EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Maestría

Estudio Experimental para Determinar la Relación entre las Métricas Estáticas y Dinámicas con las Estructuras Internas de Servicios Web

presentada por

Lic. Cruz Violeta Bautista Juárez

como requisito para la obtención del grado de

Maestra en Ciencias de la Computación

Director de tesis

Dr. René Santaolaya Salgado

Cuernavaca, Morelos, México. Septiembre de 2019.

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

Cuernavaca, Mor., 18/septiembre/2019
Oficio No. DCC/084/2019
Asunto: Aceptación de documento de tesis

DR. GERARDO VICENTE GUERRERO RAMÍREZ
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes de Comité Tutorial de la **Lic. Cruz Violeta Bautista Juárez**, con número de control M17CE086, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis profesional titulado **"Estudio experimental para determinar la relación entre las métricas estáticas y dinámicas con las estructuras internas de Servicios Web"** y hemos encontrado que se han realizado todas las correcciones y observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

DIRECTOR DE TESIS

Dr. René Santaolaya Salgado
Doctor en Ciencias de la Computación
4454821

REVISOR 1

Dr. Moisés González García
Doctor en Ciencias en la Especialidad de Ingeniería Eléctrica
7501724

REVISOR 2

Dr. Juan Carlos Rojas Pérez
Doctor en Ciencias en Ciencias de la Computación
6099372

C.p. M.E. Guadalupe Garrido Rivera - Jefa del Departamento de Servicios Escolares.
Estudiante
Expediente

NACS/lmz

"2019, Año del Caudillo del Sur, Emiliano Zapata"

Cuernavaca, Mor., 19/septiembre/2019
No. de Oficio: SAC/267/2019
Asunto: Autorización de impresión de tesis

LIC. CRUZ VIOLETA BAUTISTA JUÁREZ
CANDIDATA AL GRADO DE MAESTRA EN CIENCIAS DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"Estudio experimental para determinar la relación entre las métricas estáticas y dinámicas con las estructuras internas de Servicios Web"**, ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Conocimiento y tecnología al servicio de México"

SEP TecNM
CENTRO NACIONAL
DE INVESTIGACIÓN
Y DESARROLLO
TECNOLÓGICO
SUBDIRECCIÓN
ACADÉMICA

DR. GERARDO VICENTE GUERRERO RAMÍREZ
SUBDIRECTOR ACADÉMICO

C.p. Mtra. Guadalupe Garrido Rivera.- Jefa del Departamento de Servicios Escolares.
Expediente

GVGR/mcr

Dedicatoria

A mi madre, ***Teresa Juárez Hernández*** *que siempre está ahí, en todo momento y en cualquier circunstancia, soy quien soy gracias a ti mamá. Te agradezco tu amor incondicional, la alegría que me transmites, y por enseñarme con tu ejemplo a seguir adelante siempre a pesar de las adversidades. Gracias por tu apoyo en esta etapa de mi vida, al igual que siempre.*

A mi padre, ***Jacobo Bautista Pacheco*** *quien siempre me ha apoyado en todo momento, en cualquier decisión por pequeña o grande que sea y a pesar de las circunstancias. Gracias por tu amor, por tus consejos y valores que a pesar del tiempo me sigues inculcando para ser una persona de bien.*

A mis Hermanos, ***Angelito****,* ***Irene*** *y* ***Juan*** *quienes son el complemento de mi vida al igual que mis padres. Todos los buenos momentos que pasamos en familia los disfruto y valoro enormemente. Les agradezco todo su apoyo y cariño*

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado durante mis estudios de posgrado.

Al Tecnológico Nacional de México/Centro Nacional de Investigación y Desarrollo Tecnológico por la oportunidad de realizar mis estudios de posgrado y continuar con mi formación profesional.

A mi director de tesis, Dr. René Santaolaya Salgado, por todo su apoyo, por el tiempo, la paciencia y por siempre mostrar la disponibilidad para asesorarme y compartir sus conocimientos que fueron de vital importancia para poder concluir satisfactoriamente este trabajo. Gracias por la confianza y por aportar grandes cosas en mi formación profesional.

Al Dr. Juan Carlos Rojas Pérez y al Dr. Moisés González García por su tiempo en la revisión de esta tesis y por sus valiosas aportaciones que me fueron brindando durante todo el desarrollo, y que me sirvieron para mejorarla. De igual forma les agradezco el haber formado parte de mi formación durante mis estudios de posgrado.

Al Cuerpo Académico de Ingeniería de Software (CAIS) por las enseñanzas transmitidas en cada una de las clases y asesorías que me brindaron.

A mis compañeros y amigos de la maestría: Iván, Juan Carlos, Heidi, Xicoténcatl, Santiago y Gudiel, por brindarme su amistad, por los buenos momentos de convivencia y de trabajo. Gracias por haber estado conmigo a lo largo de esta etapa de mi vida.

Resumen

En la actualidad, los Servicios Web son ampliamente utilizados gracias a las ventajas que proporcionan, como la interoperabilidad y la reutilización. Sin embargo, para aprovechar dichas ventajas y aumentar su popularidad, estos servicios deben diseñarse e implementarse de forma correcta. Para verificar su correctitud es necesario evaluar, de manera objetiva, la calidad estructural de su arquitectura mediante métricas como las de *cohesión, coherencia* y *acoplamiento*. Además, existen otras métricas que se enfocan a evaluar atributos que denotan la calidad dinámica de los servicios, dentro de las cuales se encuentran el *tiempo de respuesta*, la *disponibilidad* y la *fiabilidad.*

A pesar de que el paradigma orientado a objetos es muy utilizado debido a las propiedades que maneja y a que facilita el reuso y mantenimiento del software. En ocasiones no se logran estas características debido a la falta de conocimiento acerca de las ventajas y desventajas que brindan diferentes arquitecturas que dan solución a una misma aplicación. Estas características se pueden verificar con base en evaluaciones a partir de las métricas de calidad existentes. De esta forma, se puede obtener conocimiento que contribuya a que los desarrolladores implementen arquitecturas de mejor calidad y conformes con los requerimientos funcionales especificados.

En este trabajo de tesis se realizó un estudio experimental para evaluar el comportamiento de las métricas de calidad *cohesión, coherencia, acoplamiento* y el *tiempo de respuesta*, en Servicios Web cuya estructura interna se organiza: en una única clase, en un conjunto de clases en colaboración y en un conjunto de clases internas. En el diseño experimental se considera el planteamiento de las hipótesis, las variables, los factores controlables y los no controlables, entre otros. Así mismo se utiliza la teoría de la estadística para el tratamiento de las muestras, así como de la interpretación de resultados.

Los resultados obtenidos se utilizaron para hacer un contraste entre las hipótesis planteadas y a partir de ellas derivar conclusiones. Estas conclusiones proporcionan conocimiento acerca de la estructura interna de Servicios Web que brinda los mejores valores de las métricas *cohesión, coherencia, acoplamiento* y el *tiempo de respuesta* en conjunto, considerando los resultados experimentales.

Palabras clave: Servicios Web, cohesión, coherencia, acoplamiento, tiempo de respuesta, estructura interna arquitectural.

Abstract

At present, web services are widely used thanks to the advantages they provide, such as interoperability and reuse. However, to exploit these advantages and increase their popularity, these services must be designed and implemented correctly. To verify its correctness is necessary to evaluate, in an objective way, the structural quality of its architecture through metrics such as cohesion, coherence, and coupling. Furthermore, other metrics focus on evaluating attributes that denote the dynamic quality of services, which include response time, availability and reliability.

Although the object-oriented paradigm is widely used due to the properties it handles and that facilitates software reuse and maintenance. Sometimes these characteristics are not achieved due to the lack of knowledge about the advantages and disadvantages offered by different architectures that give solution to the same application. These characteristics can be verified based on evaluations of existing quality metrics. In this way, knowledge can be obtained that contributes to developers implementing better quality architectures and complying with the specified functional requirements.

In this thesis work, an experimental study was performed to evaluate the behavior of the cohesion, coherence, coupling and response time metrics in web services whose internal structure is organized in a single class, in a set of collaborative classes and a set of internal classes. In the experimental design, the hypothesis, variables, controllable and non-controllable factors, among others, are considered. The theory of statistics is also used for the treatment of samples, as well as the interpretation of results.

The results were used to contrast the raised hypothesis and from them to derive the conclusions. These conclusions provide knowledge about the internal structure of web services that provides the best values of cohesion, coherence, coupling and response time metrics as a set, considering the experimental results.

Keywords: Web services, cohesion, coherence, coupling, response time, internal architectural structure.

Índice

Índice de figuras

Índice de tablas

Índice de ecuaciones

Tabla de abreviaturas

A1	Arquitectura uno. Se refiere a la arquitectura de única clase.
A2	Arquitectura dos. Se refiere a la arquitectura de conjunto de clases.
A3	Arquitectura tres. Se refiere a la arquitectura de clases internas.
CC	Conjunto de Clases. Arquitectura que consiste en un conjunto de clases con los atributos y las funciones distribuidas entre estas, las cuales participan colaborativamente en cada caso de uso para alcanzar la meta de valor de un requerimiento o capacidad del sistema.
CI	Clases Internas. Son clases definidas dentro de otras clases y sus instancias siempre están ligadas a una instancia de la clase contenedora.
COF	(Coupling Factor) Factor de acoplamiento. Es una medida de acoplamiento entre clases, excluyendo el acoplamiento debido a la herencia.
MAS	Muestreo Aleatorio Simple. Es un procedimiento de muestreo básico y fundamentado de otras estrategias de selección de muestra. Se caracteriza porque la selección se realiza de un listado de la población asignándole igual probabilidad a cada elemento.
SW	Servicio web. Son componentes de software utilizados a través de internet que se pueden integrar en aplicaciones distribuidas más complejas.
TR	Tiempo de Respuesta. Es el tiempo máximo garantizado requerido para completar una solicitud de servicio.
UC	Única Clase. Esta arquitectura consiste en la agrupación de toda la funcionalidad necesaria para satisfacer un caso de uso, en una sola clase de objetos.
ANTLR	ANother Tool for Language Recognition (Otra herramienta para reconocimiento de lenguajes). Es un potente generador de analizadores para leer, procesar, ejecutar o traducir texto binario o texto estructurado.

Glosario de términos

Acoplamiento	Dos objetos se acoplan si, y solo si, al menos uno de ellos actúa sobre el otro. Dado que el acoplamiento es el grado de interacción entre clases, la idea básica que subyace a todas las métricas de acoplamiento es muy simple: se cuenta el número de interacciones entre clases del sistema. Sin embargo, hay una variación considerable en función de lo que cuenta como una interacción, cómo se realiza el conteo y cómo se normalizan los totales (Fenton & Ohlsson, 2000).
Calidad de servicio (QoS)	La QoS es "la totalidad de las características de un producto o servicio que tienen que ver con su capacidad de satisfacer necesidades declaradas o implícitas" (ISO IEC, ISO 8402:1994).
Caso de uso	Representa la meta de interacción entre un actor y el sistema. La meta representa un objetivo medible y significativo para el actor. Capturan y por lo tanto contienen los requerimientos funcionales de los sistemas, en un formato fácil de seguir y fácil de leer (Fowler, 1997).
Clase	Una clase es la descripción de un conjunto de objetos que consta de métodos y atributos que resumen características comunes de un conjunto de objetos. Se pueden definir muchos objetos de la misma clase (Aguilar, 1996).
Coherencia	La coherencia de un conjunto de objetos mide qué tan bien contribuyen sus miembros a un propósito u objetivo común, definido externamente (Mišić, 2000).
Cohesión	La manera y el grado en que las tareas realizadas por un solo módulo de software están relacionadas entre sí (Ieee, 1990).
Meta de valor	En el contexto de esta investigación una meta u objetivo de valor a nivel de función, como unidad de programa, es realizar una única operación funcional. A nivel de módulo o subsistema, una meta de valor u objetivo es el de satisfacer un caso de uso o requerimiento específico. A nivel de sistema una meta de valor u objetivo es resolver una aplicación completa o proceso de negocio planteada por un cliente o usuario.
Métricas de QoS de Servicio Web	La documentación, el cumplimiento y las mejores prácticas son métricas estáticas extraídas, que miden la usabilidad de la interfaz del Servicio Web desde el aspecto de QoS. El tiempo de respuesta, la disponibilidad, el rendimiento, el éxito, la confiabilidad y la latencia son métricas dinámicas que miden el rendimiento y la experiencia general del Servicio Web (Wang, Kessentini, Hassouna, & Ouni, 2017).

Métricas dinámicas	La clase de métricas dinámicas de software, son aquellas que capturan el comportamiento dinámico del sistema de software y generalmente se obtienen de los rastros de ejecución del código o de los modelos ejecutables (Chhabra & Gupta, 2010). Las métricas dinámicas se calculan sobre la base de los datos recopilados durante la ejecución real del sistema, y así reflejan directamente los atributos de calidad (tales como el rendimiento, tasas de error, etc.) del software en su modo operativo (Chhabra & Gupta, 2010).
Métricas estáticas	Las métricas estáticas se centran en las propiedades estáticas del software. En la literatura se ha propuesto una serie de métricas estáticas para la medición de acoplamiento, cohesión y otros atributos de software orientado a objetos utilizando el código fuente o desde el diseño del software, que son de naturaleza estática. Las métricas estáticas pueden cuantificar varios aspectos de la complejidad del diseño o del código fuente del sistema de software (Chhabra & Gupta, 2010).
Módulo	Son las partes en las que se subdivide una aplicación de software, cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las partes restantes (Aguilar, 1996).
Servicio web	Los Servicios Web son componentes de software utilizados a través de Internet que se pueden integrar en aplicaciones distribuidas más complejas (Alonso, Casati, Kuno, Machirajuu, & Verlag, 2004)
Tiempo de respuesta	El tiempo máximo garantizado (promedio o mínimo) requerido para completar una solicitud de servicio (Gunther, 1998) .

Capítulo 1. Introducción

1.1 Presentación

En la actualidad, los Servicios Web se han convertido en una técnica importante para la construcción de sistemas orientados a servicios (Xu, et al., 2016). Sin embargo, para garantizar su popularidad y aumentar su usabilidad, deben diseñarse e implementarse correctamente (Wang et al., 2017). Un servicio puede ser compuesto o invocado por otros servicios a través de protocolos estándar. Por lo tanto, los desarrolladores de software pueden reducir la carga de trabajo en función de la alta reusabilidad y extensibilidad de los servicios. Así mismo, a través de interfaces bien diseñadas, los desarrolladores pueden reusar los Servicios Web existentes de manera fácil y eficiente (Wang et al., 2017).

La cohesión y el acoplamiento son dos métricas importantes que denotan la calidad en el nivel de diseño estructural de un sistema de software (Rathee & Chhabra, 2018). Así mismo, los atributos no funcionales de la calidad de los servicios (QoS), como el tiempo de respuesta, la disponibilidad y la fiabilidad, se consideran parte de las principales preocupaciones en la gestión de los Servicios Web (Wang et al., 2017). Adicionalmente, los desarrolladores buscan Servicios Web que no solo tienen una estructura bien diseñada, sino también buenas características de rendimiento (Wang et al., 2017).

