitectura del diseño del software.

El diagrama de despliegue ilustra la implementación física de sistema, incluyendo el hardware, las relaciones entre hardware y el sistema en que se despliega (Kendall, 2005). El modelo de despliegue representa en otras palabras la correspondencia entre la arquitectura del software elegido y la arquitectura del sistema.

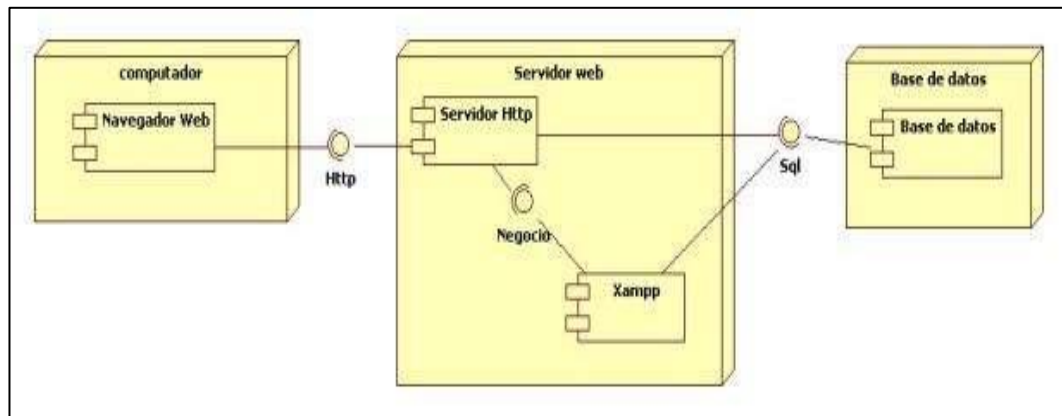


Figura 14: Diagrama de despliegue del sistema

Fuente: Booch Grady Análisis y diseño de orientado a Objetos - 2009. Pág. 194.

A continuación, se presenta un diagrama de clases en el que se muestran tan sólo las clases de entidad realizadas en el análisis del sistema ya que representan la persistencia en la aplicación, así como sus asociaciones y de un modelo de datos que sería la transformación de clases al modelo relacional de base de datos donde se muestran las tablas que participan en las relaciones definidas en el diseño del software orientado a objetos. es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos. El diagrama de clases incluye mucha más información como la relación entre un objeto y otro, la herencia de propiedades de otro objeto, conjuntos de operaciones/propiedades que son implementadas para una interfaz gráfica.

Diagrama de clases

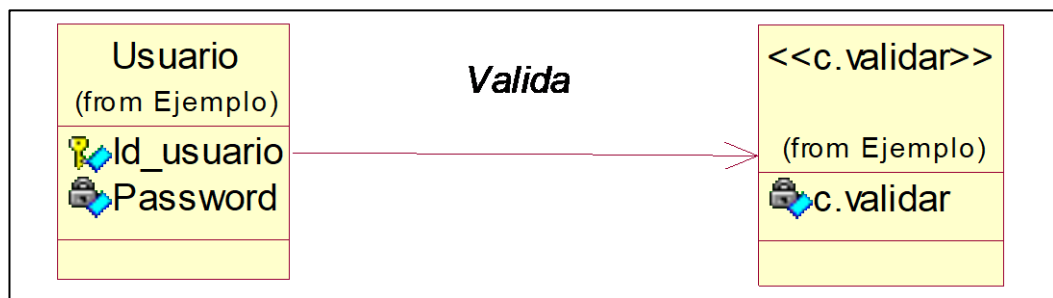


Figura 15: Diagrama de clases - autenticar usuario

Fuente: Elaboración propia.

3.2.4 Implementación

a) Elección del lenguaje de programación

PHP es un lenguaje de programación usado generalmente para la creación de contenido para sitios web. PHP es un acrónimo recurrente que significa "PHP Hypertext Pre – Processor" (inicialmente PHP tools, o, personal home page tools), y se trata de un lenguaje interpretado usado para la creación de aplicaciones para servidores, o creación de contenido dinámico para sitios web. Su interpretación y ejecución se da en el servidor, en el cual se encuentra almacenado el script, y el cliente sólo recibe el resultado de la ejecución. Se elige este lenguaje por tratarse de un lenguaje gratuito por lo que se presenta como una alternativa de fácil acceso para todos, además que es un lenguaje multiplataforma, brinda soporte para diferentes manejadores de base de datos entre ellos Mysql, posee una amplia documentación, y permite técnicas de programación orientada a objetos.

b) Elección de un framework para la arquitectura MVC

En la actualidad existen múltiples frameworks para PHP que dan soporte a la arquitectura modelo – vista – controlador. Pero dentro de ellos se eligió CodeIgniter debido a que es de fácil implementación, una curva de aprendizaje rápida y la codificación prácticamente se realiza desde cero, esto lo puede volver muy básico, pero lo que se busca en esta investigación es desarrollar prácticamente desde cero.

c) Implementación de la capa de persistencia

En este caso, la capa de persistencia viene implementada como una base de datos en PostgreSQL. Se ha elegido PostgreSQL por tratarse de una base de datos libre, así como por su buena integración de su uso con PHP. En general, sistemas en los que la velocidad y el número de accesos concurrentes sea algo primordial. Para el diseño de la base de datos se ha utilizado MySQL Workbench Visual Database Designer.

d) Implementación del módulo

En esta etapa se desarrolló la implementación del módulo de autenticación, dejando al software de la Biblioteca como un sistema dependiente de la validación de los datos ingresados por el usuario.

Para la implementación de este módulo se hizo uso del PostgreSQL, para crear una pequeña base de datos con perfiles de usuario para el momento de

validarlos. Se agregaron los perfiles de usuarios con relación a nuestras necesidades funcionales de un login y un password.

3.3 Diseño de un software orientado a aspectos

Para el desarrollo del siguiente caso, se optó por seguir el desarrollo de software orientado a aspectos con casos de uso. Porque brinda una amplia documentación acerca de todo el proceso de desarrollo de software orientado a aspectos con un enfoque práctico, promueve la separación de competencias desde tempranos inicios del software dando explicaciones sobre la forma de abordarlos y además especifica la tarea a realizar en cada etapa. (Jacobson y otros, 2004) Expresan que se necesita un enfoque integral para el desarrollo de sistemas de software con los aspectos de los requisitos, el análisis y el diseño, hasta la implementación y prueba.

3.3.1 Requerimientos

Según explican (Jacobson y otros, 2004): Es bien sabido que la orientación a aspectos ayuda a modularizar incumbencias transversales durante la implementación, pero hay una necesidad de modularizar incumbencias transversales mucho antes, incluso durante los requisitos. Los casos de uso son una excelente técnica para este propósito.

Los enfoques de modularidad convencionales como las clases y componentes que no pueden mantener las incumbencias transversales separadas.

Para resolver este problema, se necesitan dos cosas: una técnica de separación para mantener las incumbencias transversales independientes y un mecanismo de composición para componer estas incumbencias como parte del sistema deseado.

Son muchos los trabajos acerca de cómo identificar incumbencias transversales desde la identificación textual (Rago y otros, 2008) hasta herramientas de ayuda para su identificación: (Lorena y otros, 2013), (Weston y otros, 2013).

En casos de uso para aspectos, establecen reglas para el modelamiento de escenarios con aspectos, así como el tipo de relación entre un aspecto y un caso de uso a través de las relaciones extends, include y generalizaciones (Jacobson y otros, 2004).

Según este autor en esta fase se identifican dos categorías de casos de uso: uno de aplicación y otro de infraestructura. Los casos de uso de aplicación representan los requerimientos funcionales del sistema y los casos de uso de infraestructura representan las cualidades del sistema que dan soporte para cada caso de uso de aplicación.

Los términos aspectuales normalmente describen los atributos de calidad de un sistema como el de seguridad y rendimiento, descripciones de impacto de la influencia de uno de los requisitos sobre otros requisitos, y dispersas o descripción de los requisitos que aparecen repetidos en varios lugares en todo el documento de

requerimientos a menudo se consideran indicadores de las preocupaciones transversales en los requisitos.

Diagrama de casos de uso - validación de usuarios

Para la validación de usuarios será necesario que el usuario tenga una sesión registrada en el sistema, además que esté habilitado. En el registro de usuarios se debe realizar una validación de los datos que se ingresa cuidando que no se ingresen datos repetidos.

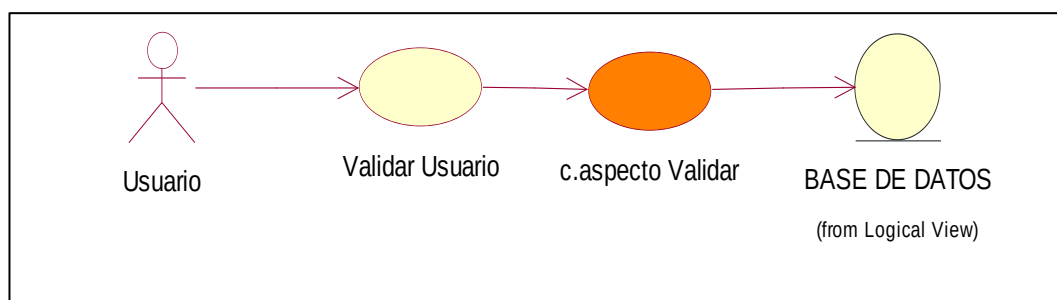


Figura 16: Caso de uso - validar usuario con POA

Fuente: Elaboración propia.

3.3.2 Análisis

Las especificaciones de casos de uso describen el comportamiento deseado del sistema desde el punto de vista externo utilizando el lenguaje de los usuarios finales, pero no describe cómo se relacionan las entidades que participan y además el papel que juega la nueva entidad llamada aspecto. Por ello se necesita una representación más precisa, esto es posible con el modelo de análisis.

El análisis es un refinamiento de los requisitos con el objetivo de crear especificación precisa de requisitos. Su propósito también es encontrar abstracciones en forma de clases que hacen los requisitos más claros en comparación a las especificaciones de casos de uso.

Según el modelo de análisis de Jacobson (2004), se identifican dos elementos:

- **Capa de aplicación**

La capa de aplicación contiene elementos que dan cuenta de los flujos de trabajo en los casos de uso de apoyo a los actores principales del sistema. Los elementos de esta capa normalmente utilizan los elementos de la capa de dominio para darse cuenta de casos de uso. Puede organizar los paquetes en la capa de aplicación de acuerdo con los siguientes criterios.

- **Capa de dominio**

La capa de dominio contiene elementos que representan conceptos de dominio significativos. Capturan información que se ha mantenido, seguido, o manipulados por el sistema y los comportamientos asociados para hacerlo. Estos elementos son normalmente compartidos a través de la realización de los casos de uso.

Se identifican los siguientes modelos de análisis incluyendo la capa de dominio y la capa de aplicación como la realización de los casos de uso en un contexto estático.