En la actualidad, existen investigaciones para transformar el software legado en componentes o servicios reusables para el desarrollo de nuevas aplicaciones a partir de éstos. Sin embargo, las malas prácticas de diseño orientado a objetos impiden distinguir adecuadamente módulos autónomos que sean completos y suficientes para cumplir las metas de valor de los componentes y/o servicios, lo cual representa un obstáculo para construir aplicaciones de software o software como servicios de procesos de negocios a partir de componentes y/o servicios reusables. Por lo tanto, desde esta perspectiva, aumentar la calidad del diseño del software es un desafío clave de investigación en el desarrollo de software orientado a objetos (Rathee & Chhabra, 2018). Para esto, existen técnicas como la refactorización de software legado que toma como referencia malas prácticas de diseño que pueden afectar tanto a factores estáticos como a factores dinámicos de calidad, tales como cohesión, coherencia, acoplamiento y rendimiento.

A la fecha no se tiene conocimiento acerca de los valores deseables de calidad de arquitecturas orientadas a objetos que proporcionen un balance entre métricas estáticas y dinámicas. Debido a que son varias las métricas que afectan en mayor o menor medida la calidad de las arquitecturas de software, en las que unas se contraponen a otras, es importante saber los niveles entre estas métricas para que los diseños arquitecturales proporcionen un equilibrio en su calidad, que permita distinguir correctamente módulos autónomos para reuso.

En el *Tecnológico Nacional de México/ CENIDET* se han desarrollado algunos trabajos de investigación enfocados a la generación de Servicios Web con diferentes estructuras internas en su arquitectura[1]. Específicamente se han considerado tres arquitecturas diferentes para un mismo Servicio Web: 1) cada servicio se representa en una arquitectura de clases de objetos con los atributos y las funciones distribuidas entre estas, las cuales participan colaborativamente para satisfacer un caso de uso (León, 2009), 2) cada servicio se construye a partir de la agrupación de toda la funcionalidad necesaria para satisfacer un caso de uso, en una sola clase de objetos. (Legorreta, 2017) y 3) cada servicio se construye a partir de un conjunto de clases internas anidadas que colaboran entre sí para satisfacer un caso de uso (Gallardo, 2018). Comparativamente, cada una de estas arquitecturas muestra ventajas y desventajas en sus características de calidad, tanto estáticas como dinámicas, para satisfacer el mismo caso de uso.

Por lo anterior, en este trabajo de tesis se realizó un estudio experimental con el fin de obtener conocimiento acerca de los valores obtenidos mediante las métricas de calidad: coherencia, cohesión, acoplamiento y tiempo de respuesta en cada una de las tres diferentes arquitecturas de los Servicios Web.

1.2 Organización del documento de tesis

En esta sección se presenta la organización del presente documento de tesis:

En el capítulo dos se presenta el estado del arte para esta investigación. Se describen algunos trabajos relacionados con el tema, considerando los que se han realizado en los últimos cinco años. Así mismo, se incluyen trabajos desarrollados en el *Tecnológico Nacional de México/ CENIDET* que forman parte de los antecedentes del presente trabajo. Se puntualizan además el planteamiento del problema, los objetivos, la justificación, los alcances y las limitaciones.

En el capítulo tres se presenta el marco teórico para esta tesis. Se incluye la descripción de los términos que se utilizan durante el desarrollo del trabajo y que forman parte del fundamento teórico que sustenta la investigación.

En el capítulo cuatro se describe el diseño experimental. Se incluyen las hipótesis, variables dependientes e independientes y demás elementos estadísticos. Adicionalmente se describe el plan de pruebas para el experimento.

[1] Nota: Debido a que las diferentes arquitecturas de servicios web consideradas en esta tesis, están relacionadas con el número de entidades de software en su estructura interna, de aquí en adelante, en este documento nos referiremos como *"niveles de granulación"* a las "*estructuras internas"* de las tres diferentes arquitecturas con las que se hizo la experimentación.

En el capítulo cinco se presentan las actividades que se realizaron y los aspectos que se tomaron en cuenta para la ejecución del experimento. Se describen las pruebas que se hicieron con las métricas estáticas y el tiempo de respuesta de los Servicios Web.

En el capítulo seis que corresponde al análisis estadístico e interpretación de resultados, se presenta el tratamiento de los datos obtenidos durante las pruebas y se describen los métodos estadísticos que se utilizaron. Así mismo, se realiza el contraste de las hipótesis y una comparación entre las arquitecturas con las que se trabajó.

Por último, en el capítulo siete se presentan las conclusiones obtenidas a partir del trabajo de tesis y se proponen algunos de los trabajos futuros que se pudieran realizar.

Capítulo 2. Marco de referencia

2.1 Estado del arte

Las arquitecturas orientadas a servicios evolucionan con éxito para actualizar las funciones existentes expuestas a los usuarios y corregir posibles errores. Este proceso de evolución puede tener un impacto negativo en la calidad de diseño de los Servicios Web (Wang et al., 2017).

Se han propuesto buenos principios de calidad para el diseño orientado a servicios, tales como el bajo acoplamiento, la capacidad de compilación y la flexibilidad. Sin embargo, varias razones podrían llevar a la violación de estos principios provocando la existencia de malas prácticas de diseño que pueden generar problemas de mantenimiento, extensibilidad o reusabilidad de los Servicios Web.

Estudios recientes han abordado los problemas antes mencionados. Sin embargo, estos se habían centrado únicamente en los detalles de implementación (código fuente) y se dejaban de un lado los aspectos de calidad de servicio (QoS), como el tiempo de respuesta. Debido a la importancia de considerar los dos aspectos antes mencionados, en los últimos años se han propuesto enfoques que consideran estos dos aspectos en conjunto (Wang et al., 2017). Actualmente existen métricas de calidad para la evaluación de los Servicios Web, de igual forma existen anti-patrones que al identificarlos y eliminarlos pueden contribuir a mejorar la calidad del software. Una definición de anti-patrón incluye: la especificación de malas prácticas en términos de elementos del modelo y las acciones a tomar para resolver el problema (Cortellessa, Di Marco, & Trubiani, 2014).

A continuación se presentan algunos anti-patrones enfocados a los Servicios Web que han sido documentados en la literatura (Král & Žemlička, 2009), (Dudney, 2003):

- *God object Web service (GOWS):* Este anti-patrón implementa una multitud de métodos relacionados con diferentes abstracciones comerciales y técnicas en un solo servicio.

- *Fine-grained Web service (FGWS*): Es un servicio demasiado fino cuya sobrecarga (comunicaciones, mantenimiento, etc.) supera su utilidad.

- *Chatty Web service (CWS):* Representa un anti-patrón en el que se requiere un alto número de operaciones para una abstracción completa.

- *Ambiguous Web service (AWS*): Es un anti-patrón donde los desarrolladores usan nombres ambiguos o sin sentido para indicar los elementos principales de los elementos de la interfaz.

- *Redundant PortTypes (RPT):* Este anti-patrón se da cuando múltiples tipos de puertos están duplicados con un conjunto similar de operaciones.

- *CRUDy Interface (CI):* Es un anti-patrón en el que el diseño fomenta los servicios del comportamiento de RPClike declarando las operaciones de creación, lectura, actualización y eliminación (CRUD), por ejemplo, createX(), readY(), etc.

- *Maybe It is Not RPC (MNR):* Es un anti-patrón en el que el servicio web proporciona principalmente operaciones de tipo CRUD para entidades comerciales importantes.

Por otra parte, también se han propuesto diferentes anti-patrones de rendimiento en donde se toman en cuenta diferentes aspectos del código y los problemas que se pudieran presentar. Para contrarrestar estos problemas, existen diferentes enfoques orientados a su detección y solución. Actualmente, existen anti-patrones de rendimiento que son los elementos más prometedores que pueden impulsar la detección. Los anti-patrones de rendimiento básicamente representan patrones típicos que, si ocurren en un modelo, pueden inducir problemas de rendimiento.

Hasta hace algunos años, de acuerdo a la literatura, los anti-patrones de rendimiento solo se han definido en lenguaje natural. Sin embargo, actualmente se están realizando trabajos enfocados a la representación formal de estos anti-patrones con el fin de apoyar su detección automática. A partir de esta detección, se han presentado soluciones típicas que consisten en alternativas de diseño que modifican el modelo arquitectónico del software original para lograr un mejor rendimiento. Cabe mencionar que en varios trabajos, la decisión sobre qué anti-patrones eliminar recae sobre los diseñadores de software, quienes determinan las acciones a realizar, ya que también se debe tomar en cuenta que el hecho de eliminar todos los anti-patrones no garantiza mejoras en el rendimiento y en la calidad del software en general (Trubiani, Koziolek, Cortellessa, & Reussner, 2014).

A pesar de que actualmente existen muchos métodos y criterios de refactorización, cuando se trata de atributos no funcionales este aspecto sigue siendo crítico, principalmente porque las características no funcionales del software son difíciles de evaluar y las acciones de refactorización apropiadas pueden ser difíciles de identificar (Arcelli, Cortellessa, & Di Pompeo, 2018).

Además de los anti-patrones, existen una gran cantidad de métricas orientadas a medir la calidad de las arquitecturas orientadas a servicios tales como el acoplamiento, la cohesión, líneas de código, tiempo de respuesta, latencia, entre otras. Se han realizado trabajos en donde se modifican arquitecturas con el fin de mejorar los valores en ciertas métricas. Sin embargo, dentro de la literatura revisada se observa que no se ha realizado experimentación con estas métricas o al menos no con las métricas en conjunto, es decir, considerando atributos estáticos y dinámicos. Así mismo, no se han probado arquitecturas específicas que en la actualidad son ampliamente utilizadas, como es el caso de las que se implementan en el paradigma orientado a objetos.

2.2 Trabajos relacionados

En esta sección se describen algunos trabajos que están relacionados con los temas que se abordan en esta investigación. Cabe mencionar que no se encontraron artículos que tengan un objetivo igual o similar al del presente trabajo de tesis.

Guilt-based handling of software performance antipatterns inpalladio architectural models (Trubiani et al., 2014)**.** En este trabajo se propone un enfoque para la detección automática de anti-patrones de rendimiento en la fase de refactorización durante el desarrollo de software. A partir de la detección de anti-patrones, se retroalimenta a los ingenieros de software, proporcionándoles alternativas de diseño que permitan resolverlos. Para optimizar este proceso, se incluye una metodología de clasificación que identifica entre un conjunto de anti-patrones, los que más contribuyen a la violación de requisitos específicos de rendimiento.

Se utilizan anti-patrones para detectar fallas en el diseño arquitectural, arquitectura de despliegue, entre otras. El objetivo es mejorar el rendimiento del sistema mediante la eliminación de malas prácticas de diseño.

Automated translation among EPSILON languages for performance-driven UML software model refactoring (Arcelli, Cortellessa, & Di Pompeo, 2016). Este artículo de investigación propone un enfoque para reducir la fragmentación que sufren los modelos de software en diferentes paradigmas, lenguajes y metamodelos. Utilizan el entorno EPSILON debido a que proporciona un conjunto de lenguajes para verificar propiedades y aplicar refactorización en modelos.
Se presenta la automatización dirigida a traducir código que define reglas de detección de anti-patrones de rendimiento y las acciones de refactorización entre tres idiomas de EPSILON. Para esto se realiza la implementación de un *Porting Engine*. La automatización ayuda a reducir el esfuerzo de escritura de código, en el contexto de la refactorización basada en el rendimiento de los modelos UML, mientras explota el soporte específico proporcionado por la semántica de ejecución diferente de los lenguajes considerados. Con

este enfoque, que es abierto y extensible, se detectan seis anti-patrones de rendimiento y se aplican siete refactorizaciones.

On the Value of Quality of Service Attributes for Detecting Bad Design Practices (Wang et al., 2017)**.** En este artículo se implementa un enfoque automatizado para generar reglas de detección de defectos de servicios, que no solo considera las métricas de nivel de código, sino también los atributos de calidad del servicio. Se implementa un algoritmo multi-objetivo NSGA-II para generar los mejores conjuntos de reglas de detección que maximizan la cobertura de anti-patrones de servicio web y minimizan la detección de ejemplos de diseño de Servicios Web bien diseñados.

Multi-view refactoring of class and activity diagrams using a multi-objective evolutionary algorithm (Mansoor, Kessentini, Wimmer, & Deb, 2017). Este trabajo de investigación propone un enfoque para generar una secuencia óptima de refactorizaciones que mejore la calidad de diferentes diagramas de modelado, preservando las restricciones de conservación del comportamiento. Se utiliza el algoritmo NSGA-II, que hace evolucionar una población de soluciones candidatas hacia la solución casi óptima, para generar la secuencia en términos de encontrar compensaciones entre la maximización de la calidad de los diagramas de clases y de actividad, y de la minimización de restricciones de conservación del comportamiento.

Performance-driven software model refactoring (Arcelli et al., 2018)**.** En este trabajo de investigación se propone un marco para la refactorización de modelos de software de manera automática. Se utiliza la plataforma EPSILON que proporciona un conjunto de lenguajes, intérpretes y herramientas para verificar propiedades de los modelos, y aplicar refactorización que resuelvan las malas practicas de diseño que podrían disminuir el rendimiento del sistema. Con base en tres lenguajes de la plataforma EPSILON que son: EPL, EVL Y EWL, se proporcionan diferentes tipos de soporte de refactorización inducidos por su semántica de ejecución distinta.

Exploiting load testing and profiling for Performance Antipattern Detection (Trubiani, Bran, van Hoorn, Avritzer, & Knoche, 2018)**.** En este articulo se proporciona un proceso sistemático, basado en pruebas de carga y datos de perfiles para identificar problemas de rendimiento con base en datos obtenidos en tiempo de ejecución. Se ejecutaron pruebas de carga basadas en características del perfiles operativos recopilados, para así producir cargas de trabajo representativas. A partir de estas pruebas se recopilan los datos de rendimiento y se identifican los anti-patrones de rendimiento, para posteriormente realizar el proceso de refactorización.

Improving Cohesion of a Software System by Performing Usage Pattern Based Clustering (Rathee & Chhabra, 2018). En este artículo se propuso una nueva métrica de cohesión para el software orientado a objetos, denominada Cohesión basada en patrón de uso

(UPBC), que se calcula a nivel de módulo. Inicialmente se considera la clase como un módulo y posteriormente el grupo de clases (es decir, un paquete) se considera como un módulo, con el objetivo de mejorar la cohesión general. Esta métrica utiliza los patrones de uso frecuente (FUP) extraído de diferentes interacciones de funciones miembro para capturar la cohesión del módulo. Además, el valor de cohesión medido se utiliza para realizar la agrupación de módulos con el fin de aumentar la cohesión y disminuir el acoplamiento entre los módulos simultáneamente.

Improving modular structure of software system using structural and lexical dependency (Amarjeet & Chhabra, 2017)**.** En este trabajo de investigación se propone un enfoque que considera varios tipos de dependencias tanto estructurales como léxicas junto con su importancia relativa para re-modularizar los sistemas orientados a objetos (OO).

Se calcula la fuerza de acoplamiento entre clases utilizando diferentes ponderaciones en términos de diversos mecanismos de dependencias léxicas y estructurales. Sobre la base de los diferentes tipos de dependencias léxicas y estructurales y según sus variantes no ponderadas / ponderadas, se diseñaron los siguientes 24 esquemas de acoplamiento: basados en la estructura, de base léxica, y una combinación estructural-léxica basada en (SLT) y SLTFIDFW). Los valores obtenidos a través de estos esquemas de acoplamiento se utilizan en la función objetivo de acoplamiento y cohesión de la AG. Junto con este objetivo, algunas funciones de apoyo, como MCI y MSI, se utilizaron para impulsar el proceso de optimización hacia una solución de modularización de buena calidad.

Architecture-level software performance abstractions for online performance prediction (Maria & Brosig, 2014)**.** En este artículo se analizaron los escenarios típicos de predicción de rendimiento en línea y propusieron un metamodelo de rendimiento para (i) expresar y resolver dependencias de parámetros y contextos, (ii) modelar abstracciones de servicios a diferentes niveles de granularidad y (iii) modelar la implementación de componentes de software en escenarios complejos de recursos. El metamodelo presentado es un subconjunto del metamodelo de Descartes (DMM) para la predicción del rendimiento en línea, específicamente diseñado para su uso en escenarios en línea. El enfoque se validó en el contexto de escenarios de predicción de rendimiento en línea realistas y representativos basados en el estándar de referencia SPECjEnterprise2010.

Applying Design Patterns to Remove Software Performance Antipatterns: A Preliminary Approach. (Arcelli & Di Pompeo, 2017)**.** El objetivo de este trabajo de investigación fue reducir la brecha entre los patrones de diseño y anti-patrones de rendimiento, proporcionando un enfoque preliminar destinado a aplicar los primeros para eliminar los últimos que se producen en artefactos de software. Se explota una asociación entre los patrones de diseño y anti-patrones de rendimiento, hacia el cumplimiento de los requisitos de rendimiento y la mejora de la calidad del software. También se trabajó en un

contexto difuso, donde valores de umbral relacionados con las métricas de anti-patrones de rendimiento no se puede determinar, pero solo lo hacen sus límites inferior y superior.

Using Cohesion and Coupling for Software Remodularization: Is It Enough? (Candela, Bavota, Russo, & Oliveto, 2016). El objetivo de este trabajo de investigación fue el análisis tanto objetiva como subjetiva del fenómeno de la búsqueda del equilibrio entre cohesión y acoplamiento. Se presentaron los resultados de un estudio que analiza la calidad de la modularización, en términos de cohesión y acoplamiento de paquetes, de 100 sistemas de código abierto, y una encuesta realizada con 29 desarrolladores con el objetivo de comprender los factores determinantes que tienen en cuenta al realizar tareas de modularización. Los resultados obtenidos se utilizaron para extraer un conjunto de lecciones aprendidas que podrían considerarse para diseñar recomendadores de re-modularización más efectivos. En particular se empleó un enfoque de dos objetivos destinado a maximizar la cohesión del paquete y minimizar el acoplamiento del paquete. Una vez que se identificó el conjunto de re-modularizaciones (casi) óptimas, se midió la efectividad de operaciones de refactorización, aplicada a la modularización original de cada sistema desde la solución que proporciona el compromiso óptimo entre cohesión y acoplamiento.

En la Tabla 1 se presenta un resumen comparativo de los trabajos relacionados al trabajo de tesis, en la cual se utilizan los siguientes criterios de comparación:

Lo que se aclama: Se refiere a lo que se presume que se hizo en cada uno de los trabajos relacionados. Se distinguen tres casos: Detección (DET), Refactorización (RE) y Experimentación (EX).

Enfoque: Se describe brevemente el método propuesto en cada uno de los trabajos relacionados.

Objetivo: Se describe brevemente el propósito o lo que se quiere lograr mediante la realización del trabajo de tesis.

Métricas estáticas: Este criterio se utiliza para indicar las métricas estáticas que se utilizan en el trabajo, se incluyen métricas como: Acoplamiento aferente (Ca), Cohesión entre métodos de la clase (CAM), Acoplamientos eferentes (Ce), Métrica de acceso a datos (DAM), Profundidad del árbol de herencia (DIT), Carencia de cohesión en los métodos (LCOM), Carencia de cohesión en los métodos (LCOM3), Líneas de código (LOC), Medida de Abstracción Funcional (MFA), Medida de Agregación (MOA), Cantidad de hijos (NOC), Número de métodos públicos (NPM), Respuesta para una clase (RFC), Métodos ponderados por clase (WMC), Complejidad promedio del método (AMC), La complejidad ciclomática de McCabe (CC), Factor de acoplamiento (COF).

Métricas dinámicas: Este criterio se utiliza para indicar las métricas que evalúan cada uno de los trabajos relacionados en tiempo de ejecución. Se incluyen métricas como: tiempo de respuesta, disponibilidad, rendimiento, éxito, fiabilidad, conformidad, latencia, entre otras.

Producto resultante: Representa el producto final obtenido en cada uno de los trabajos analizados. Para esta comparativa se incluyen los siguientes criterios: Conocimiento (CONOC) herramienta (HER), una nueva métrica (MTR), un algoritmo (ALG), un framework (FRM) o una metodología (ME).

Niveles de granulación: Este criterio se refiere a la cantidad de entidades que se integran en la estructura interna de un componente de software, tales como: clases o funciones.

Tipos de procesos: Los tipos de procesos utilizados en los trabajos de investigación se clasifican dependiendo del nivel en el que el usuario interviene, estos pueden ser: Automático (AUT) sin ninguna intervención del usuario, Semi-automático (SAU), con una mínima intervención del usuario, y Manual (MAN), máxima intervención del usuario en el proceso.

En la tabla comparativa se incluyen también algunos acrónimos como No Aplica (NA), Anti-patrones de Rendimiento (AR), No Definido (ND).

Tabla 1. Resumen comparativo de los trabajos relacionados con el tema de tesis

Artículo	Lo que se aclama	Objetivo	Enfoque	Metricas Estáticas	Métricas Dinámicas	Niveles de granula-ción	Tipo de proceso	Producto resultante
(Trubiani et al., 2014).	R	Mejorar el rendimiento de sistemas de software.	Detección automática de AR en la fase de refactorización y retroalimentación a los ingenieros para su solución.	NA	NA	ND	SA	HER PCM Bench (extensión)
(Arcelli et al., 2016).	R	Reducir la fragmentación de los modelos de software en diferentes lenguajes.	Traducción de código que define reglas de detección de AR y su respectiva refactorización en modelos UML, utilizando EPSILON.	NA	NA	ND	SA	FRM
(Wang et al., 2017)	DET AP	Detectar defectos estructurales de Servicios Web.	Generación de reglas de detección de anti-patrones mediante el algoritmo NSGA-II. Detección de diseños adecuados.	Ca CAM CBO Ce LCOM NOC	Response Availability Throughput Successability Reliability Compliance Latency	ND	SA	ALG (NSGA-II)
(Mansoor et al., 2017).	RE	Mejorar la calidad de diagramas de modelado de SW.	Secuencia óptima de refactorizaciones en diagramas de clases y de actividad en el modelado de SW, mediante algoritmo NSGA-II.	DSC NOM DCC NOP NOH CAM ANA NNO	NA	ND	SA	FRM
(Amarjeet & Chhabra, 2017)	RE	Impulsar el proceso de optimización hacia una solución de	Se consideran varios tipos de dependencias tanto estructurales como léxicas	Cohesión COF	NA	ND	MAN	MET

		modularización de buena calidad.	junto con su importancia relativa para remodularizar los sistemas orientados a objetos (OO).					
(Arcelli & Di Pompeo, 2017)	RE	Reducir la brecha entre los patrones de diseño y anti-patrones de rendimiento.	Se propone un criterio de clasificación para los patrones de diseño con el fin de impulsar la elección del más adecuado para eliminar un cierto anti-patrón	NA	NA	ND	SA	FRM
(Arcelli et al., 2018).	RE	Resolver problemas de modelado de SW que afectan el rendimiento de SW.	Refactorización en diagramas de clase, de secuencia y de implementación usando el entorno EPSILON	NA	NA	ND	AUT	FRM
(Candela, Bavota, Russo, & Oliveto, 2016).	EXP	Proporcionar un conjunto de lecciones aprendidas que puedan considerarse para diseñar recomendadores de remodularización más efectivos.	Analizar objetivamente y subjetivamente el fenómeno de la búsqueda del equilibrio entre cohesión y acoplamiento. Se analiza la calidad de la modularización, en términos de cohesión y acoplamiento de paquetes	NA	NA	ND	MAN	CONOC
(Maria & Brosig, 2014)	RE	Mejorar el rendimiento del software	Se propone un metamodelo de rendimiento para expresar y resolver dependencias de parámetros y contextos. Se analizan escenarios típicos de predicción de rendimiento en línea.	NA	Rendimiento	ND	SA	MET

(Rathee & Chhabra, 2018)	RE	Mejorar la cohesión modificando la arquitectura interna	Propuesta de una nueva métrica basada en patrón de uso, para medir la cohesión en software orientado a objetos con diferentes arquitecturas agrupadas mediante el algoritmo FUPClust.	UPBC	NA	1 clase 1 conjunto de clases	SA	MET
(Trubiani et al., 2018).	RE	Mejorar el rendimiento del sistema SW.	Identificación de AR con base en pruebas de carga y datos de perfiles en tiempo de ejecución.	NA	Rendimiento	ND	SA	MET
Trabajo de tesis	EX	Obtener conocimiento acerca del comportamiento de métricas estáticas y dinámicas en sevicios web.	Diseño experimental para realizar pruebas de comportamiento de los factores estaticos y dinámicos de Servicios Web de diferente nivel de granulación.	Cohesión Coheren-cia Acopla-miento	Rendimiento	Unica clase Conjunto de Clases Clases internas	NA	CONOC

El presente trabajo de tesis consiste en un experimento para obtener conocimiento acerca del comportamiento de métricas estáticas y dinámicas en los Servicios Web. A diferencia de los trabajos revisados en el estado del arte, en los cuales se trabaja con anti-patrones de Servicios Web y de rendimiento, en esta tesis se trabaja con métricas bien conocidas y ampliamente utilizadas para evaluar la calidad de las arquitecturas de software. Así mismo, se evalúan tres arquitecturas diferentes con el fin de que, a partir de los resultados, se aporte conocimiento que pueda servir a los desarrolladores cuando requieran decidir cuál de las arquitecturas utilizar. Cabe mencionar que la mayoría de los trabajos relacionados se enfocan en la refactorización de código o de modelado a partir de la detección de anti-patrones, así como también a partir de diferentes formas de agrupamiento de elementos del software. Sin embargo, en ninguno se realiza experimentación tomando en cuenta métricas estáticas y dinámicas en conjunto.

A continuación, se puntualizan las principales diferencias de este trabajo de tesis contra los trabajos relacionados descritos anteriormente.

- Se realiza un diseño experimental planteando hipótesis y demás elementos estadísticos.
- Se utilizan métricas de calidad de software en lugar de anti-patrones.
- Se evalúa la calidad del software con métricas estáticas y dinámicas
- No se realiza refactorización, solo se genera conocimiento.
- Se experimenta con Servicios Web en ejecución.
- Se evalúan tres arquitecturas específicas y bien conocidas del paradigma orientado a objetos.

2.3 Desarrollos tecnológicos realizados en el Tecnológico Nacional de México/ CENIDET

Alineados con las corrientes tecnológicas actuales, en el laboratorio de ingeniería de software del Cenidet se han atendido trabajos de investigación para la transformación de Marcos Orientados a Objetos hacia Marcos de Servicios Web, utilizando como criterios de agrupación diferentes niveles de granulación. A continuación, se presentan los trabajos que se han desarrollado y que anteceden a este trabajo de tesis.

Generación de Servicios Web desde Marcos Orientados a Objetos a partir de Clases Colaborativas en Casos de Uso (MOOaSW) (León, 2009). En esta investigación se propuso un procedimiento para generar Servicios Web desde el código legado de marcos orientados a objetos, en donde cada servicio satisface un caso de uso. El criterio de agrupación utilizado son las clases, atributos y funciones que participan colaborativamente en cada caso de uso. El procedimiento propuesto consta de tres pasos: 1) Analizar de forma

estática el código fuente del marco orientado a objetos. 2) Definir el código necesario para satisfacer cada caso de uso, y 3) Generar el código de los Servicios Web correspondiente. Es importante mencionar que la arquitectura de las clases del código definido para cada servicio, debe mantener la misma estructura interna que presenta el marco original.

Estudio Experimental para determinar la relación entre la estructura interna de Servicios Web y valores de QoS (Guadarrama, 2013). En esta investigación se realizó un estudio experimental para determinar cómo influye la estructura interna de la arquitectura de los Servicios Web, sobre el tiempo de respuesta, y cuál sería la estructura interna que ofrece el mejor tiempo de respuesta. Se utilizaron 4 niveles de granulación que van del más grueso al más fino.

Con base en las pruebas se concluyó que las arquitecturas con estructuras internas de grano grueso presentan un mejor rendimiento en función del tiempo de respuesta. Sin embargo, estas arquitecturas presentan ventajas y desventajas, tales como: baja cohesión, bajo acoplamiento y problemas de movilidad, lo que dificulta su reuso y mantenimiento. También se concluye que, desde el punto de vista de la ingeniería de software, es necesario balancear el tiempo de respuesta contra métricas de calidad para reuso y mantenimiento.

Transformación de Marcos de Aplicaciones Orientados a Objetos hacia Marcos con Arquitectura Orientada a Servicios (Legorreta, 2017). En atención al trabajo de (Guadarrama, 2013), en esta investigación se construyeron Servicios Web de grano grueso a partir del código legado de marcos orientados a objetos, mediante la agrupación de toda la funcionalidad necesaria para satisfacer un caso de uso, dentro de una sola clase de objetos. Esto con el propósito de mejorar la auto-suficiencia y el tiempo de respuesta de Servicios Web. El procedimiento consiste de cuatro pasos. 1) Analizar el código fuente del MOO de entrada. 2) Definir el código necesario para cada caso de uso. 3) Integrar el código de cada caso de uso en una única entidad y 4) generar el Servicio Web correspondiente. En la investigación se concluyó, que con el criterio de agrupación utilizado se tiene un mejor tiempo de respuesta con respecto al enfoque propuesto en (León, 2009), sin embargo, presenta problemas de cohesión y coherencia.

Generación de Servicios Web desde Marcos Orientados a Objetos con el Enfoque de Clases Internas (MOOinWS) (Gallardo, 2018). Basados en los tres trabajos descritos anteriormente, el enfoque de esta investigación fue alcanzar un balance entre las métricas de cohesión, coherencia, acoplamiento y tiempo de respuesta, para lo cual se propuso que cada caso de uso se integrara en un conjunto de clases internas colaborativas para satisfacer el caso de uso, con el objetivo de mejorar la modularidad de los Servicios Web obtenidos. Después de las pruebas y la comparación con los enfoques de (Legorreta, 2017) y (León, 2009), en esta tesis se comprobó que con el enfoque de clases internas utilizado se alcanzó el objetivo propuesto, produciendo un balance y los mejores resultados de las métricas planteadas.

2.4 Planteamiento del problema

El problema radica en que no se tiene conocimiento acerca del nivel de granulación que proporcione un balance entre métricas de calidad estáticas (coherencia, cohesión y acoplamiento) y dinámicas (como el tiempo de respuesta) en componentes de software y/o servicios de software reusables. No contar con este conocimiento, limita la capacidad de decisiones de los desarrolladores de software durante la creación de software como servicios o como aplicaciones. Esta limitación da como resultado que los procesos de software no respondan en tiempo y forma como esperan los clientes.

2.5 Objetivos

2.5.1 Objetivo general

El objetivo de este proyecto es generar conocimiento acerca de las diferencias en valores de calidad que proporcionan las arquitecturas de software con diseños estructurales diferentes. Mediante un diseño experimental que permita realizar pruebas de comportamiento de estos factores sobre la estructura interna de los Servicios Web, con diferente nivel de granulación.