Modelo de análisis – validar usuario

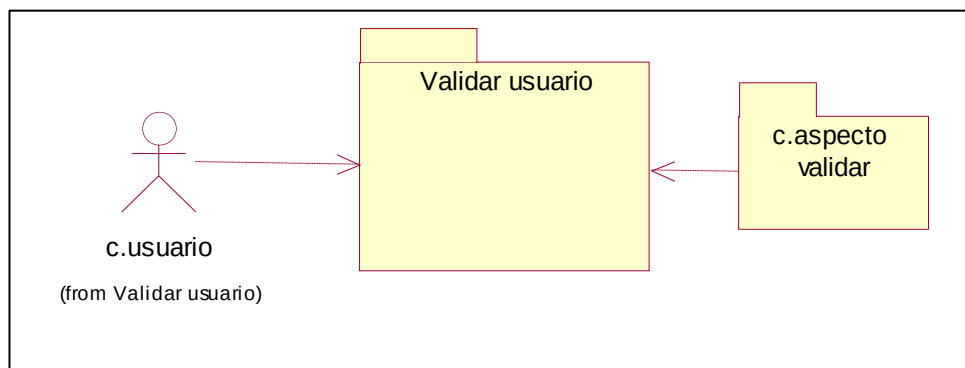


Figura 17: Validar usuario

Fuente: Elaboración propia.

Se ha descrito el comportamiento estático del sistema incluyendo clases de entidad, frontera, controlador y aspectos, pero aún no se ha definido la interacción entre estos. Un modelo dinámico ayuda a comprender este comportamiento, trabajos como (Vidal y otros, 2012), (Sánchez y otros, 2013), (Zhang y otros, 2009), describen la adaptación del diagrama de secuencias del UML para el modelado orientado a aspectos, con el objeto de lograr implementar el modelo dinámico en el cual participa un aspecto.

3.3.3 Diseño

El diseño orientado a aspectos es el proceso de diseñar un sistema que utilice los aspectos para implementar las competencias transversales y extensiones que se identificaron durante el proceso de ingeniería de requerimientos. En esta etapa es necesario traducir las competencias que se relacionan con el problema que hay que resolver a los aspectos correspondientes en el programa que implementará la solución. También es indispensable comprender cómo se combinarán estos aspectos con otros componentes de sistema y garantizar que no surjan ambigüedades de composición (Sommerville, 2008).

El modelo de diseño es un refinamiento del análisis realizado con los aspectos, se ha definido una estructura de alto nivel, ahora se definirá la arquitectura que dará soporte al sistema.

a) Análisis de conflictos entre aspectos

Una etapa adicional que se da en el diseño orientado a aspectos es la identificación temprana de conflictos entre aspectos para evitar que no generen un comportamiento impredecible en el sistema una vez implementado. Trabajos como (Sardinha y otros, 2013), (Casas y otros, 2006), (Casas y otros, 2005) (Hormazábal, 2010), (Brito y otros, 2013), proveen conocimiento acerca del tratamiento que se da para estructurar los aspectos de forma que solo se ejecuten

como se han diseñado. En términos simples “Un conflicto ocurre cuando dos o más aspectos compiten por una activación”.

b) Diseño de nombres

Para permitir un manejo global con la intervención de los puntos de corte se establecen nombres comunes en todas las clases como son registrar, actualizar, eliminar y buscar. Y demás nombres que se identificarán en el archivo aspectos.xml. Por ejemplo, para los nombres de registrar se ejecutarán los aspectos de autenticación, autorización, validación y looging.

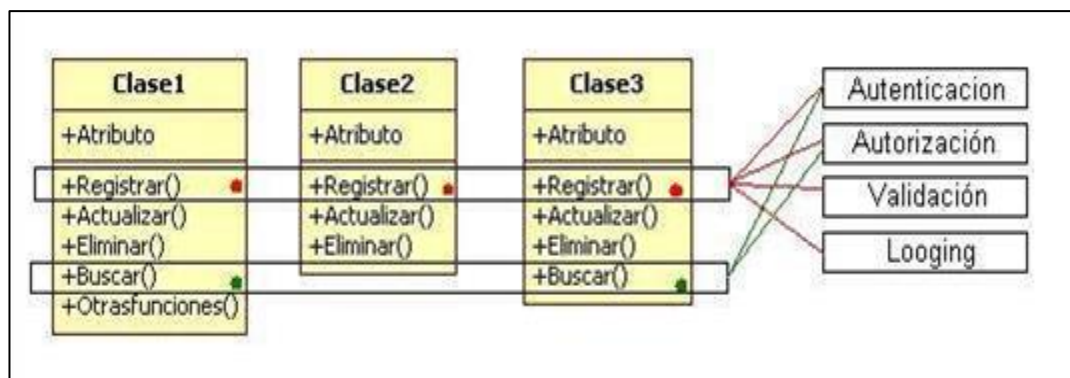


Figura 18: Ilustración de la ejecución de los puntos de corte

Fuente: Bahit Eugenia POO y MVC en PHP. - 2011. Pág. 69.

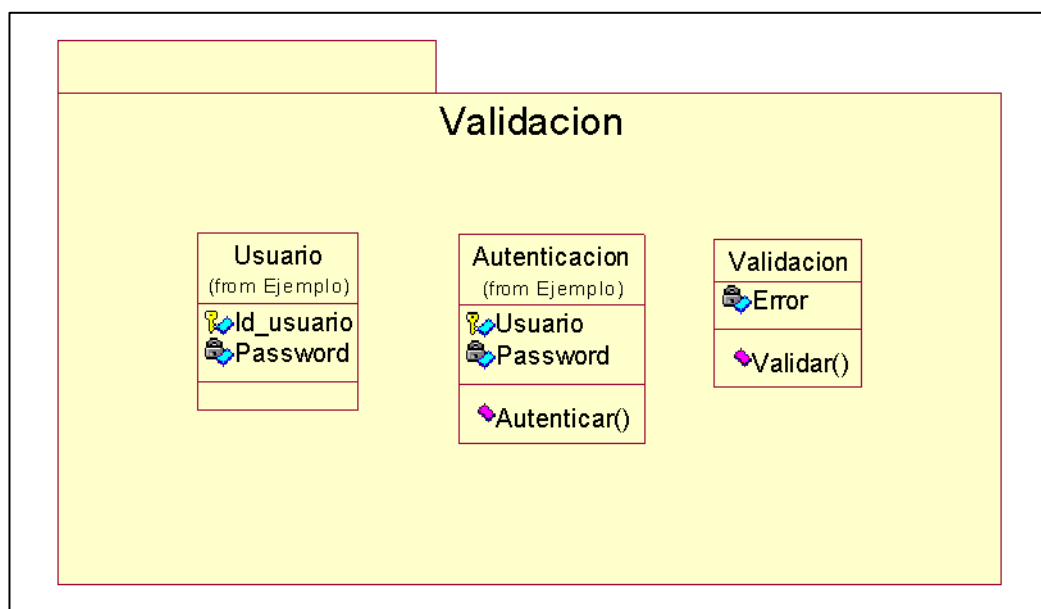


Figura 19: Estructura de elementos del diseño

Fuente: Elaboración propia.

c) Arquitectura de software

Una buena arquitectura mantiene incumbencias separadas, lo que significa que los cambios en una parte del sistema no se propagan a otras partes del sistema. Incluso si lo hacen, se puede identificar claramente qué cambios se deben hacer. Si existe la necesidad de extender la arquitectura, el impacto debería ser mínimo. Todo lo que ya trabaja debería seguir trabajando. Para un sistema que aplica la orientación aspecto, las diferentes incumbencias del sistema pueden mantenerse separadas de manera efectiva (Jacobson y otros, 2004). Se selecciona la arquitectura de modelo-vista-controlador con la inclusión de aspectos.

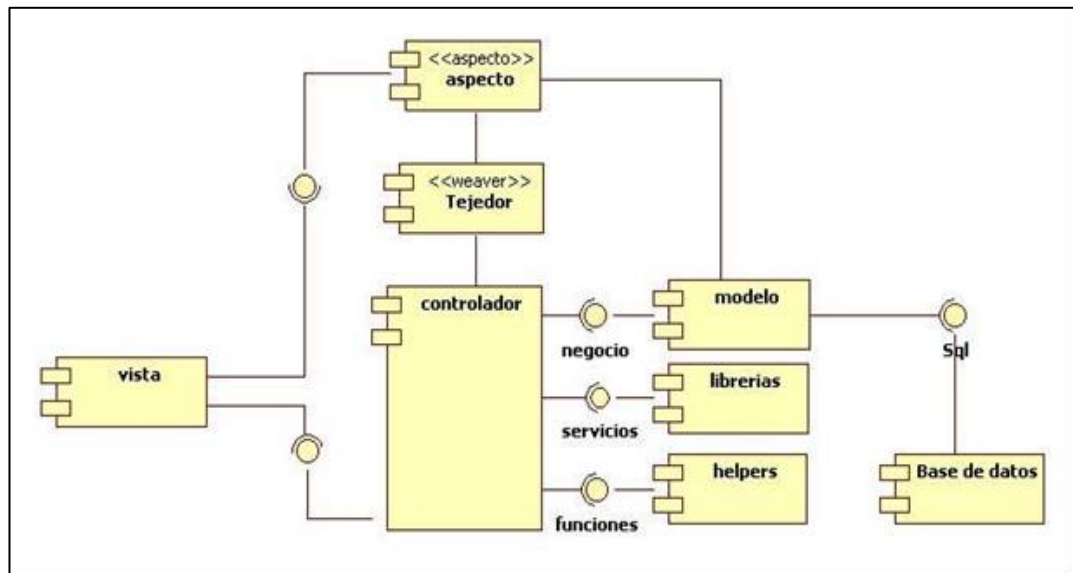


Figura 20: Diagrama de componentes MVC con la inclusión de aspectos

Fuente: Bahit Eugenia POO y MVC en PHP. - 2011. Pág. 70.

d) Estructura de implementación.

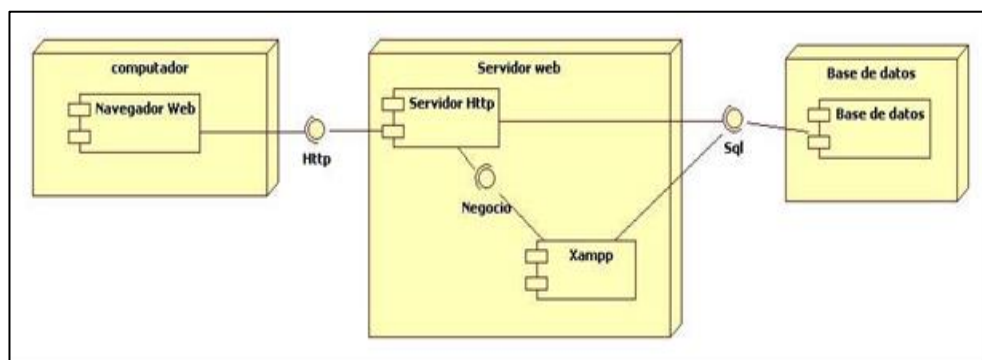


Figura 21: Diagrama de despliegue del módulo.

Fuente: Bahit Eugenia POO y MVC en PHP. - 2011. Pág. 74.