2.5.2 Objetivos específicos

- Aportar conocimiento acerca del comportamiento en términos de los diferentes valores que toman los factores de calidad coherencia, cohesión, acoplamiento y tiempo de respuesta de Servicios Web, ante tres diferentes diseños estructurales internos en su arquitectura: en una única clase, en arquitecturas de clases relacionadas y en arquitecturas con clases internas anidadas; con el objetivo de saber cuál de estas tres arquitecturas proporciona más ventajas al implementarla.

- Generar conocimiento acerca de qué arquitectura es la más conveniente de utilizar para la construcción de servicios web, de acuerdo a los requerimientos. Y que este conocimiento pueda ser útil durante la etapa de diseño de software.

2.6 Justificación

Un Servicio Web es una aplicación lógica programable, accesible utilizando protocolos de comunicación estándar, tal como los protocolos de Internet (Wang et al., 2017). Como los componentes, los Servicios Web ofrecen un conjunto de funcionalidades, encerradas en cajas negras, que pueden ser reusados sin preocuparse de cómo están implementados internamente.

Un servicio web puede tener diferentes niveles de granulación en su estructura interna. En un nivel extremo un Servicio web podría envolver a una aplicación de software

completa, pero en el otro extremo cada función o método podría ser un servicio web. En medio de estos dos extremos un subsistema de software, o un conjunto de clases o una única clase podrían envolverse para conformar a un servicio web. La pregunta es ***¿Cuál es el nivel de granulación de Servicios Web que propicie un balance entre métricas estáticas como la coherencia, cohesión y acoplamiento, y métricas dinámicas, como el tiempo de respuesta?***

Para responder a esta pregunta se necesita realizar experimentación que arroje conocimiento al respecto. Hasta el momento, no existen estudios experimentales que respondan completamente a esta pregunta, solo se intuye que un servicio web muy grande pudiera tener mucha funcionalidad que a lo mejor no se utilizaría en un contexto de reuso de funcionalidades en nuevas aplicaciones. Estaría fuertemente integrado, con pocas dependencias funcionales y con bajo nivel de acoplamiento. Un servicio web de estas características es difícil de mantener y controlar su evolución, pero su tiempo de respuesta sería mejor.

Por otro lado, un servicio web muy pequeño no sería auto-suficiente, ni auto-contenido por lo que necesitaría de otros servicios para completar ciertas funcionalidades que satisfacen requerimientos de software como metas de valor, y su rendimiento sería menor debido a que, por ejemplo, habría varios canales de comunicación en su arquitectura, lo cual incrementaría el tiempo de respuesta. Adicionalmente, el esfuerzo de integración y/o composición sería sustancial, mayormente si no se contara con herramientas automatizadas para realizar estas tareas.

Este tema de tesis se enfoca hacia realizar una investigación experimental que nos indique el comportamiento de los factores estáticos: coherencia, cohesión y acoplamiento, y el tiempo de respuesta como un factor dinámico, ante diferentes niveles de granulación en la estructura interna de Servicios Web. La importancia del proyecto de tesis radica en que es posible construir Servicios Web a partir del software legado, pero no se sabe cómo agrupar las diferentes partes, módulos, funciones o clases en Servicios Web, por desconocer los diferentes valores que deben de tener los factores de calidad mencionados utilizados en conjunto.

2.7 Alcances y limitaciones

2.7.1 Alcances

- Se realizan pruebas con métricas estáticas y dinámicas en Servicios Web construidos a partir del lenguaje *Java*.
- Se contrastan las métricas de coherencia, cohesión, acoplamiento y el tiempo de respuesta en los diferentes niveles de granulación.

- Se evalúan Servicios Web con tres niveles de granulación diferentes: conjunto de clases, única clase y clases internas.
- Se realizan las pruebas de las métricas estáticas mediante la herramienta de medición desarrollada como parte de este trabajo de tesis.

2.7.2 Limitaciones

- Únicamente se realizaron pruebas en equipos de cómputo con las siguientes características:
  Equipo cliente:
  DESKTOP - OLHI5KT
  Procesador: Intel (R) Core (TM) i7-8550U CPU @ 1.80GHz 1.99GHz
  Memoria RAM: 12.0 GB
  Tipo de sistema: Sistema operativo de 64 bits, procesador x64
  Equipo servidor:
  DESKTOP-IEJE6QP
  Procesador: Intel(R) Core(TM) i3-3110M CPU @ 2.40GHz 2.40GHz
  Memoria RAM: 4.0 GB
  Tipo de sistema: Sistema operativo de 64 bits, procesador x64

- No se realizaron pruebas de tiempo de respuesta con más de 50 peticiones simultáneas a los Servicios Web.

2.8 Resumen del capítulo

En este capítulo se presentó el estado del arte para esta investigación, describiendo algunos trabajos relacionados que fueron realizados en los últimos cinco años. Así mismo, se presentaron trabajos realizados en el Cenidet y que están relacionados con este trabajo. Posteriormente se presentaron: el planteamiento del problema, los objetivos, la justificación, los alcances y las limitaciones. A partir de estos puntos se puede tener un panorama claro de la investigación. Sin embargo, es necesario definir los conceptos teóricos que se utilizan, estos conceptos se presentan en el siguiente capítulo que corresponde al marco teórico de este trabajo de tesis.

Capítulo 3. Marco teórico

En esta sección se presentan los conceptos teóricos que se utilizan en el presente trabajo y que forman parte del fundamento teórico que sustenta la investigación.

3.1 Diseño experimental

3.1.1 Introducción a los diseños experimentales

Ronald A. Fisher, el padre del diseño experimental moderno, observó que los experimentos son "*solo experiencia cuidadosamente planificada de antemano y diseñados para formar una base segura de nuevos conocimientos*". Los experimentos implican la manipulación de una o más variables independientes, seguida de la observación sistemática de los efectos de la manipulación en una o más variables dependientes. El énfasis en la experimentación en los siglos XVI y XVII como una forma de establecer relaciones causales marcó el surgimiento de la ciencia moderna desde sus raíces en la filosofía natural (Hacking, 1983).

Existen diferentes clasificaciones de los experimentos, definidas por diferentes autores. Sin embargo, en este documento se retoma la clasificación que los divide en absolutos y comparativos. En los primeros, el objetivo es la determinación de propiedades absolutas de un conjunto de objetos. Por otra parte, en los experimentos comparativos, el objetivo es establecer comparaciones entre objetos que reciben tratamientos diferentes (Diaz, 2009) .

Los estudios comparativos pueden ser experimentales u observacionales. En los experimentales, el investigador controla, es decir, mantiene constantes o varía deliberadamente aquellos factores que parecen tener mayor influencia en los resultados del fenómeno en estudio. El control de las condiciones experimentales, le permite al científico establecer relaciones causa-efecto entre las variables de respuesta y los factores controlados. A estos también se les llama experimentos diseñados, y son universalmente reconocidos como los métodos explicativos más poderosos en la ciencia (Diaz, 2009).

Un diseño experimental es un plan para asignar unidades experimentales a los niveles de tratamiento y el análisis estadístico asociado con el plan (Garavan & Murphy, 2016). El diseño de un experimento según (Kirk, 2013) involucra cinco actividades interrelacionadas:

1. Formulación de hipótesis estadísticas que sean pertinentes a la hipótesis científica. Una hipótesis estadística es una afirmación sobre: (a) uno o más parámetros de una población, y (b) la forma funcional de una población. Las hipótesis estadísticas son formulaciones comprobables de hipótesis y rara vez son idénticas a las hipótesis científicas.

2. Determinación de las condiciones experimentales (variable independiente) a manipular, la medición (variable dependiente) a registrar, y las condiciones extrañas (variables de ruido) que deben ser controladas.

3. Especificación del número de unidades experimentales requeridas y la población de los cuales serán muestreados.

4. Especificación del procedimiento de asignación al azar para definir las unidades experimentales a los niveles de tratamiento.

5. Determinación del análisis estadístico que se realizará.

Un diseño experimental adecuado conseguirá en muchos casos que los datos obtenidos cumplan con las premisas paramétricas y permitirá la aplicación de pruebas estadísticas poderosas y útiles para la interpretación de resultados. Por otra parte, la variación de un factor incontrolado puede provocar un sesgo o error en los resultados obtenidos en cualquier experimento (Serrano, 2003).

3.1.2 Hipótesis

Una hipótesis es una afirmación sobre el parámetro o parámetros en una o más poblaciones. Una hipótesis nula es una declaración de no diferencia entre los parámetros involucrados. La hipótesis nula nunca puede ser probada por ninguna cantidad finita de experimentación. Dada una hipótesis nula particular sin una hipótesis alternativa específica, el experimentador rechaza o no la hipótesis nula. El término "no rechazar" no implica la aceptación de la hipótesis nula para el caso en el que no se postula una alternativa específica. La evidencia experimental puede llevar al rechazo de la hipótesis nula pero no a su aceptación (Vivanco, 2005).

3.1.3 Muestra

La muestra es un conjunto de observaciones o medias tomadas a partir de una población dada. Es decir, es un subconjunto de la población. La muestra debe lograr una representación adecuada de la población, en la que se reproduzca de la mejor manera los rasgos esenciales de dicha población y de esta manera se pueda evitar un sesgo u error (Zubillaga, 2006). En la Figura 1 se muestra una población, y una muestra que se obtiene por medio de un método de muestreo. Se observa que la muestra es representativa al contener individuos de todos los tipos que se tienen en la población.

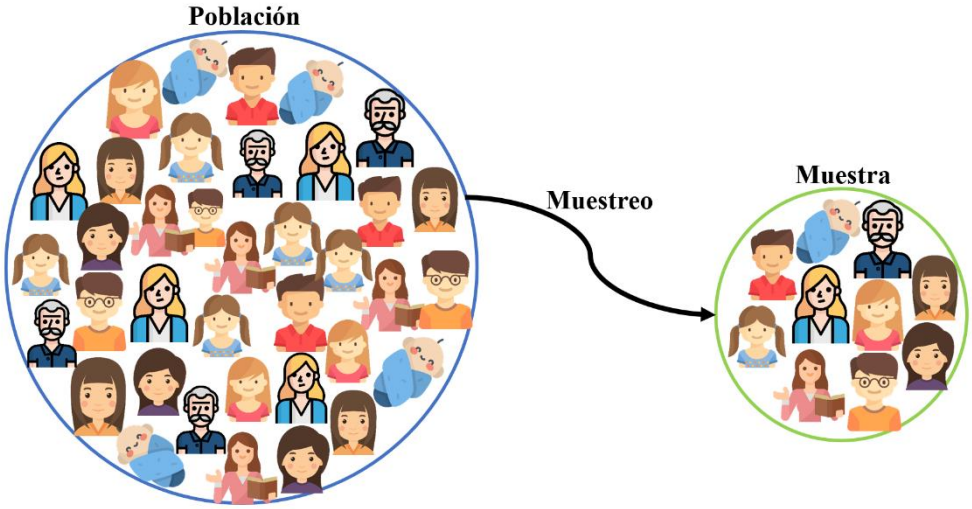

Figura 1. Ejemplo de una muestra representativa

3.1.4 Muestreos probabilísticos

Los muestreos probabilísticos son estrategias de selección de elementos que se sustentan en el principio de selección aleatoria. En la práctica esto significa que todos los elementos de la población tienen una probabilidad conocida y distinta de cero de pertenecer a la muestra. La aleatoriedad no es un atributo de una muestra, sino del proceso de selección utilizado.

Existen diversos procedimientos de muestreo probabilístico. Las diferencias tienen que ver con el modo en que se seleccionan los elementos y con la precisión de las estimaciones. Dentro de los muestreos probabilísticos de uso común se encuentra el Muestreo Aleatorio Simple, que se describe a continuación.

3.1.5 Muestreo Aleatorio Simple

El Muestreo Aleatorio Simple (MAS) es un procedimiento de muestreo básico y fundamentado de otras estrategias de selección de muestra. Se caracteriza porque la selección se realiza de un listado de la población asignándole igual probabilidad a cada elemento. Además, cada muestra de tamaño *n* tiene igual probabilidad de seleccionarse. Es el prototipo

del muestreo equiprobable y autoponderado. Además, existe el muestreo con reposición y sin reposición. En el muestreo sin reposición una vez seleccionado un elemento no se puede seleccionar nuevamente. En el muestreo con reposición un elemento seleccionado podría seleccionarse posteriormente (Vivanco, 2005).

3.1.6 Tamaño de la muestra

Uno de los problemas del muestreo probabilístico es la determinación del tamaño de la muestra, ya que el objetivo primordial al determinarlo es obtener información representativa, válida y confiable para toda la población. Para obtener más exactitud en la información es necesario seleccionar una muestra mayor. Sin embargo, el solo hecho de contar con una muestra grande no garantiza su representatividad (Rodríguez, 2005). El tamaño de la muestra está relacionado con los objetivos del estudio y las características de la población, además de los recursos y del tiempo de que se dispone. El tamaño absoluto de la muestra y sus varianzas son los que ejercen mayor influencia en el error estándar. El tamaño de la muestra se puede determinar con base en la fórmula para estimar la varianza.

A continuación se puntualizan las etapas para determinar el tamaño de la muestra en el muestreo aleatorio simple (Rodríguez, 2005).

a) Determinar el nivel de confianza con el que se desea trabajar. (Al 66, 95 0 99% de confianza). Es más usual al 95%.
b) Estimar las características del fenómeno investigado. Para ello se determina la probabilidad de que se realice el evento (p) o la de que no se realice (q); cuando no se posea suficiente información de la probabilidad del evento, se le asignan los máximos valores. P = 0.5 q=0.5. Cabe recalcar que la suma de p + q siempre debe ser igual a uno.
c) Determinar el grado de error máximo aceptable en los resultados de la investigación. Este puede ser hasta del 10%, normalmente lo más aconsejable es trabajar con variaciones del 1% al 6%, ya que las variaciones mayores del 10% reducen demasiado la validez de la información.
d) Se aplica la fórmula del tamaño de la muestra de acuerdo con el tipo de población.
    - Infinita. Cuando no se sabe el número exacto de unidades del que está compuesta la población.
    - Finita. Cuando se conoce cuantos elementos tiene la población.

Fórmula para calcular el tamaño de la muestra en poblaciones infinitas.

$$\boldsymbol{n = \frac{Z^2 pq}{e^2}}$$

En donde:

n = Tamaño de la muestra.
e^2= Error en la estimación al cuadrado.
p = Probabilidad de éxito.
q = Probabilidad de fracaso.
Z^2 = Valor de la distribución normal de Z de acuerdo al nivel de confianza, al cuadrado.

El error de estimación se utiliza para estimar la precisión necesaria y para determinar el tamaño de muestra más adecuado.

A continuación, se expone un conjunto de conceptos relacionados con métricas de software, que son otro de los aspectos importantes que se retoman en este trabajo de tesis.

3.2 Métricas de calidad de software

Dado que el desarrollo de software es una actividad centrada en el ser humano, es propenso a defectos de diseño y rendimiento no deseados. Por lo tanto, el proceso de desarrollo de software debe evaluarse continuamente y controlar la evolución del software a lo largo del tiempo, para cumplir los requisitos del cliente y eliminar otros defectos identificados (Fuggetta, 2000). Esto ayuda a mejorar el diseño del software y por lo tanto su calidad.

3.2.1. Métricas estáticas

Estas métricas se centran en las propiedades del software inmutables en el tiempo. Las métricas estáticas pueden cuantificar varios aspectos de la complejidad del diseño o del código fuente del sistema de software (Chhabra & Gupta, 2010). En la literatura se ha propuesto un conjunto de métricas estáticas para la medición de acoplamiento, cohesión y otros atributos de software orientado a objetos, utilizando código fuente o desde el modelado del software. Para este trabajo se utilizaron tres métricas estáticas; coherencia, cohesión y acoplamiento, mismas que se describen a continuación.

3.2.1.1 Coherencia

Tradicionalmente, se considera que la cohesión de un componente de software es una medida de similitud de sus partes constituyentes. Sin embargo, varios autores han adoptado también un enfoque diferente: a saber, que la cohesión debe medir qué tan bien contribuyen los miembros de un conjunto de objetos dado, a un propósito u objetivo común. En (Mišić, 2000) se propone una medida de cohesión genérica basada en el segundo enfoque, a la que Mišić llamó coherencia, y que enfatiza su utilización en el contexto del segundo enfoque. Así pues, la coherencia de un conjunto de objetos mide qué tan bien contribuyen sus miembros a un propósito u objetivo común, definido externamente.

Un sistema consiste en una colección de objetos, que describen varias partes del sistema en diferentes niveles de abstracción. Cada objeto tiene una parte externa y otra interna. La parte interna implementa la estructura y el comportamiento de un objeto. En la mayoría de los casos, la implementación de un objeto es en realidad una colección (posiblemente ordenada) de otros objetos que a su vez se implementan como colecciones de otros objetos, y así sucesivamente.

Para propósitos de esta tesis se retomó el concepto de coherencia, considerando a los métodos como los miembros que contribuyen a un objetivo común. A continuación, se presenta la métrica formulada de acuerdo a lo anterior.

Métrica CrCU (Coherencia de caso de uso)

Esta métrica mide la cantidad de métodos que interactúan para cumplir un objetivo, en relación al número total de funciones de clases contenidas en un módulo, y se refiere a una secuencia interactiva de métodos implicados para cumplir con un caso de uso (Gallardo, 2018). En la Ecuación 1 se presenta la expresión matemática para esta métrica. Un valor de cero o cercano a cero significa que el módulo carece de coherencia y, por lo tanto, no todos los métodos están contribuyendo. Un valor que tiende a uno significa mayor coherencia, es decir que todos los métodos están contribuyendo al cumplimiento del caso de uso.

$$CrCU = \frac{n}{m} \qquad \text{(Ecuación 1)}$$

En donde:

n = Total de métodos en una secuencia interactiva que define el caso de uso.
m = Total de métodos del módulo o paquete.

3.2.1.2 Cohesión

En este apartado se retoma el primero de los dos enfoques para la medición de cohesión que se mencionan en (Mišić, 2000), este enfoque considera el término cohesión como una medida de similitud de las partes constituyentes de un artefacto de software; que en consecuencia, es una propiedad interna de ese artefacto.

El término cohesión tiene su origen en el diseño estructural y se refiere a la relación entre los atributos y métodos de clases contenidas en un módulo dado. Es un indicador importante de la calidad del diseño de software y de la modularidad. Un valor de cohesión más alto de un módulo indica que este proporciona una funcionalidad casi única, mientras que un valor inferior impide el reuso de un módulo de software (Rathee & Chhabra, 2018). A continuación, en la Figura 2 se muestra un ejemplo de una clase altamente cohesiva. La

ClaseA tiene atributos y funciones, estas funciones están altamente relacionadas al compartir todos los atributos de la clase.

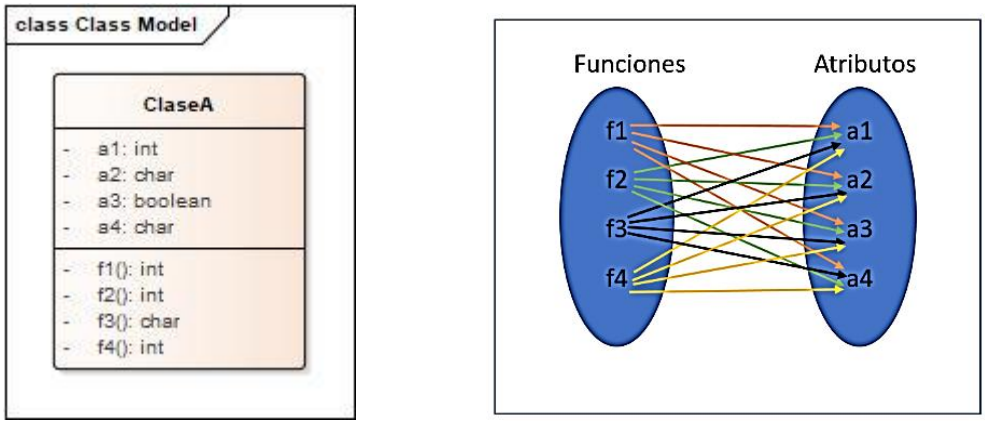

Figura 2. Ejemplo de alto grado de cohesión en una clase.

Existen distintas métricas para la medición de cohesión. Sin embargo, para efectos de esta tesis se eligió la métrica LCOM* debido a que, con base en sus características, se determinó que es la adecuada para las pruebas que se requieren en el experimento.

Métrica LCOM* (Carencia de cohesión en los métodos)

LCOM es una medida de la carencia de cohesión de una clase, mide el número de atributos comunes usados por diferentes métodos e indica la calidad de la abstracción hecha en la clase. En la ecuación 2 se muestra la expresión matemática que representa esta métrica.

$$LCOM* = \frac{\left(\frac{1}{a}\sum_{j-1}^{a}\mu(A_j)\right) - m}{1 - m} \qquad \text{(Ecuación 2)}$$

Esta métrica considera un grupo de métodos {Mi} (i=1,…, m) accediendo a un conjunto de atributos {Aj} (j=1,…, a) y el número de métodos que acceden a cada atributo como μ(Aj). Esta métrica solo puede calcularse cuando m>1. Cuando todos los métodos acceden a todos los atributos, entonces Σμ(Aj) = ma, y por lo tanto LCOM* = 0 (Henderson-Sellers, 1996).

3.2.1.3 Acoplamiento

Dos objetos se acoplan si y solo si al menos uno de ellos actúa sobre el otro. Dado que el acoplamiento es el grado de interacción entre clases, la idea básica que subyace a todas las métricas de acoplamiento es muy simple: se cuenta el número de interacciones entre clases que hay en el sistema. Sin embargo, hay una variación considerable en función de lo que cuenta como una interacción, es decir, cómo se realiza el conteo y cómo se normalizan los totales (Fenton & Ohlsson, 2000).

Con respecto a esta medición, se decidió utilizar la métrica COF que es una de las más utilizadas para medir el acoplamiento en arquitecturas de software.

Métrica COF (Factor de acoplamiento)

Esta métrica se propuso como una medida de acoplamiento entre las clases, excluyendo el acoplamiento debido a la herencia. Se calcula considerando todos los posibles pares de clases y consultando si las clases en el par están relacionadas, ya sea por transmisión de mensajes o por enlaces de asociación semántica (referencias por una clase a un atributo o método de otra clase) (Brito e Abreu & Melo, 2002). En la Ecuación 3 se presenta la expresión matemática correspondiente a esta métrica.

$$COF = \frac{\sum_{i=1}^{TC}\left|\sum_{j=1}^{TC} es\_cliente(C_i\, C_j)\right|}{TC^2 - TC} \qquad \text{(Ecuación 3)}$$

En donde:

TC = Total de clases.

$$es\_cliente = \begin{cases} 1 \Leftrightarrow C_C \Rightarrow C_S \wedge C_C \neq C_S \\ 0 \qquad\qquad caso\ contrario \end{cases} \qquad \text{(Ecuación 4)}$$

En la Ecuación 4 se muestra la relación cliente-servidor, en donde $C_C \Rightarrow C_S$ significa que C_C (clase de cliente) contiene al menos una referencia de no herencia a una característica (método o atributo) de la clase C_S (clase servidora).

3.2.2. Métricas dinámicas

Las métricas dinámicas son la clase de métricas de software que capturan el comportamiento mutable en el tiempo del sistema de software. Generalmente se obtienen de los rastros de ejecución del código o de los modelos ejecutables (Chhabra & Gupta, 2010).

Tiempo de respuesta

Existen definiciones de tiempo de respuesta que pueden variar en cuanto a parámetros y ciertas características. Para este trabajo se realizó la medición del tiempo de respuesta considerando las siguientes definiciones.

El tiempo máximo garantizado (promedio o mínimo) requerido para completar una solicitud de servicio (Gunther, 1998).

El tiempo necesario para enviar una solicitud y recibir una respuesta (Al-Masri & Mahmoud, 2007).

A continuación, se presenta una breve descripción de los servicios web, que son las entidades de software con las que se realizó la experimentación.

3.3 Servicios Web

Un servicio web es cualquier servicio disponible a través de internet, utiliza un sistema de mensajería XML estandarizado y no está vinculado específicamente a algún sistema operativo o lenguaje de programación (Cerami, 2002). Un servicio web debe ser auto-descriptivo. Si se publica un nuevo servicio, también se debe publicar su interfaz pública. Así también, debe incluir documentación legible para que otros desarrolladores puedan integrarlo fácilmente. Otra característica es que debe ser detectable, si se crea un servicio web, debería haber un mecanismo relativamente simple para que se publique el nuevo servicio. Hay tres elementos principales dentro de la arquitectura del servicio web (Cerami, 2002):

Proveedor de servicio: El proveedor de servicios implementa el servicio y lo pone a disposición en internet.

Solicitante de servicio: El solicitante utiliza un servicio web existente al abrir una conexión de red y al enviar una solicitud XML.

Registro de servicio: Este es un directorio de servicios centralizado lógicamente. El registro proporciona un lugar central donde los desarrolladores pueden publicar nuevos servicios o encontrar los existentes. Por lo tanto, sirve como un centro de información centralizado para las empresas y sus servicios.

A continuación, se presentan algunos de los aspectos que se utilizan en el paradigma orientado a objetos. Adicionalmente, se describen algunos conceptos relacionados con las arquitecturas de software y que se utilizan en el contexto de este trabajo.

3.4 Arquitecturas de software

A medida que aumenta el tamaño y la complejidad de los sistemas de software, el problema de diseño va más allá de los algoritmos y las estructuras de datos de la computación: el diseño y la especificación de la estructura general del sistema han emergido como nuevos tipos de problemas.

Existe una gran cantidad de trabajo en el tema de diseño de la arquitectura de software, que incluye lenguajes de interconexión de módulos, plantillas y marcos para sistemas que satisfacen las necesidades de dominios específicos, y modelos formales de mecanismos de integración de componentes. Aunque actualmente no existe una terminología o notación bien definida para caracterizar las estructuras arquitectónicas, los buenos ingenieros de software

hacen uso común de los principios arquitectónicos al diseñar software complejo. Muchos de los principios representan reglas prácticas, paradigmas o patrones que han surgido de manera informal a lo largo del tiempo. Otros están más cuidadosamente documentados como estándares industriales y científicos (Garlan & Shaw, 2008).

3.5 Diseño orientado a objetos

El diseño orientado a objetos proporciona una manera natural e intuitiva de ver el proceso de diseño de software: a saber, modelando las entidades de software tales como clases de objetos, atributos, operaciones, y sus relaciones. Este paradigma de programación proporciona la característica de encapsulamiento de atributos y operaciones; los atributos y las operaciones de un objeto se enlazan entre sí. Los objetos tienen la propiedad de ocultamiento de información. Esto significa que los objetos pueden saber cómo comunicarse entre sí a través de interfaces bien definidas, pero por lo general no se les permite saber cómo se implementan otros objetos; los detalles de la implementación están ocultos.

3.5.1 Clases y Objetos

Un objeto es una unidad de instanciación encapsulada con identidad única, que tiene un estado implícito (atributos), el cual puede ser persistente, y operaciones funcionales (comportamiento). Los objetos pueden ser organizados en categorías o clases. Es decir, una clase describe, en forma abstracta, todos los objetos de un tipo en particular. Los objetos se crean a partir de clases y la clase describe la categoría del objeto (Barnes & Kölling, 2007).

3.5.2 Clases internas

Las clases internas son clases definidas dentro de otras clases y sus instancias siempre están ligadas a una instancia de la clase contenedora. En general, las clases internas son exclusivamente de uso interno de la clase, por lo que casi nunca son públicas.

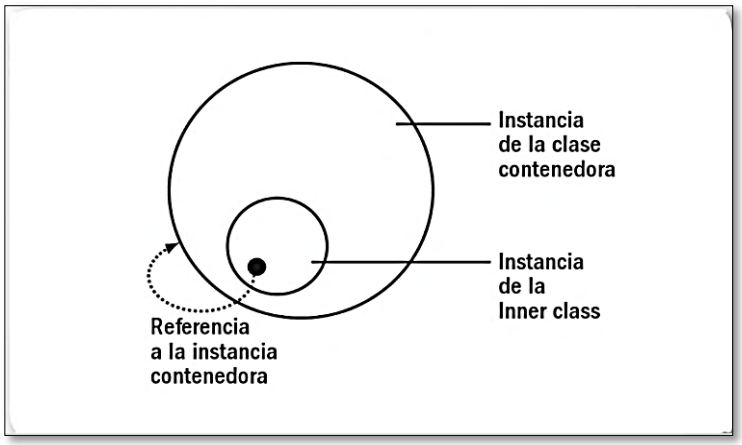

Figura 3. Relación entre una instancia de una clase interna y una instancia contenedora (Vivona, 2011).

Antes de poder crear un objeto de una clase interna, debe haber primero un objeto de la clase de nivel superior que contenga a la clase interna. Esto se requiere debido a que un objeto de la clase interna tiene implícitamente una referencia al objeto de su clase de nivel superior. También hay una relación especial entre estos objetos: el objeto de la clase interna puede acceder directamente a todas las variables de instancia y métodos de la clase externa, aun cuando sean calificadas como privadas (Deitel & Deitel, 2008).

3.6 Componentes de software

Un componente de software es una unidad de composición, de parte de terceros, que puede ser desarrollado, adquirido y utilizado independientemente, y que define las interfaces por medio de las cuáles puede ser ensamblado con otros componentes para proveer y usar servicios. (Clements, 1995).

3.7 Reuso de software

El reuso de software se refiere al uso de métodos y demás elementos de software existentes como bloques de construcción para crear nuevos programas (Deitel & Deitel, 2008). El reusar presenta ciertas ventajas dentro de las que se encuentran: una mayor confiabilidad, es decir, que el software sea robusto, verificado y probado; mayor fiabilidad debido a la alta probabilidad de operación libre de fallas durante un tiempo específico; reducción de costos por mantenimiento de software; entre otras.

3.8 Resumen del capítulo

En este capítulo se describieron los conceptos más importantes que se utilizan en este trabajo de tesis, primeramente, se describieron los diseños experimentales debido a que la experimentación es la actividad central de la tesis, así mismo se describieron las métricas de calidad de software con las cuales se realiza la experimentación, y por último se incluyen aspectos relacionados con las arquitecturas de software que se consideraron. Con base en esta información, en el siguiente capítulo se presenta el diseño experimental, en donde se definen las características y condiciones bajo la cuales se realiza el experimento.

Capítulo 4. Diseño experimental

4.1 Formulación de preguntas e hipótesis

4.1.1 Hipótesis general

El nivel de granulación, en función de la estructura interna de Servicios Web, influye en el tiempo de respuesta y en sus atributos de calidad como entidades de software.

4.1.2. Hipótesis específica

El número de canales de comunicación entre unidades de programa sobrecargan el sistema operativo durante la ejecución de la aplicación de software, debido a que este almacena y recupera el bloque de contexto del procesador por cada petición funcional en una secuencia interactiva, por lo tanto, esta sobrecarga influye en el tiempo de respuesta del servicio.