3.3.4 Implementación

a) Elección del lenguaje de programación

La elección del lenguaje de programación para la implementación del software con la programación orientada a aspectos se debe realizar hacia los objetivos de la investigación, así mismo, para que se realicen las pruebas bajo las mismas condiciones se elige PHP.

b) Elección de un framework para la arquitectura MVC

Existen frameworks orientados a aspectos en PHP como son:

- PHPAspect
- Aspect-Oriented PHP
- AOP-PHP
- GO! Aspect-Oriented
- MFAOPHP

En algunos casos como PHPAspect, Aspect-Oriented PHP, MFAOPHP no proveen soporte para la arquitectura MVC, en otro como GO Aop exige la instalación de dependencias lo que al realizarlo en el lado servidor escapa del alcance del proyecto, otro inconveniente está ligado a que no se realicen las pruebas bajo las mismas condiciones. Por esos motivos se elige CodeIgniter. Se realizó el sistema orientado a aspectos incluyendo JoinPoints, PointCuts, Advices y Aspectos.

c) Elección del tipo de tejido de los aspectos

Para lograr la mayor flexibilidad y por ende la mantenibilidad del sistema se opta por un tejido dinámico, ya que cuenta con la posibilidad de modificar un aspecto según la información generada en ejecución, como también introducir o remover dinámicamente aspectos. Además, PHP es un lenguaje de programación interpretado, no se vería el impacto de la programación orientada a aspectos con un tejido estático.

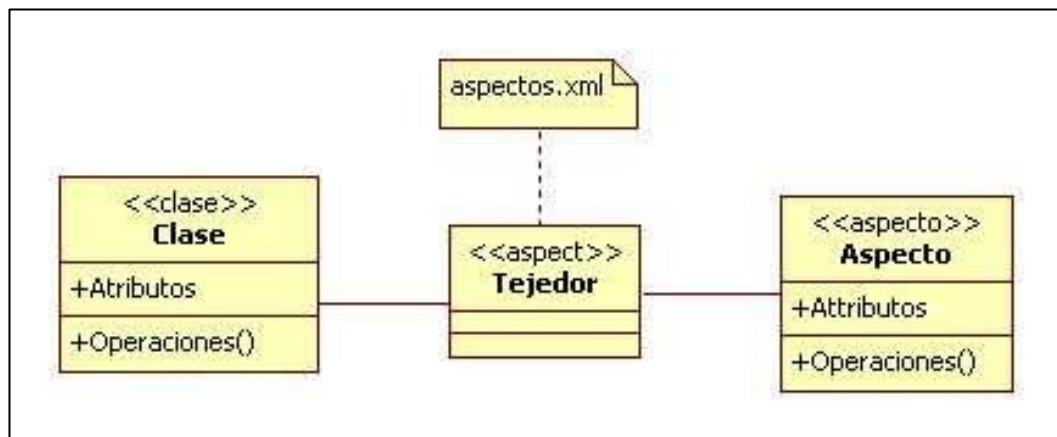


Figura 22: Tejido de aspectos con las clases base

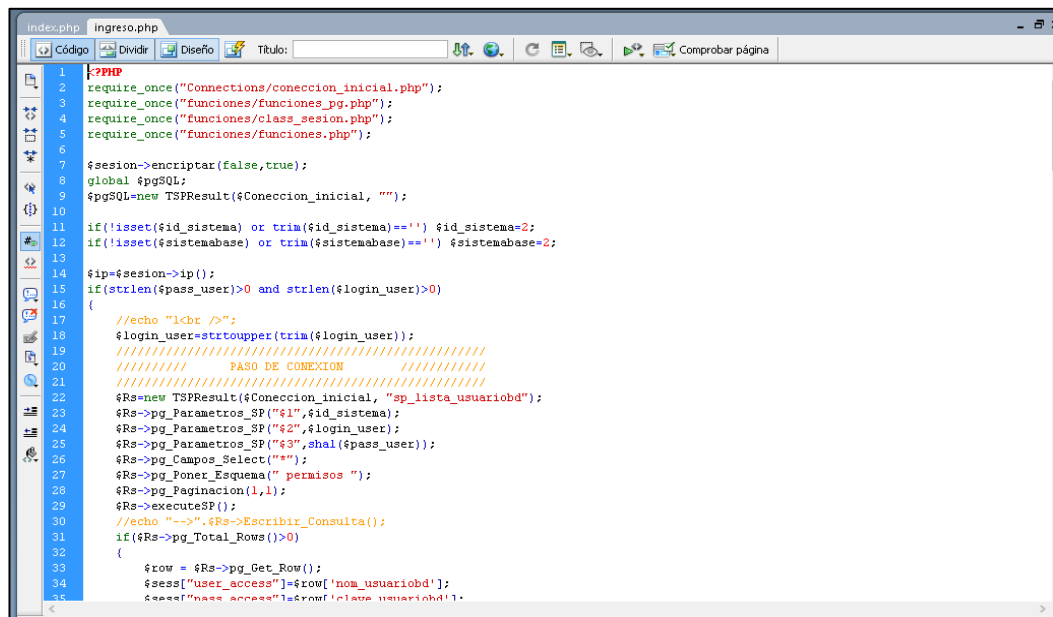
Fuente: Bahit Eugenia POO y MVC en PHP. - 2011. Pág. 56.

En el archivo `aspectos.xml` se estructuran los puntos de unión donde se ejecutarán los avisos (advices) pudiéndose ejecutar antes, después o durante, puntos de corte donde se establece las condiciones para que entren en acción a los puntos de unión.

Durante el tiempo de ejecución el tejedor captura el nombre de la clase y el nombre de la función y los compara con los puntos de corte y ejecuta los puntos de unión los cuales indican qué aspectos deben realizar su trabajo.

d) Implementación de la capa de aplicación

En esta etapa se desarrolló como primera etapa la capa de aplicación que incluye solo el requerimiento funcional de validación de usuarios, ya que en esta sección se codificaron el registro de usuarios con la inclusión en la base de datos de los usuarios habilitados para el ingreso al sistema.



```

1  <?PHP
2  require_once("Connections/conexion_inicial.php");
3  require_once("funciones/funciones_pg.php");
4  require_once("funciones/class_sesion.php");
5  require_once("funciones/funciones.php");
6
7  $sesion->encriptar(false,true);
8  global $pgSQL;
9  $pgSQL=new TSPResult($Conexion_inicial, "");
10
11 if(!isset($id_sistema) or trim($id_sistema)=='') $id_sistema=2;
12 if(!isset($sistemabase) or trim($sistemabase)=='') $sistemabase=2;
13
14 $ip=$sesion->ip();
15 if(strlen($pass_user)>0 and strlen($login_user)>0)
16 {
17     //echo "<br />";
18     $login_user=strtoupper(trim($login_user));
19     ///////////////////////////////////////////////////
20     //PASO DE CONEXION ///////////////////////////////////////////////////
21     ///////////////////////////////////////////////////
22     $Rs=new TSPResult($Conexion_inicial, "sp_lista_usuariobd");
23     $Rs->pg_Parametros_SP("$1", $id_sistema);
24     $Rs->pg_Parametros_SP("$2", $login_user);
25     $Rs->pg_Parametros_SP("$3", sha1($pass_user));
26     $Rs->pg_Campos_Select("*");
27     $Rs->pg_Poner_Eschema(" permisos ");
28     $Rs->pg_Paginacion(1,1);
29     $Rs->executeSP();
30     //echo "<br />"; $Rs->Escribir_Consulta();
31     if($Rs->pg_Total_Rows()>0)
32     {
33         $row = $Rs->pg_Get_Row();
34         $sess["user_access"]=$row['nom_usuariobd'];
35         $sess["name_access"]=$row['name_usuariobd'];

```

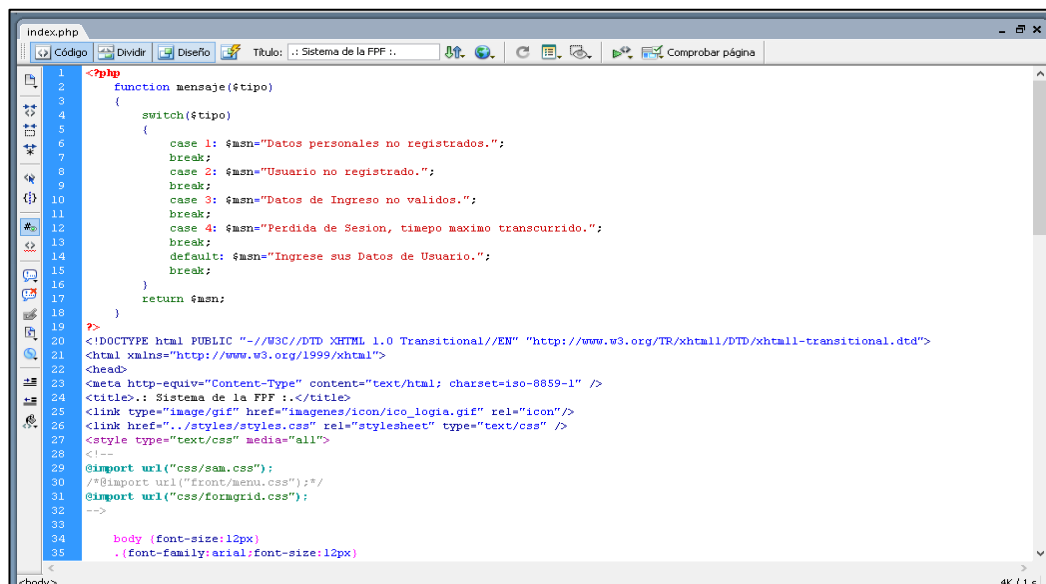
Figura 23: Implementación del módulo de autenticación.

Fuente: Elaboración Propia.

e) Implementación de la capa de dominio

En esta etapa se centró en realizar la unión entre los aspectos y las clases mediante el tejedor con el apoyo del archivo aspectos.xml que contiene los puntos de corte que ejecutaran los puntos de unión.

Aspecto de autenticación: Se codificó para verificar si un usuario está registrado o no en la base de datos, así mismo si no valida o no ingresa dato alguno, será redireccionado al mismo lugar que trató de ingresar o a su sitio de inicio. Se estableció una regla en la cual este aspecto realiza su comprobación en todas las clases (Anexo 01).



```
1 <?php
2 function mensaje($tipo)
3 {
4     switch($tipo)
5     {
6         case 1: $msn="Datos personales no registrados.";
7         break;
8         case 2: $msn="Usuario no registrado.";
9         break;
10        case 3: $msn="Datos de Ingreso no validos.";
11        break;
12        case 4: $msn="Perdida de Sesion, tiempo maximo transcurrido.";
13        break;
14        default: $msn="Ingrese sus Datos de Usuario.";
15        break;
16    }
17    return $msn;
18 }
19
20 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
21 <html xmlns="http://www.w3.org/1999/xhtml">
22 <head>
23 <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
24 <title>.: Sistema de la FPF :.</title>
25 <link type="image/gif" href="imagenes/icon/ico_logia.gif" rel="icon"/>
26 <link href="./styles/styles.css" rel="stylesheet" type="text/css" />
27 <style type="text/css" media="all">
28 <!--
29 @import url("css/saa.css");
30 /*@import url("front/menu.css");*/
31 @import url("css/formgrid.css");
32 -->
33
34 body {font-size:12px}
35 .{font-family:arial;font-size:12px}
36
37 </body>
```

Figura 24: Aspecto de autenticación

Fuente: Elaboración propia.