4.1.3. Hipótesis estadísticas

A continuación, se presentan las hipótesis estadísticas que se definieron para el experimento.

Dado que:

TRuc es el tiempo de respuesta de un servicio web cuya arquitectura está organizada en una única clase.

TRcc es el tiempo de respuesta de un servicio web cuya arquitectura está organizada en un conjunto de clases relacionadas.

TRci es el tiempo de respuesta de un servicio web cuya arquitectura está organizada en un conjunto de clases internas anidadas.

COFuc es el acoplamiento de un servicio web cuya arquitectura está organizada en una única clase.

COFcc es el acoplamiento de un servicio web cuya arquitectura está organizada en un conjunto de clases relacionadas.

COFci es el acoplamiento de un servicio web cuya arquitectura está organizada en un conjunto de clases internas anidadas.

CrCUuc es la coherencia de un servicio web cuya arquitectura está organizada en una única clase.

CrCUcc es la coherencia de un servicio web cuya arquitectura está organizada en un conjunto de clases relacionadas.

CrCUci es la coherencia de un servicio web cuya arquitectura está organizada en un conjunto de clases internas anidadas.

LCOMuc es la carencia de cohesión de un servicio web cuya arquitectura está organizada en una única clase.

LCOMcc es la carencia de cohesión de un servicio web cuya arquitectura está organizada en un conjunto de clases relacionadas.

LCOMci es la carencia de cohesión de un servicio web cuya arquitectura está organizada en un conjunto de clases internas anidadas.

Donde:

0 <= TR <=∞, el mejor tiempo de respuesta es aquel que tiende a cero.

0 <= COF <=1, el valor óptimo de acoplamiento es cero, el peor valor es 1.

0 <= LCOM <=1, el valor óptimo de LCOM es 0, el peor valor de LCOM es 1.

0 <= CrCU <=1, el valor óptimo de coherencia es 1, el peor valor de coherencia es 0.

Se definieron las siguientes hipótesis

Hipótesis para la experimentación con Servicios Web con granularidad de clase única.

Hipótesis nula (H_0):

$$(TRuc < (TRcc \vee TRci)) \wedge (COFuc < (COFcc \wedge COFci)) \wedge (CrCUuc = (CrCUcc \vee CrCUci)) \wedge (LCOMu > (LCOMcc \vee LCOMci))$$

Hipótesis alternativa (H_1):

$$(TRuc < (TRcc \vee TRci)) \wedge (COFuc < (COFcc \vee COFci)) \wedge (CrCUuc > (CrCUcc \vee CrCUci)) \wedge (LCOMu > (LCOMcc \vee LCOMci))$$

Hipótesis para la experimentación con Servicios Web con granularidad de conjunto de clases relacionadas.

Hipótesis nula (H_0):

$$(COFcc > (COFuc \vee COFci)) \wedge (TRcc > (TRuc \vee TRci))$$

Hipótesis alternativa (H_1):

$$(COFcc < (COFuc \vee COFci)) \wedge (TRcc < (TRcc \vee TRci))$$

4.2 Determinación de las condiciones experimentales

Una variable experimental es un atributo que al ser medido en diferentes situaciones es susceptible de adoptar distintos valores. Para el presente estudio experimental, se identificaron tres tipos de variables: las variables dependientes, las variables independientes y las variables intervinientes o factores de ruido.

4.2.1. Variables dependientes

Son las variables de respuesta que se observan en el estudio experimental y que podrían estar influenciadas por los valores de las variables de entrada. Se espera que cambien o difieran como resultado de la aplicación del tratamiento (Pfleeger, 1995). Para el presente estudio experimental se definieron las siguientes variables dependientes:

Tiempo de respuesta. El tiempo máximo garantizado (promedio o mínimo) requerido para completar una solicitud de servicio (Gunther, 1998) . Con base en la medición de tiempos de respuesta de los Servicios Web, se puede determinar en qué nivel de granulación se obtiene el mejor tiempo de respuesta.

Coherencia: El grado de coherencia se mide a partir de las características de la estructura interna de los Servicios Web. Los valores para la métrica que se utiliza en este trabajo están entre cero y uno, en donde el valor óptimo es uno y el peor valor es cero.

Cohesión: En una arquitectura de software es deseable un alto grado de cohesión. En este trabajo se utiliza una métrica que mide la cohesión en términos de su carencia, por lo cual los valores deseables son los más cercanos a cero y el valor óptimo es cero.

Acoplamiento: El valor de acoplamiento se mide a partir de la dependencia que existe entre los elementos de una arquitectura de software. Los valores de la métrica que se utiliza en este trabajo están entre uno y cero, en donde el valor óptimo es cero y el peor valor es uno.

4.2.2. Variables independientes

Las variables de estado o las variables independientes son aquellas variables que pueden influir en la aplicación de un tratamiento y, por lo tanto, indirectamente en el resultado del experimento (Pfleeger, 1995). Son las variables que pueden cambiar libremente su valor y que no dependen de otra variable. Estas variables son manipuladas en valor para inducir un cambio en las variables dependientes, las cuales son observadas en el experimento.

Para el presente estudio experimental se definió una única variable independiente:

Nivel de granulación: Determinado por la estructura interna de los Servicios Web utilizados como pruebas. Los casos de prueba fueron diseñados en función de tres niveles de granulación las cuales son: única clase, conjunto de clases relacionadas y clases internas.

4.2.3. Variables intervinientes o factores de ruido

Para realizar un experimento robusto y confiable, es necesario aislar todos los factores externos que pudieran ocasionar ruido experimental y sesgar los resultados. Algunos de los factores de ruido son los siguientes:

Factores controlables: Entre los factores controlables se encuentran las especificaciones técnicas de los equipos de cómputo, la gestión de procesos en segundo plano, la configuración del servicio de internet, el acceso de equipos a la red local y la configuración de la misma.

Factores no controlables: En este grupo se encuentran los factores de gestión de la memoria, el funcionamiento interno de la máquina virtual de Java, la forma en que el servidor internamente administra las peticiones.

4.3 Especificación del número de unidades experimentales requeridas y la población de la cual serán muestreadas

4.3.1 Población

La población consiste en un conjunto de Servicios Web desarrollados en lenguaje Java, a los que se tuvo acceso libre, ya sea porque son de código libre, de un repositorio de servicios o bien de proyectos escolares.

4.3.2 Muestra

Se utiliza la técnica del Muestreo Aleatorio Simple para seleccionar las unidades experimentales. A continuación, se describen los pasos que se realizaron para determinar el tamaño de la muestra.

1.- Se determinó el nivel de confianza del 90% para el experimento, por lo cual el valor de Z correspondiente en la tabla de Z es de 1.645.

2.- De conformidad al grado de conocimiento o experiencia del responsable del experimento en cuanto al comportamiento del fenómeno en estudio, se estimaron los siguientes valores para p y q.

Probabilidad de éxito (p): 0.7

Probabilidad de fracaso (q): 0.3

3.- Se determinó el grado de error máximo aceptable en los resultados de la investigación. Se determinó un grado de error al 10% siendo un valor aconsejable de acuerdo a la literatura, y tomando en cuenta que las variaciones mayores al 10% reducen demasiado la validez de la información.

4.- Se aplicó la fórmula del tamaño de la muestra de acuerdo con el tipo de población, que en este caso es infinita.

Fórmula:

$$n = \frac{Z^2 pq}{e^2}$$

Sustitución:

$$n = \frac{1.645^2 (0.7)(0.3)}{0.10^2}$$

$$n = 56.82$$

4.4 Especificación del procedimiento de asignación al azar para definir las unidades experimentales a los niveles de tratamiento

La selección de las unidades se realizó de un listado de la población asignándole la misma probabilidad a cada uno. Además, la elección se realizó sin reposición debido a que una vez seleccionado un elemento, no se puede seleccionar nuevamente.

4.5 Determinación del análisis estadístico para el tratamiento de los datos

El análisis estadístico que se determinó realizar consiste en el uso de herramientas de la estadística descriptiva como histogramas y polígonos de frecuencia. Así también tablas descriptivas que permiten mostrar las características principales de los datos obtenidos.

A partir del análisis del comportamiento de los datos obtenidos, se obtiene la información necesaria que permite tomar una decisión con respecto a la aceptación o rechazo de las hipótesis establecidas para el experimento.

A continuación, se describen las pruebas que se realizan para la experimentación en este trabajo de tesis.

4.6 Diseño de pruebas

Se consideran sistemas completos orientados a objetos o marcos de aplicaciones orientados a objetos que atienden dominios específicos existentes. A estos sistemas se les aplican los algoritmos estratégicos del sistema *SR2 transforming*, desarrollado en investigaciones anteriores, para fragmentarlos en unidades separadas o Servicios Web, en los cuales cada uno atiende a un requerimiento o capacidad como meta de valor. La estructura interna de cada uno de los Servicios Web obtenidos se configura en los tres niveles de granulación que se asignaron a la variable independiente. Una vez obtenidos los Servicios Web de prueba, se realiza el cálculo de las métricas para medir su cohesión, coherencia, acoplamiento y tiempo de respuesta. Adicionalmente, se utilizan para las pruebas otros Servicios Web previamente desarrollados. Estos Servicios Web también se configuran en su estructura interna en los tres niveles de granulación que se asignaron a la variable independiente.

Los resultados obtenidos se interpretan con estadística descriptiva para descubrir información que permita comprobar las hipótesis planteadas. Así mismo, se deducen aspectos que se pueden presentar en el fenómeno y que no son evidentes a simple vista, los cuales pueden contribuir a generar nuevos conocimientos acerca del fenómeno de estudio.

- Las pruebas se realizan con 57 Servicios Web, cada uno con tres diferentes niveles de granulación (clase única, conjunto de clases relacionadas y clases internas).

- Las pruebas experimentales incluyen dos tipos de pruebas, pruebas de atributos estáticos de calidad y pruebas de atributos dinámicos.

    - Pruebas de atributos estáticos de calidad. Se consideran tres métricas estáticas: coherencia, cohesión y acoplamiento. Para la medición se utiliza un marco orientado a objetos que implementa estas métricas, el cual forma parte de este trabajo de tesis.

Entradas: Conjunto de clases que forman un servicio web en los tres niveles de granulación.

Salida: Valores de coherencia, acoplamiento y cohesión en cada uno de los tres niveles de granulación.

- Pruebas de atributos dinámicos. Para estas pruebas se utiliza una herramienta disponible en la red, que sirve para medir dinámicamente el tiempo de respuesta.

Entradas: Conjunto de clases que forman un servicio web en los tres niveles de granulación.

Salidas: Tiempo de respuesta de los Servicios Web en cada uno de los niveles de granulación.

- Los resultados obtenidos de cada prueba se documentan de manera ordenada y se someten a un análisis estadístico de resultados.

4.7 Resumen del capítulo

En este capítulo se plantearon las hipótesis de acuerdo a lo que se quiere comprobar, y se calcularon los valores para los parámetros estadísticos requeridos. Así mismo, se establecieron las variables y las condiciones bajo las cuales se realiza el experimento. Por último, se presentó la descripción de las pruebas, el tipo de prueba, así como las entradas y salidas para cada una. Contando con esta información, se procedió a la ejecución del experimento, proceso que se describe en el siguiente capítulo.

Capítulo 5. Ejecución del experimento

5.1 Desarrollo de una herramienta de software para el cálculo de métricas estáticas

Durante esta actividad se realizó el estudio y análisis del funcionamiento de meta-compiladores que pueden ser utilizados para analizar y extraer información de las entidades de software, debido a que esta información es necesaria para el cálculo de las métricas de coherencia, cohesión y acoplamiento. Se seleccionó el meta-compilador ANTLR (Parr, 2014) y se procedió a desarrollar el software para la medición.

5.1.1 ANTLR

ANTLR (Parr, 2014) es un potente generador de analizadores para leer, procesar, ejecutar o traducir texto binario o texto estructurado. A partir de la descripción de un lenguaje formal específico expresado en sus reglas gramaticales, ANTLR genera un analizador para ese lenguaje, y construye automáticamente árboles de análisis sintáctico que representan cómo una gramática coincide con la entrada. ANTLR también genera analizadores que pueden construir y recorrer árboles de análisis.

5.1.2 Marco Orientado a Objetos para la Medición de Calidad de Arquitecturas de Software

En conjunto con el equipo de trabajo que actualmente utiliza métricas de calidad de software para sus investigaciones en el *Tecnológico Nacional de México/ CENIDET*, se desarrolló un Marco Orientado a Objetos para la Medición de Calidad de Arquitecturas de Software. Como parte de este trabajo de tesis, este marco orientado a objetos fue extendido con la implementación de funciones para el cálculo de las métricas de coherencia, cohesión y acoplamiento.

Con la representación de una gramática en ANTLR, se genera un scanner o lexer con el cual se realiza el reconocimiento del lenguaje a nivel léxico, así mismo ANTLR genera un parser para el reconocimiento a nivel sintáctico. Para efectos de esta tesis se generaron los archivos JavaLexer.g4 y JavaParser.g4 para el reconocimiento del lenguaje *Java*. El parser obtiene componentes léxicos (tokens) a partir de los caracteres de entrada, y pasa una secuencia de tokens como entrada al parser. Este proceso de análisis léxico y sintáctico es utilizado para extraer los datos del programa fuente original que se necesitan para el cálculo de las métricas de calidad mencionadas anteriormente.

A continuación, en la Figura 4 se muestra la arquitectura de clases del marco para la medición de calidad de arquitecturas orientadas a objetos que fue extendido con las métricas implementadas en esta tesis.