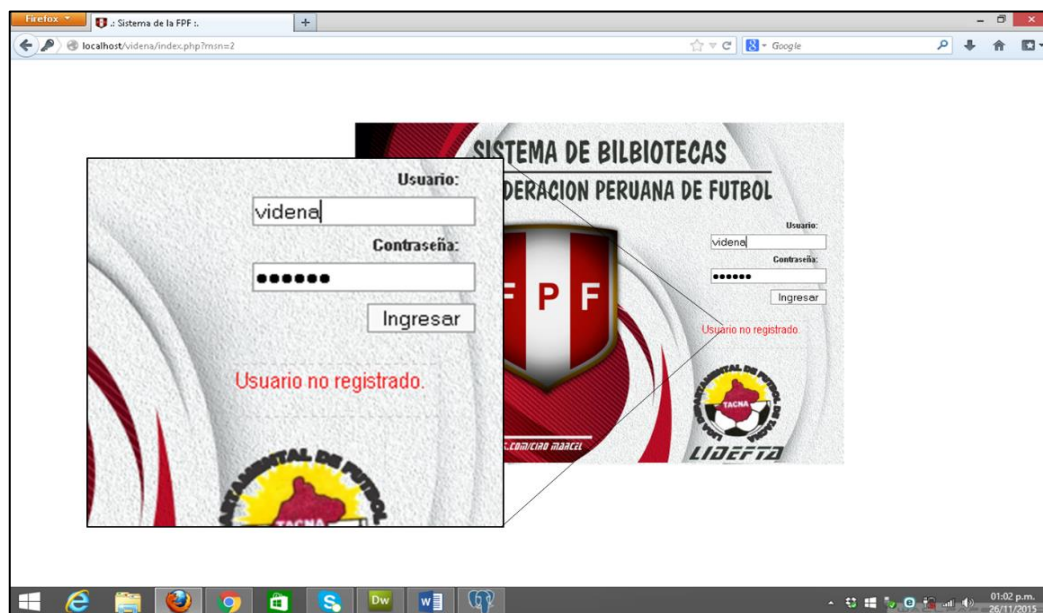


Figura 25: Pantalla de errores

Fuente: Elaboración propia.

CAPÍTULO IV

RESULTADOS

4.1 Resultados obtenidos de la prueba del tamaño (LCD).

Ésta es una medida del tamaño de un programa. Por lo general, cuanto más grande sea el tamaño del código de un componente, más probable será que el componente sea complejo y proclive a errores. La longitud del código podría indicar la proclividad al error en los componentes, un recuento muy alto podría indicar que un tipo o método intenta hacer demasiado trabajo y debe dividirse. También puede indicar que el tipo o método podría ser difícil de mantener.

Tabla 5:

Resultados de cuantificar líneas de código

CUADRO COMPARATIVO CANTIDAD DE LÍNEAS DE CÓDIGO (LCD)		
TIPO DE PROGRAMACIÓN	POA	POO
Módulo Autenticación	216	503

Fuente: *Elaboración propia*

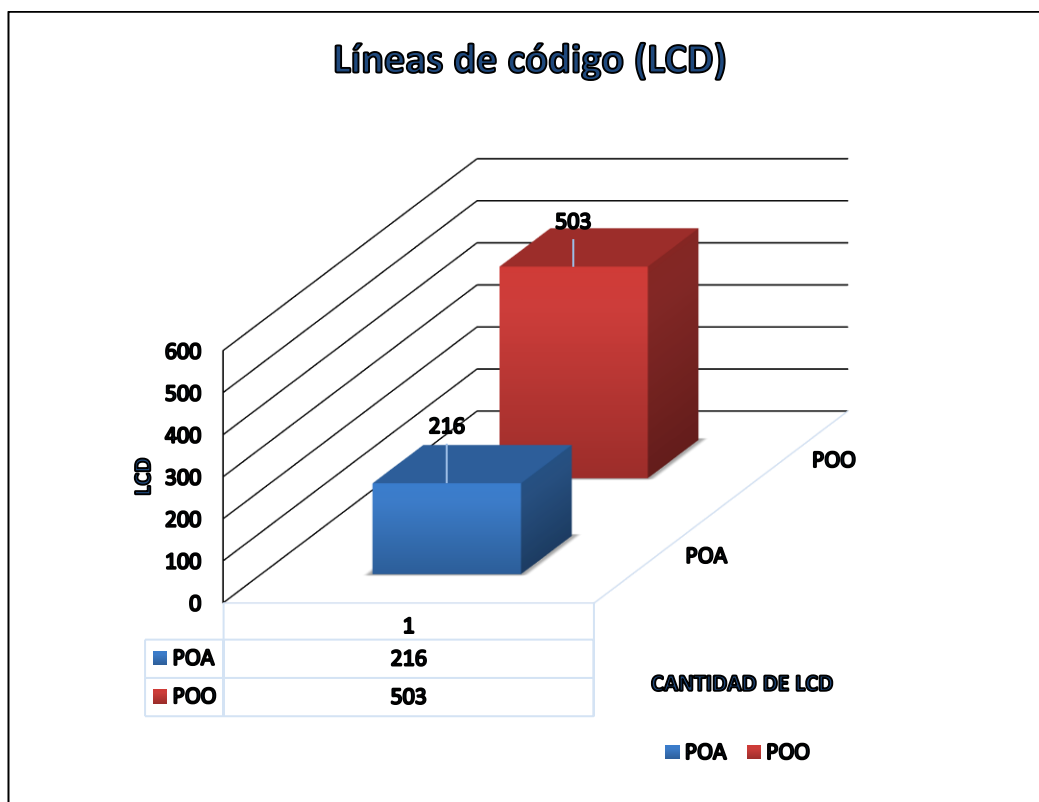


Figura 26: Cuadro comparativo cantidad de LCD

Fuente: Elaboración Propia.

En la Figura 26 del cuadro comparativo de cantidad de líneas de código, se observa que el total de líneas de código con la programación orientada a objetos es de 503 líneas de código y con la programación orientada a aspectos es de 216 líneas de código, esto corresponde al desarrollo de las clases de tipo controlador ya que es donde se realiza la lógica de operaciones del sistema.

Tabla 6:

Tamaño de líneas de código aspectos

PROGRAMACIÓN ORIENTADA A ASPECTOS	LÍNEAS DE CÓDIGO
Autenticación	132
Validación	67
TOTAL	199

Fuente: *Elaboración propia*

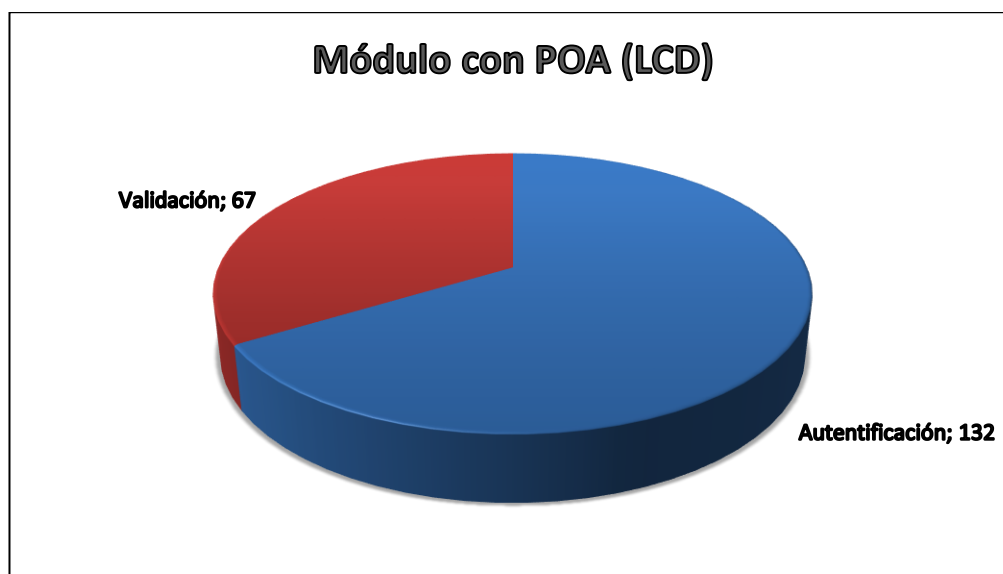


Figura 27: Tamaño de LCD aspectos. Validación

Fuente: Elaboración propia.

Queda por indicar que la cantidad de líneas de código orientadas a objetos corresponden al desarrollo de codificación tanto con la especificación funcional y no funcional, por el contrario, en el caso de la programación orientada a aspectos corresponde solo a especificación funcional, lo cual en un primer análisis indica

que se está logrando una separación de competencias siguiendo el principio de diseño del software.

No obstante, los aspectos también ocupan espacios de líneas de código, y haciendo referencia a la Tabla 06 de tamaño de líneas de código de aspectos se puede observar un total de 199 líneas de código en la cual se encapsulan la especificación no funcional o requerimientos del dominio de la aplicación.

Tabla 7:

Cuadro resumen comparativo

DENOMINACIÓN	COMPARACIÓN	
	POA	POO
CLASES	216	503
ASPECTOS	199	-
TOTAL	415	503

Fuente: *Elaboración propia*

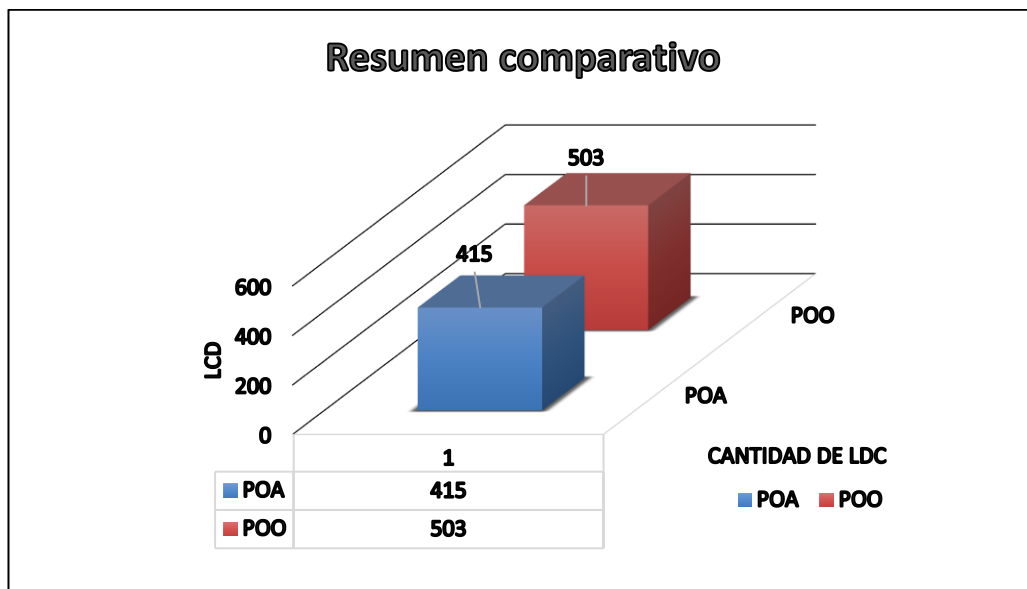


Figura 28: Resumen comparativo

Fuente: Elaboración propia.

Finalmente, en la Tabla 07 el cuadro resumen comparativo entre la programación orientada a aspectos y objetos, en su resultado total da 503 líneas de código realizados con la programación orientada a objetos y 415 líneas de código con la programación orientada a aspectos, lo cual representa un 18 % menos, indicando un impacto positivo aplicando aspectos.

4.2 Resultados obtenidos de la prueba complejidad ciclomática

La complejidad ciclomática calcula el número de puntos en donde se toman decisiones. Se basa en el recuento del número de caminos lógicos individuales contenidos en un programa. Para calcular la complejidad, Thomas McCabe utilizó la teoría y flujo de grafos. Para hallar la complejidad ciclomática, el programa se

representa como un grafo y cada instrucción que contiene, un nodo del grafo. Las posibles vías de ejecución a partir de una instrucción (nodo) se representa en el grafo como aristas. Por lo tanto, proporciona una medición cuantitativa de la complejidad lógica de un programa.

$$\mathbf{V(G) = e - n + 2p} \quad [1]$$

Donde:

$\mathbf{V(G)}$ = Complejidad ciclomática.

$\mathbf{e}$ = Número de aristas del grafo. Una arista conecta dos vértices si una sentencia puede ser ejecutada inmediatamente después de la primera.

$\mathbf{n}$ = Número de nodos del grafo correspondientes a sentencias del programa.

$\mathbf{p}$ = Número de componentes conexos, nodos de salida.

Analizando el Grafo de flujo se obtiene los siguientes resultados, que corresponden a un promedio entre lo realizado con la programación orientada a aspectos y el resultado directo de lo realizado con programación orientada a objetos:

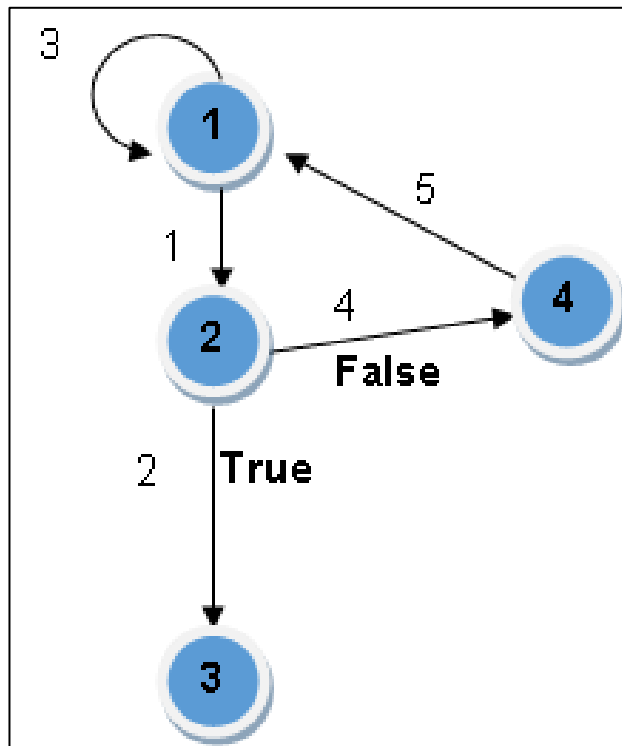


Figura 29: Grafo de flujo del módulo de autenticación

Fuente: Elaboración propia.

En la figura 29 se observa el grafo de flujo para el caso de uso, siendo los nodos que se utilizan en el análisis los siguientes:

Nodo 1: Introducir los datos

Nodo 2: Validar datos

Nodo 3: Ingresar al sistema

Nodo 4: Datos incorrectos

Tabla 8:

Cuadro comparativo de la complejidad ciclomática

Denominación	Programación orientada a aspectos (POA)	Programación orientada a objetos (POO)
Validación	1,00	
Modulo autenticación	3,00	3,00
Promedio	2,00	3,00

Fuente: Elaboración propia

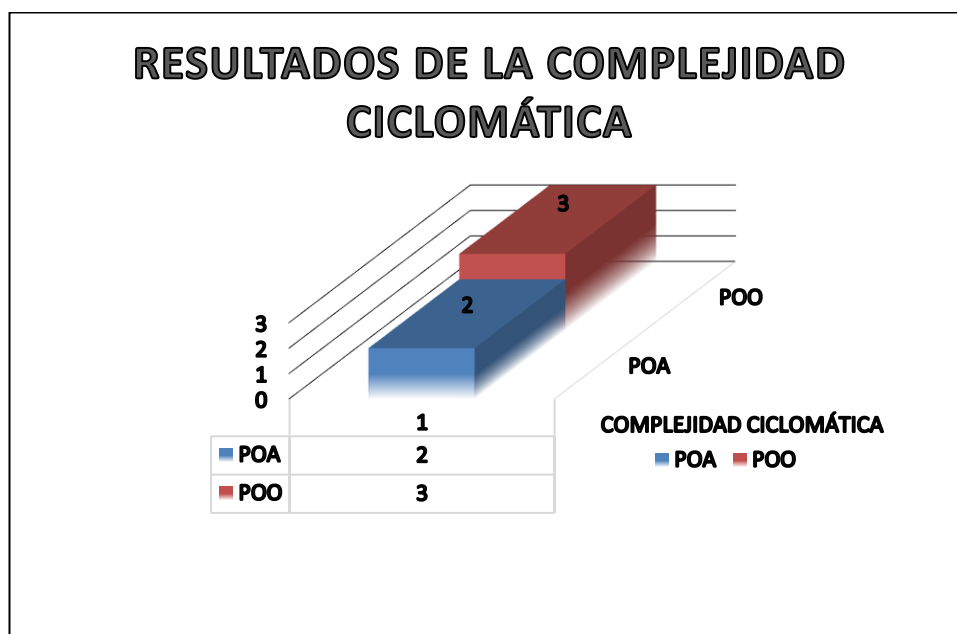


Figura 30: Cuadro comparativo complejidad ciclomática

Fuente: Elaboración propia.

En la Tabla 08 cuadro comparativo, se puede observar la diferencia de la complejidad ciclomática entre lo realizado con la programación orientada a aspectos y lo realizado con la programación orientada a objetos, al realizar una comparación general se obtiene 2,00 como promedio para la POA y 3,00 para POO

en complejidad ciclomática, y al llevarlo a la tabla de escalas se evidencia que para lo realizado con la orientación a objetos tiene una complejidad moderada y con lo realizado con la orientación a aspectos se obtiene un poco de complejidad.

Tabla 9:

Tabla de escalas de la complejidad ciclomática

Escala	Descripción
1 - 2	Poca complejidad
3 - 4	Complejidad moderada
5 - 6	Complejidad alta
6+	Complejidad muy alta

Fuente: PRESSMAN, R. *Ingeniería de Software*, 2005. Pág.210

La Tabla 09 refleja la escala de complejidad de cada clase, se demuestra el efecto que tiene la orientación a aspectos en la complejidad del sistema donde hace que baje notoriamente comparada con la orientada a objetos. Se puede decir que entre más complejidad tenga una rutina o un software, más se dificulta su mantenibilidad, la probabilidad de tener errores será mucho más alta e incluso añadir nuevas funcionalidades se volverá más complicado.

La conclusión es que un software resulta más eficiente cuanto menor sea su complejidad, con la orientación a aspectos se logra que a través del diseño usando aspectos la complejidad del software disminuya en comparación con la orientación a objetos.

4.3 Resultados obtenidos de la eficiencia computacional (uso de memoria RAM)

En las Tablas 10 y 11 especificamos las características de los recursos computacionales, tanto hardware como software, que se utilizaron para las pruebas en ambos casos de estudio.

Tabla 10:

Hardware para las pruebas de uso de memoria

CPU (Hardware)	
Componente	Descripción
Placa Madre	Intel DH55TC
Procesador	Intel ® Core™ i5 3230 M @ 2.6GHZ
Memoria RAM	kingston 4 GB DDR3
Disco Duro	Toshiba 500 GB 7200 RPM SATA

Fuente: *Elaboración propia*

Tabla 11:

Software para las pruebas de uso de memoria

Sistemas (Software)	
Aplicación	Descripción
Sistema Operativo	Windows 8.1 Pro
Medidor de Memoria	FreeMeter
SPSS	V20 en español
XAMPP	V 5.5.30
PostgreSQL	9.1.19

Fuente: *Elaboración propia*

Para este caso se considera la cantidad de tiempo de ejecución y el consumo de memoria que se logra con cada caso de estudio. Se obtuvieron los siguientes resultados:

Tabla 12:*Resultados de uso de memoria en diferentes test del módulo*

Nro.	Programación orientada a aspectos		Programación orientada a objetos	
	Segundos	MB	Segundos	MB
Test 1	0,141	2,22MB	0,172	2,18MB
Test 2	0,141	2,22MB	0,141	2,18MB
Test 3	0,141	2,22MB	0,141	2,18MB
Test 4	0,156	2,22MB	0,152	2,18MB
Test 5	0,141	2,22MB	0,141	2,18MB
Test 6	0,156	2,22MB	0,156	2,18MB
Test 7	0,141	2,22MB	0,156	2,18MB
Test 8	0,141	2,22MB	0,141	2,18MB
Test 9	0,141	2,22MB	0,078	2,18MB
Test 10	0,141	2,22MB	0,141	2,18MB
Test 11	0,141	2,22MB	0,125	2,18MB
Test 12	0,141	2,22MB	0,141	2,18MB
Test 13	0,156	2,22MB	0,156	2,18MB
Test 14	0,156	2,22MB	0,156	2,18MB
Test 15	0,141	2,22MB	0,172	2,18MB
	0,141	2,22MB	0,147	2,18MB

Fuente: *Elaboración propia*

En la Tabla 12 se muestra es el promedio de tiempo de ejecución de quince pruebas (15 test) de nuestro modulo, lo cual da un total de 0,141 segundos para lo desarrollado con la programación orientada a aspectos (POA), y un total de 0,147

con la programación orientada a objetos (POO), esto indica que si se diseña un software orientado a aspectos la ejecución de esta no incidirá de manera negativa.

En los promedios de consumo de memoria da un consumo de 2,22MB en el software orientado a aspectos y 2,18MB para el software orientado a objetos, este consumo de memoria extra se debe al tejido dinámico que es el tejido que se adoptó para este caso de estudio. Para estos resultados, se hizo la prueba de hipótesis, con lo que se demostró estadísticamente la veracidad del presente trabajo.

Definición de la hipótesis nula y alterna

H_1 = **Existe** una diferencia significativa entre la media de los resultados obtenidos del uso de la memoria en diferentes compilaciones con la POO y la media de los resultados obtenidos del uso de la memoria en diferentes compilaciones con la POA.

H_0 = **NO Existe** una diferencia significativa entre la media de los resultados obtenidos del uso de la memoria en diferentes compilaciones con POO y la media de los resultados obtenidos del uso de la memoria en diferentes compilaciones con POA.

Determinar el nivel de significancia ALFA

Se define el nivel alfa en 5 % = 0,05

Elección de la prueba estadística

Al tratarse de un estudio transversal de dos grupos, cuyas pruebas paramétricas de la variable aleatoria es numérica, se eligió a la prueba estadística es T-Student para muestras independientes, para la prueba de hipótesis.