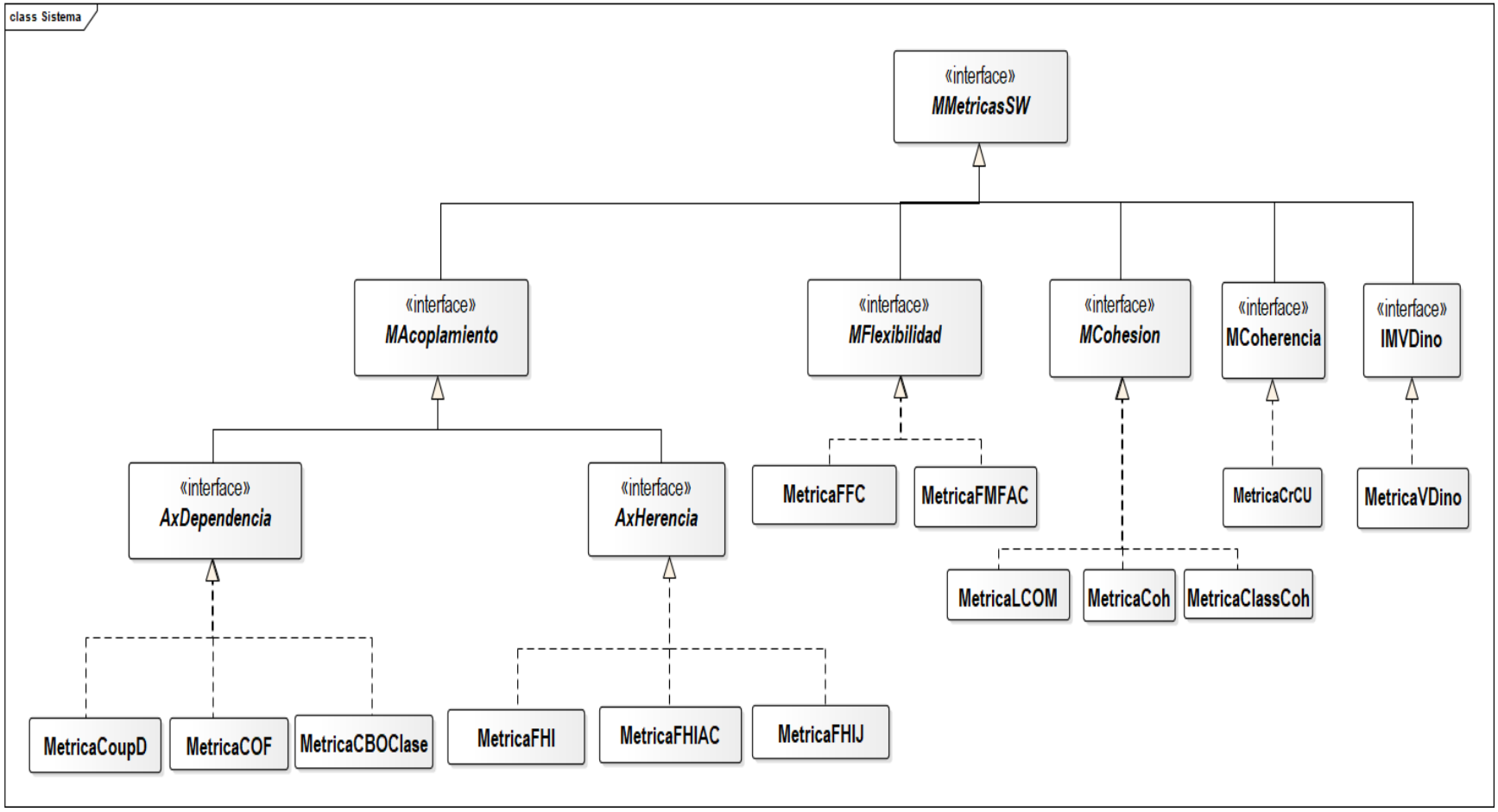

Figura 4. Arquitectura de clases del Marco Orientado a Objetos para la Medición de Calidad en Arquitecturas de Software

La arquitectura se diseñó conforme a los principios de diseño orientado a objetos: abierto/cerrado, inversión de dependencias, sustitución, única responsabilidad y no-repetición. Esta arquitectura garantiza el desarrollo del marco orientado a objetos con las mejores prácticas de la ingeniería de software para efectos de su mantenimiento y control de su evolución en el tiempo.

El funcionamiento de la herramienta de medición consiste básicamente en ingresar los archivos *.java* que conforman la estructura interna de un servicio web. A través del analizador se extraen los datos que se requieren sobre las clases, funciones, atributos, invocación de métodos, entre otros elementos que contiene el código, y con base en esta información la herramienta realiza los cálculos necesarios de acuerdo a la fórmula de la métrica elegida.

5.2 Generación de Servicios Web en diferentes arquitecturas.

Para la realización del experimento planteado, se procedió a modificar la estructura interna de los Servicios Web bajo los tres enfoques trabajados en investigaciones anteriores: única clase (Legorreta, 2017), conjunto de clases relacionadas (León, 2009) y clases internas (Gallardo, 2018). Para el desarrollo de esta actividad, se utilizó el framework Spring Boot y la herramienta STS (Spring Tool Suite) como entorno de desarrollo, en donde se organizó el código de tal forma que cada paquete corresponde a una arquitectura diferente de los Servicios Web. A continuación, en la Figura 5 se muestra la interfaz de la herramienta, así como un ejemplo de la organización del código.

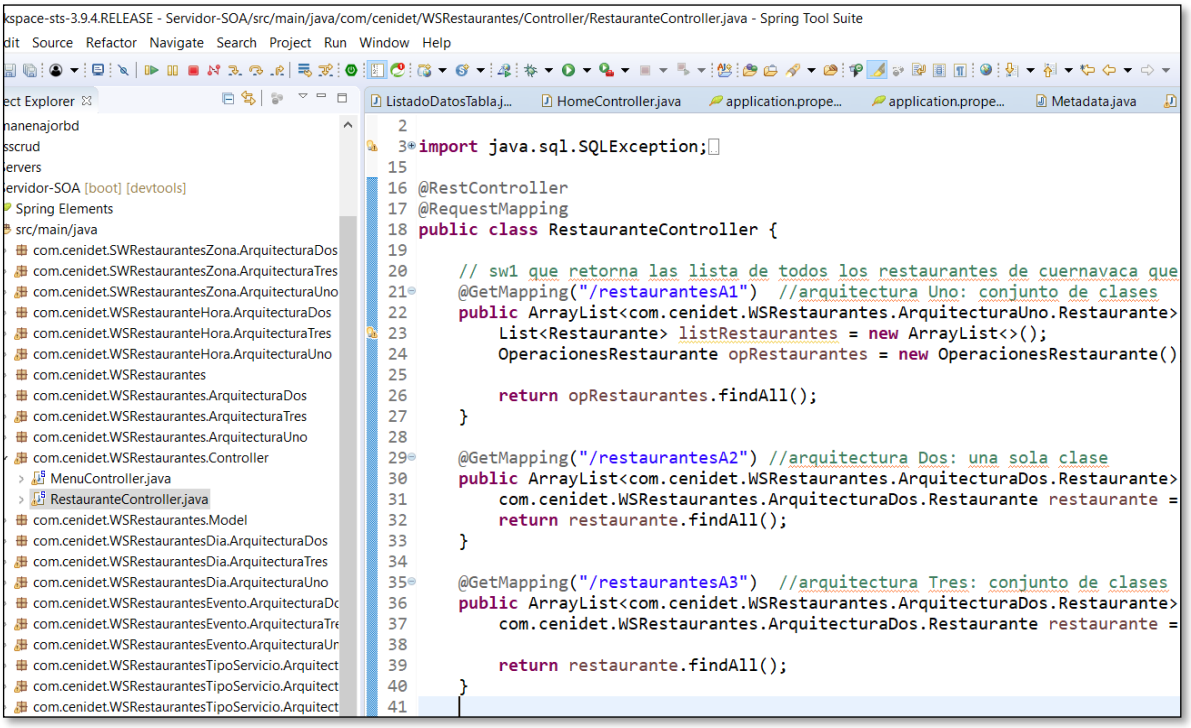

Figura 5. Organización de paquetes y arquitecturas para los Servicios Web

De igual forma, en las Figuras 6, 7 y 8 se presenta un ejemplo del código que conforma cada una de las tres arquitecturas, las cuales son tres implementaciones encargadas de realizar la misma funcionalidad, únicamente cambia su estructura. Así pues, tomando en cuenta que para cada servicio se generan tres arquitecturas, y que de acuerdo al diseño experimental se requerían 57 Servicios Web diferentes, se obtuvieron un total de 171 servicios para realizar las pruebas. Cabe mencionar que estos servicios están enfocados hacia diferentes funcionalidades, dentro de las cuales se incluyen, operaciones matemáticas, operaciones con cadenas, métodos de ordenamiento, operaciones con bases de datos, entre otras.

```
public class CalcDesviacionEstandar {
     double prom, sum;
     int i, n;
     double valorDesviacion;
     double suma;

      public  double calculaDesviacion ( double [ ] array ) {
            sum = 0;
            n = array.length;
            valorDesviacion=0;
            suma =0;
            for(int j=0; j<array.length; j++) {
                suma = suma + array[j];
            }
            prom = suma/array.length;

            for ( i = 0; i < n; i++ ) {
                sum += Math.pow(array [ i ] - prom, 2);
            }
            valorDesviacion = (double) Math.sqrt(sum / ( double ) n );

            return valorDesviacion;
          }

}
```

Figura 6. Código generado bajo el enfoque de única clase

```java
public class RaizCuadrada {

    double raizCuadrada;

    public double calculaRaizCuadrada(double num) {
        raizCuadrada=0;
        raizCuadrada = (double) Math.sqrt(num);
        return raizCuadrada;
    }


}
```

```java
public class DesviacionEstandar {

    double prom, sum ;
    int i, n;
    double valorDesviacion;
    Media promedio;
    CalculaCuadrado cuadrado;
    RaizCuadrada raiz;
     public  double calculaDesviacion ( double [ ] array ) {
           prom =0;
           sum = 0;
           n = array.length;
           valorDesviacion=0;
           promedio = new Media();
           cuadrado = new CalculaCuadrado();
           raiz = new RaizCuadrada();
           prom = promedio.calculaMedia(array);

           for ( i = 0; i < n; i++ ) {
               sum += cuadrado.elevaAlCuadrado(array [ i ] - prom);
           }

           valorDesviacion = raiz.calculaRaizCuadrada(sum / ( double ) n );

           return valorDesviacion;
       }
}
```

```java
public class CalculaCuadrado {
    double cuadrado;
    public double elevaAlCuadrado(double num) {
        cuadrado=0;
        cuadrado = Math.pow(num, 2);
        return cuadrado;
    }
}
```

```java
public class Media {
    double promedio;
    double suma;
    int i;

    public double calculaMedia( double[] array) {
        promedio = 0;
        suma = 0;

        for(i=0; i<array.length; i++) {
            suma = suma + array[i];
        }

        promedio = suma/array.length;
        return promedio;
    }

}
```

Figura 7. Código generado bajo el enfoque de conjunto de clases

```
3
4  public class CalculaDesviacionEstandar {
5       double prom, sum;
6       int i, n;
7       double valorDesviacion;
8       Media promedio;
9       CalculaCuadrado cuadrado;
10      RaizCuadrada raiz;
11       public  double calculaDesviacion ( double [ ] array ) {
12             sum = 0;
13             n = array.length;
14             valorDesviacion=0;
15             promedio = this.new Media();
16             cuadrado =this.new CalculaCuadrado();
17             raiz = this.new RaizCuadrada();
18             prom = promedio.calculaMedia(array);
19
20             for ( i = 0; i < n; i++ ) {
21                 sum += cuadrado.elevaAlCuadrado(array [ i ] - prom);
22             }
23
24             valorDesviacion = raiz.calculaRaizCuadrada(sum / ( double ) n );
25
26             return valorDesviacion;
27           }
28
29       private class CalculaCuadrado {
30           double cuadrado;
31             public double elevaAlCuadrado(double num) {
32                 cuadrado=0;
33                 cuadrado = Math.pow(num, 2);
34                 return cuadrado;
35             }
36         }
37
38       private class Media {
39           double promedio;
40           double suma;
41             public double calculaMedia( double[] array) {
42                 promedio=0;
43                 suma =0;
44                 for(int i=0; i<array.length; i++) {
45                     suma = suma + array[i];
46                 }
47                 promedio = suma/array.length;
48                 return promedio;
49             }
50
51         }
52
53       private class RaizCuadrada {
54           double raizCuadrada;
55             public double calculaRaizCuadrada(double num) {
56                 raizCuadrada=0;
57                 raizCuadrada = (double) Math.sqrt(num);
58                 return raizCuadrada;
59             }
60
61
62         }
63
64 }
```

Figura 8. Código generado bajo el enfoque de clases internas

A continuación, en la Tabla 2 se muestran los nombres de los 57 Servicios Web que se utilizaron para las pruebas.

Tabla 2. Servicios Web que forman parte del conjunto de unidades de prueba para el experimento

#Sw	Servicio web	#Sw	Servicio web
sw1	Calcula acoplamiento	**sw30**	Calcula valor máximo y valor mínimo
sw2	Calcula coherencia	**sw31**	Calcula media artimética
sw3	Calcula cohesión	**sw32**	Calcula medidas de un cilindro
sw4	Consulta ciudades de México	**sw33**	Calcula medidas de un cono
sw5	Consulta hoteles de México	**sw34**	Calcula medidas de un cubo
sw6	Genera nombres de persona I	**sw35**	Convierte números decimales a binarios I
sw7	Inserción y generación de nombres de personas I	**sw36**	Convierte números decimales a binarios II
sw8	Consulta de nombres de personas	**sw37**	Identifica números primos I
sw9	Genera nombres de persona II	**sw38**	Identifica números primos II
sw10	Inserción y generación de nombres de personas II	**sw39**	Operaciones aritméticas (varios números)
sw11	Consulta de nombres por apellidos	**sw40**	Operaciones aritméticas (dos números)
sw12	Consulta de restaurantes de una ciudad	**sw41**	Ordenamiento burbuja I
sw13	Consulta de restaurantes por día	**sw42**	Ordenamiento burbuja II
sw14	Consulta de restaurantes por hora	**sw43**	Identifica palabra mayor de un texto
sw15	Consulta de restaurantes por evento	**sw44**	Identifica palabra menor de un texto
sw16	Consulta de restaurantes por tipo de servicio	**sw45**	Muestra tablas de multiplicar
sw17	Consulta de servicios por zona	**sw46**	Calcula parámetros de regresión múltiple
sw18	Agrupa palabras por tamaño	**sw47**	Calcula tamaño de muestra (población desconocida)
sw19	Cuenta palabras de un texto	**sw48**	Calcula tamaño de palabras de un texto
sw20	Calcula desviación estandar I	**sw49**	Calcula varianza I
sw21	Calcula desviación estandar II	**sw50**	Calcula varianza II
sw22	Consulta de establecimientos en méxico	**sw51**	Calcula mediana (método burbuja) I
sw23	Consulta de establecimientos por ID	**sw52**	Calcula mediana (método quicksort) I

sw24	Consulta de establecimientos por estado	**sw53**	Calcula mediana (metodo shell) I
sw25	Consulta de establecimientos por municipio	**sw54**	Calcula mediana (método burbuja) II
sw26	Consulta de establecimientos por CP	**sw55**	Calcula mediana (método quicksort) II
sw27	Agrupa palabras por letra inicial A-I	**sw56**	Calcula mediana (metodo shell) II
sw28	Agrupa palabras por letra inicial J-Q	**sw57**	Calcula moda
sw29	Agrupa palabras por letra iniciaL R-Z		

5.3 Pruebas del tiempo de respuesta

5.3.1 Software utilizado

Para realizar las pruebas del tiempo de respuesta, se procedió a montar y desplegar cada uno de los Servicios Web en el servidor de aplicaciones WildFly. WildFly es una herramienta de código abierto implementado en *Java*, y es muy popular por las ventajas que proporciona, como la capacidad de respuesta, la conectividad y su alto rendimiento (Red Hat, 2017). En la Figura 9 se muestra la interfaz del servidor WildFly con un servicio web desplegado. Para realizar las peticiones se cerró esta interfaz y se dejó únicamente el servidor corriendo desde la consola para evitar que otros procesos adicionales interfirieran en los resultados.