Calcular P-VALOR

Al calcular el P-Valor, se corroboró que la variable aleatoria (resultado de los test) en ambos grupos se distribuye normalmente y también se debe probar el supuesto de igualdad de varianza, al trasladar los resultados al SPSS se obtiene que: (Anexo 02).

Tabla 13:

Pruebas de normalidad con SPSS

Pruebas de normalidad							
Programación		Kolmogorov-Smirnova			Shapiro-Wilk		
		Estadístico	gl	Sig.	Estadístico	gl	Sig.
Segundos	POO	,284	15	,200	,716	32	,156
	POA	,417	15	,200	,629	32	,257

a. Corrección de significación de Lilliefors

Fuente: *Elaboración propia.*

Viendo la Tabla 13 se puede decir que, al tener la muestra menor de 30, corresponde usar los resultados de grado de significancia de Shapiro Wilk. En ambos casos se observa que los resultados para la normalidad son mayores que nuestro α , entonces:

Tabla 14:*Prueba de normalidad*

Normalidad		
P-valor (POO) = 0,156	>	$\alpha = 0,05$
P-valor (POA) = 0,257	>	$\alpha = 0,05$

Fuente: *Elaboración Propia.*

Como vemos en la Tabla 14, el P-valor $\geq \alpha$ por ende se puede concluir que los datos de uso de memoria provienen de una distribución normal. Siguiendo con las pruebas, se debe corroborar el supuesto de igualdad de varianzas, para lo cual se usa la prueba de Levene, obteniendo los siguientes resultados:

Tabla 15:*Prueba de igualdad de varianzas*

Igualdad de Varianza		
P-valor = 0,699	>	$\alpha = 0,05$

Fuente: *Elaboración Propia.*

Viendo la Tabla 15, el P-valor $\geq \alpha$ se puede concluir que las varianzas son iguales. De esta forma se corrobora que se cumple con los dos supuestos de normalidad e igualdad de varianzas. Luego, no es necesario usar más el software SPSS, ya que se puede rescatar el valor de P-valor de la prueba t-Student para la igualdad de medias.

Tabla 16:

Prueba T-Student

Prueba T-Student		
P-valor = 0,661	>	$\alpha = 0,05$

Fuente: *Elaboración propia.*

En la Tabla 16 vemos que el P-Valor $> \alpha$, por lo tanto se acepta la H_0 , y se puede concluir que: **NO Existe** una diferencia significativa entre la media de los resultados obtenidos del uso de la memoria en diferentes compilaciones con programación orientada a objetos (POO) y la media de los resultados obtenidos del uso de la memoria en diferentes compilaciones con programación orientada a aspectos (POA).

DISCUSIONES

Del análisis de resultados del presente estudio se puede afirmar que la mayor abstracción, que una clara modularidad y la separación de competencias de la programación orientada a aspectos incidió positivamente en el diseño e implementación del módulo de autenticación en desarrollo.

Si comparamos los resultados del trabajo de investigación de Arismendi Almonacid, Rodrigo y Venegas Hernández (2008), cuyo resultado favorable al cuantificar las líneas de código con programación orientada a aspectos (POA) es del 15,79 % y los resultados obtenidos de esta investigación cuyo resultado favorable es del 18 % se puede ver que ambos están dentro del rango de satisfactorio, debiendo aclarar que en el primer trabajo si bien es menor el porcentaje de abstracción de la LCD, por la complejidad del sistema dicho resultado es muy satisfactorio para su caso de estudio y en el trabajo de investigación actual se desarrolla un módulo de autenticación, el resultado del presente estudio está dentro de satisfactorio.

Ahora si bien en el trabajo de tesis de Jimenez Murillo, Karen Rocío (2012), también se implementa un módulo de autenticación aunque en un sentido diferente a este caso de estudio, es similar puesto que su módulo de contraseñas tiene la misma complejidad ciclomática que este software.

Finalmente, en el informe de tesis de Cabrera Hormazábal, Carlos Sebastián (2010), habla sobre una comparación y reducción del uso de memoria de un producto software pero habla en general, es decir evaluó al software de manera superficial al hacer pruebas con un software que mide el uso de la memoria de todo el ordenador en uso, por lo que después de desarrollar su investigación llegaron a una diferencia significativa en las diferentes compilaciones que le hicieron al ordenador siendo sus resultados satisfactorios en dicha investigación. En el presente caso de estudio se demostró estadísticamente que no existe una diferencia significativa entre la media de los resultados obtenidos del uso de la memoria en diferentes compilaciones que se le hizo al módulo mismo.

Programación orientada a aspectos (Promedios d tiempo y uso de memoria)		Programación orientada a objetos (Promedios d tiempo y uso de memoria)	
Segundos	MB	Segundos	MB
0,141	2,22MB	0,147	2,18MB

CONCLUSIONES

Se estableció que al comparar la programación orientada a aspectos (POA) y la programación orientada a objetos (POO) en la implementación de un módulo de autenticación, la POA incidió positivamente en el diseño e implementación porque permitió una clara modularidad y separación de competencias, debido a que se identificó desde tempranos inicios del desarrollo del software en comparación con la programación orientada a objetos. Todo esto en base a las tres características que se definió como eficiencia del sistema. Se cuantificó las tres características que se definió como eficiencia del sistema (complejidad ciclomática, cantidad de líneas de código y uso de memoria), con la finalidad de hacer el análisis comparativo de ambos paradigmas de la programación.

Se evaluó el módulo de autenticación implementado con POA con los tres indicadores de eficiencia del sistema, cuantificando los resultados en tablas y gráficas a lo largo de la presente investigación.

Se evaluó el módulo de autenticación implementado con POO con los tres indicadores de eficiencia del sistema, cuantificando los resultados en tablas y gráficas a lo largo del presente trabajo de investigación. Se determinó que usando la programación orientada a objetos en el diseño de un software se favorece el modularidad, pero al no poder brindar soporte para funcionalidades comunes del sistema hace que la complejidad del software se incremente. Es decir, la eficacia del módulo en estudio, con la POO es desfavorable en función a los indicadores.

Se comprobó que, al cuantificar las líneas de código y uso de memoria y complejidad ciclomática con la programación orientada a aspectos (POA) da resultados favorables de un 18 % menos que la programación orientada a objetos (POO). Además, con la POA se logra que a través del diseño usando aspectos la complejidad ciclomática del software disminuya en comparación con la POO. Se demostró estadísticamente que no existe una diferencia significativa entre la media de los resultados obtenidos del uso de la memoria en diferentes compilaciones con programación orientada a objetos y la media de los resultados obtenidos del uso de la memoria en diferentes compilaciones con programación orientada a aspectos.

RECOMENDACIONES

Se plantea la elaboración de métricas orientadas a aspectos, como son la incidencia de los puntos de unión y corte sobre el código funcional, manejando una tabla de escalas que ayuden a verificar cuando se está incurriendo en un uso excesivo a nivel de codificación.

Se plantea que se elabore una metodología orientada a aspectos que realice la separación de requerimientos desde la especificación hasta la implantación del software, brindando en todas sus etapas procedimientos que garanticen la calidad, así como también brindar un lenguaje de modelado simple pero efectivo.

Se plantea realizar estudios acerca del efecto de aplicar el enfoque de la programación orientada a aspectos en múltiples equipos de desarrollo de software, así como su influencia en las actividades colaborativas entre los equipos de trabajo. Además de realizar estudios de técnicas para identificar aspectos tempranos, o la elaboración de sistemas automatizados que ayuden en esta tarea, para de esta forma ayudar a un buen diseño desde el principio.

REFERENCIAS BIBLIOGRÁFICAS

- Acosta Rodríguez, Lester. (2012). *Caracterización de la programación orientada a aspectos (POA)*: Editorial Académica Española, - pág. 76-268.
- Arismendi Almonacid, Rodrigo y Venegas Hernández, Susan. (2008). *Programación Orientada a Aspectos*. - Concepción.
- Asteasuain, Fernando y Contreras, Bernardo E. (2002). *Programación orientada a aspectos - Análisis del paradigma*. - Buenos aires.
- Bahit, Eugenia. (2011). *POO y MVC en PHP*.
- Barraza Velarde, Olinda. (2006). *Introducción a la programación orientada a objetos*. - México: Pearson Educación.
- Booch, Grady. (2009). *Análisis y diseño de orientado a objetos*.
- Boticki I. Katic, M. y Martin S. (2013). Exploring the Educational Benefits of Introducing Aspect-Oriented Programming Into a Programming Course.
- Brichau Johan y D'Hondt. (2005). The Aspect-Oriented Software Development.
- Cabrera Hormazábal, Carlos Sebastián. (2010). *Control de reentrenada de aspectos en aspectj*. Santiago de Chile.

Casas, Sandra. (2006). Conflictos entre Aspectos en Etapas del Desarrollo de Software.

Cugola Gianpaolo, Ghezzi Carlo y Monga Mattia. (1999). Coding Different Design Paradigms for Distributed Applications with Aspect-Oriented Programming - Milano - Italy.

Díaz Carrasco, Sergio. (2007). *Metodología de la investigación científica*. Editorial San Marcos.

David A. William Watt. (2010). *Findlay Programming Language Design Concepts - England: Second Edition*, - pág. 143-188.

D'Ary, L., Ch. Jacobs y A. Razavieh (1982). *Introducción a la Investigación Pedagógica*, México: Ed. Interamericana, Segunda edición.

Echevarría Rodríguez Roberto, Sanchez Sosa Encarna y Ramos Prieto Álvaro. (2004). *Programación orientada a objetos*.

Elhaibe, Rodrigo. (2006). H Metodologías Ágiles y Programación orientada a Aspectos Estudio de una posible integración - Buenos aires: [s.n.].

Elmasri Ramez y Navathe Shamkant B. (1992). *Sistemas de bases de datos*. - México: Addison-Wesley Iberoamericana, Segunda edición.

Erdody, Diego. (2006). M.S. *Un framework para la visualización de patrones de diseño distribuidos y concurrentes implementados con programación orientada a aspectos.*

Gonzales Apaza, Renán Darío. (2014). *Efecto de la programación orientada a aspectos y objetos en el diseño de un software* - Universidad Nacional de Moquegua

Guzman Martin, Tojin Karen, Hurtarte Héctor, Sánchez Kevin. (2007). *Metodologías para análisis y diseño orientado a objetos y MDA* [Publicación periódica] // Scribd. -. - págs. 3-13.

<https://weblogs.java.net/blog/potty/archive/2014/01/20/introduction-object-oriented-programming-oop-part-1>

Jacobson, Ivar y Ng Pan. (2004). *Wei Aspect-Oriented Software Development with Use Cases: Addison Wesley Professional.*

Jimenez Murillo, Karen Rocío. (2012). *Propuesta de metodología y estándares para la administración de proyectos en las pequeñas y medianas empresas de software con base en los estándares del pmi.* - San José, Costa Rica.

Joyanes Aguilar, Luis. (1996). *Programación orientada a objetos.* – Madrid.

Kendall Kendall. (2005). *Análisis y diseño de sistemas.* - México. : Pearson Education.

Keuler Thorsten. (2011). *An Aspect-oriented approach for improving architecture design efficiency*. - Kaiserslautern: Vol.33. - pág. 43-53.

Laddad Ramnivas. (2009). *AspectJ in Action*.

Martinez Abdel Java.net [En línea]. (20 de 01 de 2015. - 09 de 05 de 2015). *The Source for Java Technology Collaboration*.

Ortín Ibáñez, María José y García Molina, Jesús. (2000). Modelado basado en roles con UML.

Quispe Platon, Griseldo. (2008). *Palomino Investigación cualitativa y cuantitativa en educación*. - Puno: FCEDUC. - Tercera edición.

Reina Quintero, Antonia Maria. (2003). *Visión general de la programación Rational Software Rational Unified Process*.

Reyes González Yunia, González Bravo Lisbet M. y Ruiz Constanten Yadira. (2008). Estudio comparativo sobre las principales metodologías pesadas y orientadas a objeto en el desarrollo de software.

Sampieri Hernández Roberto, Fernández Collado Carlos y Lucio Pilar Baptista. (2006). *Metodología de la investigación*. - Itzapalapa - México: pág. 78-312.

Sánchez, Pablo. (2013). Mapping Aspects from Requirements to Architecture.

Silvia Tabares Marta, Alf  rez Salinas Germ  n y Alf  rez Salinas Edward. (2008).

El desarrollo de software orientado a aspectos: un caso pr  ctico para un sistema de ayuda en linea.

Sofia Brito Isabel, Moreira Ana, Almeida Ribeiro Rita, Ara  jo Jo  o. (2013).

Handling conflicts in aspect-oriented requirements engineering
[Publicaci  n peri  dica]. - Portugal: [s.n.].

Sommerville, Ian. (2008). *Ingenier  a de software.*: Pearson.

Tarr P. Ossher H. Harrison. (1999). W. y S.M. Sutton N Degrees of separation:

Multi- Dimensional Separation of Concerns.

Vidal Cristian, Schmal Rodolfo, Rivero Sabino y Villarroel Rodolfo. (2012).

Extensi  n del Diagrama de Secuencias UML (Lenguaje de Modelado Unificado) para el Modelado Orientado a Aspectos - Chile.

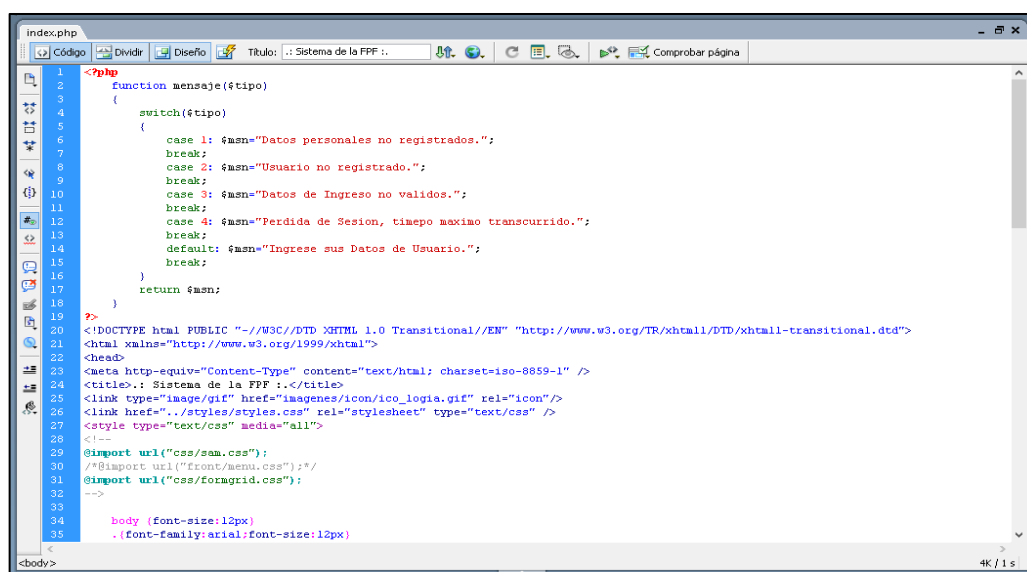
Welker K.D y Oman P.W. (1995). Software Maintainability Metrics Models in

Practice.

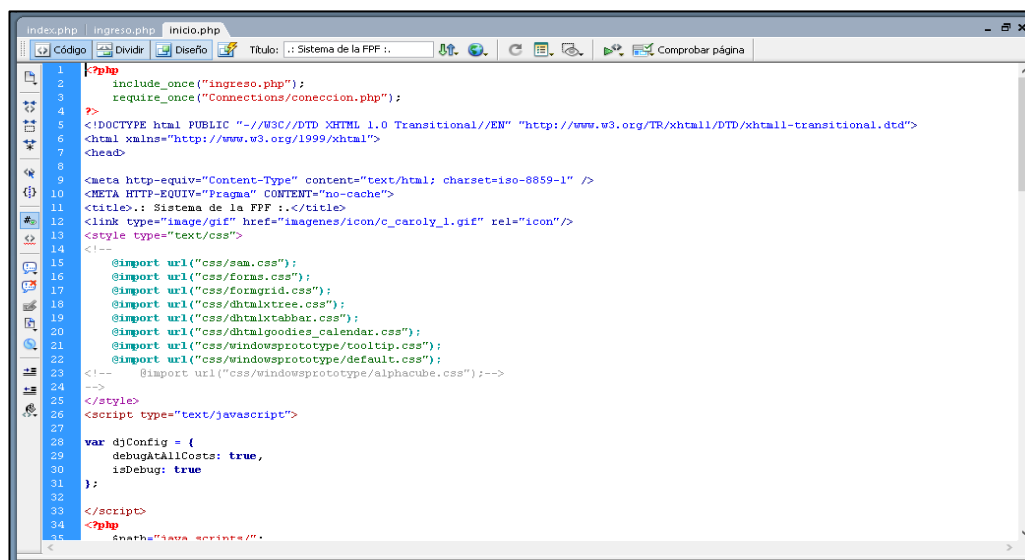
Xerox The Home of AspectC++ [En l  nea]. (08 de 04 de 2014. - 12 de 05 de 2014).

ANEXOS

ANEXO 01: Interfaz gráfica del módulo de autenticación y mensajes de error.



```
1 <?php
2 function mensaje($tipo)
3 {
4     switch($tipo)
5     {
6         case 1: $msn="Datos personales no registrados.";
7             break;
8         case 2: $msn="Usuario no registrado.";
9             break;
10        case 3: $msn="Datos de Ingreso no validos.";
11            break;
12        case 4: $msn="Perdida de Sesion, tiempo maximo transcurrido.";
13            break;
14        default: $msn="Ingrese sus Datos de Usuario.";
15            break;
16    }
17    return $msn;
18 }
19
20 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
21 <html xmlns="http://www.w3.org/1999/xhtml">
22 <head>
23 <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
24 <title>.: Sistema de la FFF :.</title>
25 <link type="image/gif" href="imagenes/icon/ico_logia.gif" rel="icon"/>
26 <link href="../styles/styles.css" rel="stylesheet" type="text/css" />
27 <style type="text/css" media="all">
28 <!--
29 @import url("../css/sam.css");
30 /*@import url("../front/Menu.css");*/
31 @import url("../css/formgrid.css");
32 -->
33
34 body {font-size:12px}
35 .{font-family:arial;font-size:12px}
36
37 </body>
```



```
1 <?php
2 include_once("ingreso.php");
3 require_once("Connections/conexion.php");
4
5 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
6 <html xmlns="http://www.w3.org/1999/xhtml">
7 <head>
8
9 <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
10 <META HTTP-EQUIV="Pragma" CONTENT="no-cache">
11 <title>.: Sistema de la FFF :.</title>
12 <link type="image/gif" href="imagenes/icon/c_caroly_1.gif" rel="icon"/>
13 <style type="text/css">
14 <!--
15 @import url("../css/sam.css");
16 @import url("../css/forms.css");
17 @import url("../css/formgrid.css");
18 @import url("../css/dhtmltree.css");
19 @import url("../css/dhtmlxtable.css");
20 @import url("../css/dhtmlgoodies_calendar.css");
21 @import url("../css/windowsprototype/tooltip.css");
22 @import url("../css/windowsprototype/default.css");
23 @import url("../css/windowsprototype/alphacube.css");-->
24 -->
25 </style>
26 <script type="text/javascript">
27
28 var djConfig = {
29     debugAllCosts: true,
30     isDebug: true
31 };
32
33 </script>
34
35 <?php
36 $vartha="aaa erprints";
```

```

index.php | ingreso.php | inicio.php | admincon.js
Código | Dividir | Diseño | Título:
157 }
158 }
159 else
160 if(strlen($sess["user_access"])<=0)
161 {
162     if(isset($pass_user) and isset($login_user))
163     {
164         guarda_registro (5, $id_sistema, $sess["id_persona"], $ip, 'INGRESO NO VALIDO - USUARIO: '.$login_user.' - PASSWORD: '.$pass_user, '0', $Conexion_inicial);
165         $msn=3;
166     }
167     else
168     {
169         if(isset($Conexion) and $Conexion<>"")
170         {
171             guarda_registro (6, $id_sistema, $sess["id_persona"], $ip, 'PERDIDA DE SESION', '0', $Conexion);
172         }
173         $msn=4;
174     }
175 }
176 $path_ini="index.php";
177 header("Location: ".$path_ini."?msn=$msn");
178 exit;
179 }
180 else
181 {
182     require_once("Connections/conexion.php");
183     //if($salir!=1)
184     // if ($sess["pass_user"]=="123") echo "<script>
185     window.open('panel/ventana_contra.php','','top=350,left=400,width=370,height=155,menubar=no,toolbar=no,scrollbars=si,resizable=yes,status=no') </script>";
186 }
187
7K / 1 s

```

```

index.php | ingreso.php | inicio.php | admincon.js | almacen.js | maestro.js | principal_ajax.js
Código | Dividir | Diseño | Título:
1 function objetoAjax(){
2
3 var xmlhttp=false;
4
5 try {
6
7
8 xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
9
10 } catch (e) {
11
12 try {
13
14 xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
15
16 } catch (E) {
17
18 xmlhttp = false;
19
20 }
21
22 }
23
24 if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
25 xmlhttp = new XMLHttpRequest();
26 }
27 return xmlhttp;
28 }
29
30
31 function enviarDatosBusqueda(){
32 divResultado = document.getElementById('resultado');
33 divEstado = document.getElementById('estado');
34 //valores de los inputs
35 type = document.sol_form.sel_nec.value;
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
25
```





ANEXO 02: Realización de las pruebas estadísticas

IBM SPSS Statistics Editor de datos

	Nombre	Tipo	Anchura	Decimales	Etiqueta	Valores	Perdidos	Columnas	Alineación	Medida	Rol
1	Numero	Numérico	8	0	Decimales	Ninguno	Ninguno	8	Derecha	Escala	Entrada
2	Programacion	Numérico	8	2		{1.00, POO}...	Ninguno	8	Derecha	Nominal	Entrada
3	Segundos	Coma	8	4		Ninguno	Ninguno	8	Derecha	Escala	Entrada
4											
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											
22											
23											
24											
25											

Vista de datos Vista de variables

IBM SPSS Statistics Processor está listo Unicode ON

IBM SPSS Statistics Editor de datos

Archivo Editar Ver Datos Transformar Analizar Marketing directo Gráficos Utilidades Ventana Ayuda

64: Programacion

Visible: 3 de 3 variables

	Numero	Programacion	Segundos	var	var	var	var	var	var	var	var	var	var	var	var	var
1	1	POO	.1719													
2	2	POO	.1406													
3	3	POO	.1406													
4	4	POO	.1563													
5	5	POO	.1406													
6	6	POO	.1563													
7	7	POO	.1563													
8	8	POO	.1406													
9	9	POO	.0781													
10	10	POO	.1406													
11	11	POO	.1250													
12	12	POO	.1406													
13	13	POO	.1563													
14	14	POO	.1563													
15	15	POO	.1719													
16	16	POO	.1563													
17	17	POO	.1563													
18	18	POO	.1406													
19	19	POO	.1563													
20	20	POO	.0938													
21	21	POO	.1563													
22	22	POO	.1563													
23	23	POO	.1406													

Vista de datos Vista de variables

IBM SPSS Statistics Processor está listo Unicode ON

IBM SPSS Statistics Editor de datos

Archivo Editar Ver Datos Transformar Analizar Marketing directo Gráficos Utilidades Ventana Ayuda

64: Programacion

Visible: 3 de 3 variables

	Numero	Programaci...	Segur...	var	var	var	var	var	var	var	var	var	var	var	var	var
1	1	POO														
2	2	POO														
3	3	POO														
4	4	POO														
5	5	POO														
6	6	POO														
7	7	POO														
8	8	POO														
9	9	POO														
10	10	POO														
11	11	POO														
12	12	POO														
13	13	POO														
14	14	POO														
15	15	POO														
16	16	POO														
17	17	POO														
18	18	POO														
19	19	POO														
20	20	POO														
21	21	POO														
22	22	POO														
23	23	POO														

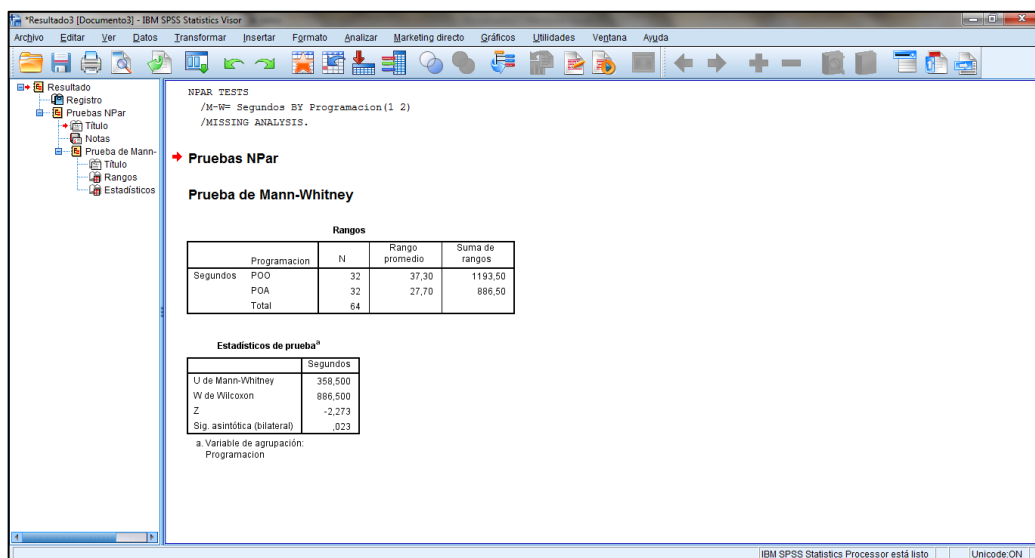
Vista de datos Vista de variables

IBM SPSS Statistics Processor está listo Unicode ON

Analizar

- Informes
- Estadísticos descriptivos
- Tablas personalizadas
- Comparar medias
- Modelo lineal general
- Modelos lineales generalizados
- Modelos mixtos
- Correlaciones
- Regresión
- Lglineal
- Redes neuronales
- Clasificar
- Regulación de dimensiones
- Escala
- Pruebas no paramétricas
- Predicciones
- Supervivencia
- Respuesta múltiple
- Análisis de valores perdidos...
- Imputación múltiple
- Muestras complejas
- Simulación...
- Control de calidad
- Cuaja COR...
- Modelado espacial y temporal...

- Muestras relacionadas...
- Muestras independientes...
- Una muestra...
- Cuadros de diálogo antiguos
- Chi-cuadrado...
- Binomial...
- Rachas...
- K-S de 1 muestra...
- 2 muestras independientes...
- 5 muestras independientes...
- 2 muestras relacionadas...
- K muestras relacionadas...



ANEXO 03: Instrumento utilizado para la recolección de datos. Ficha de registro de los indicadores

INDICADOR 01.

```

170 },
171 determineIfGameIsFinished: function()
172 {
173     if (!this.findAnyAdjacentPair()) {
174         Sound.play('audio/17213_flatfly');
175         this.container.select('div').each(
176             function() {
177                 new Effect.Pulsate(square, {duration: 1000});
178             }
179         );
180     }
181     findAnyAdjacentPair: function() {
182         return this.container.select('div')
183             .findDirectNeighbors().bind(this);
184     }
185 }

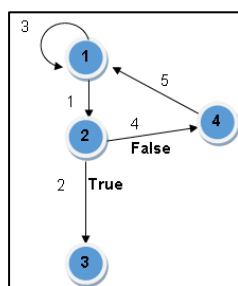
```

CUADRO COMPARATIVO CANTIDAD DE LÍNEAS DE CÓDIGO (LCD)

TIPO DE	POA	POO
Módulo Autenticación	216	503

Fuente: Elaboración Propia

INDICADOR 02



$$V(G) = 5 - 4 + 2(1)$$

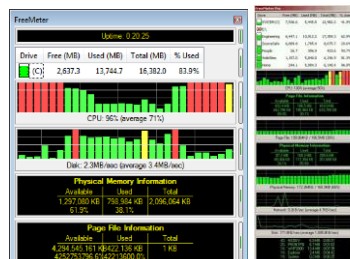
$$V(G) = 3$$

CUADRO COMPARATIVO DE LA COMPLEJIDAD CICLOMÁTICA

Denominación	Programación orientada a aspectos	Programación orientada a objetos
Validación	1,00	
Modulo Autenticación	3,00	3,00
Promedio	2,00	3,00

Fuente: Elaboración Propia

INDICADOR 03



RESULTADOS DE USO DE MEMORIA TESTEANDO EL MÓDULO

Nro.	Programación orientada a aspectos		Programación orientada a objetos	
	Segundos	MB	Segundos	MB
Test 1	0,141	2,22MB	0,172	2,18MB
Test 2	0,141	2,22MB	0,141	2,18MB
Test 3	0,141	2,22MB	0,141	2,18MB
Test 4	0,156	2,22MB	0,152	2,18MB
Test 5	0,141	2,22MB	0,141	2,18MB
Test 6	0,156	2,22MB	0,156	2,18MB
Test 7	0,141	2,22MB	0,156	2,18MB
Test 8	0,141	2,22MB	0,141	2,18MB
Test 9	0,141	2,22MB	0,078	2,18MB
Test 10	0,141	2,22MB	0,141	2,18MB
Test 11	0,141	2,22MB	0,125	2,18MB
Test 12	0,141	2,22MB	0,141	2,18MB
Test 13	0,156	2,22MB	0,156	2,18MB
Test 14	0,156	2,22MB	0,156	2,18MB
Test 15	0,141	2,22MB	0,172	2,18MB
	0,141	2,22MB	0,147	2,18MB

Fuente: Elaboración Propia

ANEXO 04: Matriz de consistencia.

TÍTULO: ANÁLISIS COMPARATIVO DE LOS PARADIGMAS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS Y ASPECTOS EN LA CONSTRUCCIÓN DEL MÓDULO DE AUTENTICACIÓN DEL SOFTWARE DE LA BIBLIOTECA DE LA LIGA DEPARTAMENTAL DE FÚTBOL DE TACNA - 2015			
Problema	Objetivos	Hipótesis	Variables e indicadores
Problema general: ¿Mediante un análisis comparativo se puede determinar la eficiencia de la programación orientada a aspectos y la programación orientada a objetos en la construcción de un módulo de autenticación?	Objetivo General: Comparar la Programación Orientada a Objetos y la Programación Orientada a Aspectos teniendo como base la eficiencia en el desarrollo del Módulo de Autenticación del software de la Biblioteca de la Liga Departamental de Futbol de Tacna.	Hipótesis General: En la construcción de un módulo de autenticación el diseño con la programación orientada a aspectos la efectividad es mejor que la orientada a objetos en el software de la Biblioteca de la Liga departamental de Futbol de Tacna.	Variables e indicadores • V₁: Módulo de autenticación con Programación orientada a aspectos. • V₂: Módulo de autenticación con Programación orientada a objetos.
Problemas Específicos	Objetivos Específicos:	Sub Hipótesis:	Indicadores
a) ¿Se podrá establecer y cuantificar la cantidad de líneas de código, complejidad ciclomática y uso de memoria como características que definen la eficiencia en el desarrollo con Programación Orientada a Aspectos del Módulo de Autenticación del software de la Biblioteca de la Liga Departamental de Futbol de Tacna? b) ¿Se podrá establecer y cuantificar la cantidad de líneas de código, complejidad ciclomática y uso de memoria como características que definen la eficiencia en el desarrollo con Programación Orientada a Objetos del Módulo de Autenticación del software de la Biblioteca de la Liga Departamental de Futbol de Tacna? c) ¿Se podrá realizar un análisis comparativo veraz de los resultados obtenidos de cuantificar nuestros índices de eficiencia?	a) Cuantificar y evaluar la cantidad de líneas de código, complejidad ciclomática y uso de memoria con Programación Orientada a Aspectos en el Módulo de Autenticación del software de la Biblioteca de la Liga Departamental de Futbol de Tacna. b) Cuantificar y evaluar la cantidad de líneas de código, complejidad ciclomática y uso de memoria con Programación Orientada a Objetos en el Módulo de Autenticación del software de la Biblioteca de la Liga Departamental de Futbol de Tacna. c) Hacer un análisis comparativo de los resultados que de las evaluaciones de las características.	a) En la construcción de un módulo de autenticación con la programación orientada a aspectos el rendimiento es favorable. b) En la construcción de un módulo de autenticación con la programación orientada a objetos el rendimiento es desfavorable. c) En la comparación de módulos la diferencia de rendimientos es significativa.	• Cantidad de líneas de código • Coeficiente de complejidad ciclomática • Promedios de tiempo de ejecución y el consumo de memoria
Población: Todos los veinticinco (25) módulos del Software de la Biblioteca de la Liga Departamental de Futbol de Tacna.			
Muestra: El único (01) módulo de autenticación del Software de la Biblioteca de la Liga Departamental de Futbol de Tacna.			
Diseño: Descriptivo Comparativo. El diseño es con grupos experimentales y post prueba únicamente			
Análisis de datos: Estadística descriptiva e inferencial. Los datos recopilados fueron procesados y analizados con el paquete estadístico.			