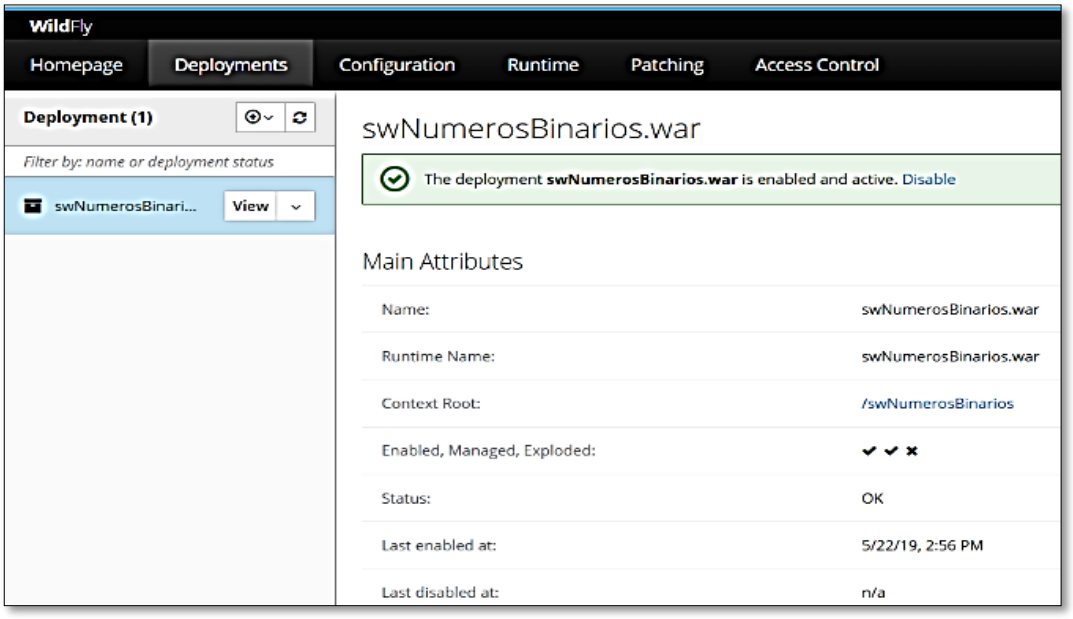

Figura 9. Interfaz gráfica de WildFly al desplegar un servicio

Por otra parte, en la máquina cliente, se instaló la herramienta *JMeter* para realizar la medición del tiempo de respuesta de los servicios. Apache *JMeter* es un software de código abierto, una aplicación Java 100% pura, diseñada para cargar el comportamiento funcional

de la prueba y medir el rendimiento y otras características de aplicaciones web principalmente, aunque su uso se ha expandido y se puede usar para realizar pruebas tanto en recursos estáticos como dinámicos (Apache Foundation, 2019). La interfaz gráfica de este entorno de pruebas utilizado se ilustra en la Figura 10.

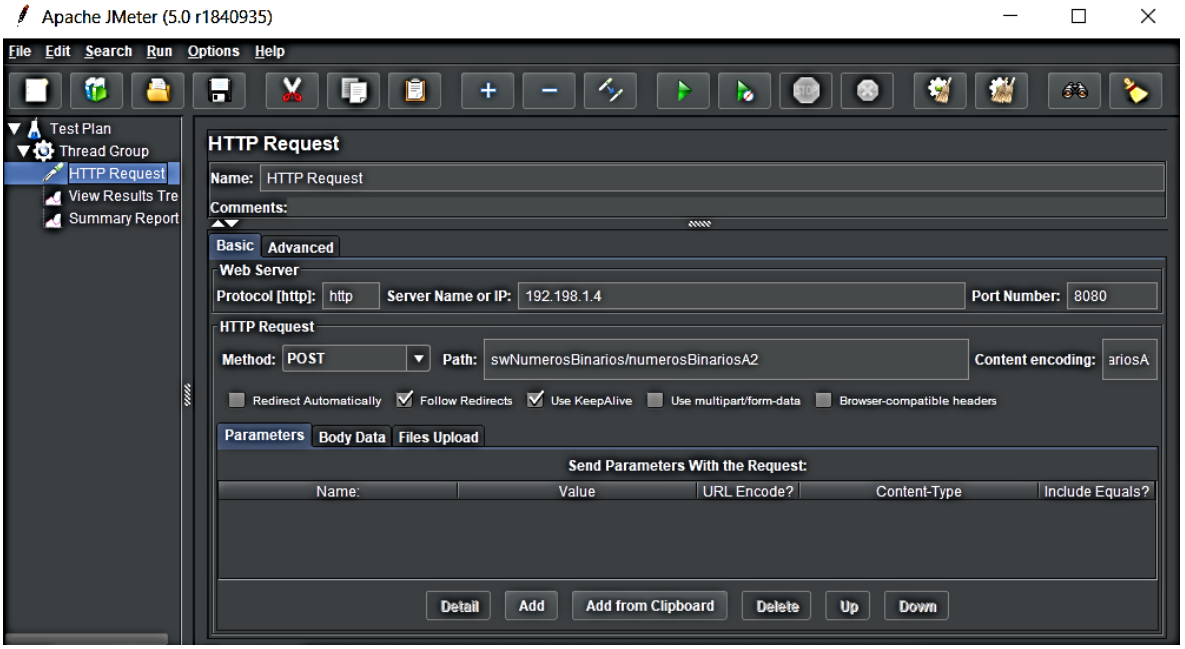

Figura 10. Interfaz gráfica de *JMeter* con los datos para hacer una petición

5.3.2 Hardware utilizado

Con respecto a los elementos de hardware, se utilizaron dos equipos de cómputo propios, así como algunos otros recursos que se encontraron disponibles en el departamento de cómputo de la institución. A continuación, en la Tabla 3, se muestran las especificaciones técnicas de los elementos de hardware utilizados durante el experimento.

Tabla 3. Especificaciones técnicas de los elementos utilizados en el experimento

Elemento	Nombre	Características
Cliente	DESKTOP - OLHI5KT	**Procesador:** Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99GHz. **Memoria RAM**: 12.0 GB **Tipo de sistema**: Sistema operativo de 64 bits, procesador x64

Servidor	DESKTOP-IEJE6QP	**Procesador:** Intel(R) Core(TM) i3-3110M CPU @ 2.40GHz 2.40GHz. **Memoria RAM**: 4.0 GB **Tipo de sistema**: Sistema operativo de 64 bits, procesador x64
Topología de Conexión	Cable cruzado	Cable UTP, conectores RJ45
Red	Ethernet 2	Realtek PCIe FE Family Controller 2 **Unidad de transmisión máxima:** 1500 **Dirección IP Cliente**: 192.198.1.4 **Dirección IP Servidor:** 192.168.1.3

5.3.3. Eliminación del ruido experimental

Los factores de ruido son variables que pueden o no controlarse en el experimento, pero que no pueden controlarse en el proceso, o que no pueden controlarse bien (Walpole, Myers, & Myers, 2012). Debido a que estos factores pueden afectar los resultados de las pruebas, se procedió a aislar los factores que pudieran causar ruido en el experimento.

5.3.3.1 Ruido ocasionado por componentes de hardware

Para evitar que el tráfico en la red por el acceso de varios usuarios interfiera en el tiempo de respuesta que se mide, se decidió trabajar con una topología de cable cruzado en el que únicamente interactúa el equipo cliente con el equipo servidor. Por otra parte, para garantizar que la comunicación fuera estable, se verificó que el cable estuviera en buenas condiciones y se contó con el apoyo de personal del departamento de cómputo para verificar su buen funcionamiento.

5.3.3.2. Ruido ocasionado por componentes de software

Los procesos que se ejecutan en segundo plano, podrían interferir en las mediciones del tiempo de respuesta ya que estarían compartiendo recursos de cómputo con los servicios y las peticiones. Para evitar esto, se procedió a detener los procesos que se ejecutan en segundo plano tales como antivirus, actualizaciones, entre otros.

5.3.4. Ejecución de las pruebas

Las pruebas de tiempo de respuesta se llevaron a cabo desplegando únicamente un servicio web a la vez, con la finalidad de que en el servidor estuviera corriendo únicamente el servicio que se prueba y de esta forma los resultados no fueran afectados. Se procedió a realizar diferentes cantidades de peticiones ya que, de acuerdo a la definición de tiempo de respuesta, es necesario realizar varias peticiones para que esta medición sea válida. El número de peticiones fue desde 1, 5, 10, 20 hasta 50 peticiones de manera simultánea. Para cada una de las pruebas que incluyen las diferentes arquitecturas que se desean comparar en este experimento, se realizaron 30 iteraciones consecutivas y se registraron los resultados obtenidos. Debido al número de peticiones e iteraciones realizadas, se obtuvo un número grande de datos, por lo cual posteriormente se realizó su tratamiento mediante métodos estadísticos.

A continuación, en la Tabla 4 se presenta el promedio de los tiempos de respuesta (**TR**) de cada servicio web, en donde **A1** corresponde a la arquitectura de conjunto de clases relacionadas, **A2** a la arquitectura de clase única y **A3** a la arquitectura de clases internas.

Tabla 4. Tiempo de respuesta promedio en milisegundos de los Servicios Web que forman parte del conjunto de unidades de prueba para el experimento

Núm. SW	TR (A1)	TR (A2)	TR (A3)	Núm. SW	TR(A1)	TR(A2)	TR(A3)
sw1	321	315	313	**sw30**	73	67	68
sw2	350	275	307	**sw31**	137	133	137
sw3	350	275	307	**sw32**	70	48	49
sw4	18	15	14	**sw33**	42	26	36
sw5	66	31	33	**sw34**	36	28	26
sw6	112	108	110	**sw35**	32	27	31
sw7	117	123	115	**sw36**	55	53	54
sw8	12	11	12	**sw37**	265	255	262
sw9	122	118	120	**sw38**	55	53	55
sw10	120	126	118	**sw39**	165	159	157
sw11	25	14	12	**sw40**	186	163	156

sw12	122	106	94	**sw41**	85	61	70
sw13	70	37	73	**sw42**	108	93	97
sw14	37	36	39	**sw43**	115	100	105
sw15	90	71	67	**sw44**	94	75	83
sw16	95	76	72	**sw45**	106	99	104
sw17	110	100	107	**sw46**	5187	5130	5239
sw18	114	104	111	**sw47**	11	8	9
sw19	173	137	135	**sw48**	81	55	73
sw20	20	13	15	**sw49**	106	101	104
sw21	24	14	17	**sw50**	97	67	72
sw22	180	150	160	**sw51**	209	196	175
sw23	187	154	159	**sw52**	76	65	66
sw24	18	15	16	**sw53**	79	62	75
sw25	26	23	24	**sw54**	71	59	65
sw26	22	19	20	**sw55**	73	67	66
sw27	247	210	228	**sw56**	77	62	70
sw28	184	179	198	**sw57**	69	59	68
sw29	299	177	281				

De acuerdo a los datos presentados en la tabla anterior, se puede observar que en 48 de las 57 pruebas, el mayor tiempo de respuesta lo obtuvieron los Servicios Web con arquitectura de conjunto de clases, este comportamiento era el esperado. Sin embargo, en el resto de las pruebas el mayor tiempo lo obtuvieron las otras arquitecturas. Por otra parte, la arquitectura que lleva el menor tiempo de respuesta es la que corresponde a una única clase ya que, en 43 de las 57 pruebas se observó este comportamiento, que era el esperado. Finalmente, la arquitectura de clases internas quedó en el nivel intermedio, al ser más rápidas que el conjunto de clases, pero más lentas que la arquitectura de única clase. Cabe mencionar que las variaciones que se presentaron en los resultados, al no mostrarse el mismo comportamiento

de cada arquitectura en todas las pruebas, pudieron ser causadas por el planificador de tareas de los procesos y servicios que están en ejecución en el sistema operativo y que no se pueden detener debido a que son necesarios para su correcto funcionamiento.

5.4 Pruebas de métricas estáticas

5.4.1 Software utilizado

Para realizar las pruebas con las métricas estáticas de coherencia, cohesión y acoplamiento, se utilizó el Marco Orientado a Objetos para la medición de calidad en arquitecturas de software, que se extendió previamente como parte de este trabajo de tesis. La herramienta recibe como entrada una carpeta de archivos que puede contener uno o más archivos de código java.

5.4.2 Hardware utilizado

En lo que se refiere a los elementos de hardware que se utilizaron para estas pruebas, únicamente se requirió del equipo de cómputo en el que se ejecutó la herramienta de medición.

5.4.3. Ejecución de las pruebas

Para realizar las mediciones a nivel de código de los Servicios Web en sus diferentes arquitecturas, se procedió a ejecutar la herramienta de medición y se inició con las pruebas a los 57 Servicios Web. En cada prueba se ingresó una carpeta con los archivos *java* para ser analizados y realizar los cálculos correspondientes. Los resultados obtenidos para cada una de las tres métricas se documentaron de manera ordenada, utilizando hojas de trabajo de Excel para su posterior análisis e interpretación.

A continuación, en las Tablas 5, 6 y 7 se presentan los valores de coherencia, cohesión y acoplamiento que se obtuvieron durante las pruebas. En las tablas se presentan los valores de las métricas para cada una de las tres arquitecturas que se midieron, en donde **A1** corresponde a la arquitectura de conjunto de clases relacionadas, **A2** a la arquitectura de clase única y **A3** a la arquitectura de clases internas.

Tabla 5. Resultados obtenidos en la medición de coherencia (CrCU) de los Servicios Web que forman parte del conjunto de unidades de prueba para el experimento realizado

Núm. SW	Cr (A1)	Cr (A2)	Cr (A3)	Núm. SW	Cr (A1)	Cr (A2)	Cr (A3)
sw1	1	1	1	**sw30**	1	1	1
sw2	1	1	1	**sw31**	1	1	1
sw3	1	1	1	**sw32**	1	1	1
sw4	1	1	1	**sw33**	1	1	1
sw5	1	1	1	**sw34**	1	1	1
sw6	1	1	1	**sw35**	1	1	1
sw7	1	1	1	**sw36**	1	1	1
sw8	1	1	1	**sw37**	1	1	1
sw9	1	1	1	**sw38**	1	1	1
sw10	1	1	1	**sw39**	1	1	1
sw11	0.6666	0.6666	0.6666	**sw40**	1	1	1
sw12	0.6666	0.6666	0.6666	**sw41**	1	1	1
sw13	0.6666	0.6666	0.6666	**sw42**	1	1	1
sw14	0.6666	0.6666	0.6666	**sw43**	1	1	1
sw15	0.6666	0.6666	0.6666	**sw44**	1	1	1
sw16	0.6666	0.6666	0.6666	**sw45**	1	1	1
sw17	0.6666	0.6666	0.6666	**sw46**	1	1	1
sw18	1	1	1	**sw47**	1	1	1
sw19	1	1	1	**sw48**	1	1	1
sw20	1	1	1	**sw49**	1	1	1
sw21	1	1	1	**sw50**	1	1	1
sw22	1	1	1	**sw51**	1	1	1

sw23	0.6666	0.6666	0.6666	**sw52**	1	1	1
sw24	1	1	1	**sw53**	1	1	1
sw25	1	1	1	**sw54**	1	1	1
sw26	1	1	1	**sw55**	1	1	1
sw27	1	1	1	**sw56**	1	1	1
sw28	1	1	1	**sw57**	1	1	1
sw29	1	1	1				

Tabla 6. Resultados obtenidos en la medición de cohesión (LCOM*) de los Servicios Web que forman parte del conjunto de unidades de prueba para el experimento realizado

Núm. SW	LCOM* (A1)	LCOM* (A2)	LCOM* (A3)	Núm. SW	LCOM* (A1)	LCOM* (A2)	LCOM* (A3)
sw1	0.5934	0.4743	0.5934	**sw30**	0	0.05069	0
sw2	0.5934	0.4743	0.5934	**sw31**	0	0.75	0
sw3	0.5934	0.4743	0.5934	**sw32**	0	0.75	0
sw4	0	0.1875	0	**sw33**	0	0.8571	0
sw5	0.1466	1	0.1466	**sw34**	0	0.825	0
sw6	0.2962	0.8484	0.2962	**sw35**	0	0.8055	0
sw7	0.2222	0.8518	0.2222	**sw36**	0	0.8333	0
sw8	0.2962	0.8484	0.2962	**sw37**	0	0.8333	0
sw9	0.2962	0.8484	0.2962	**sw38**	0	0.8333	0
sw10	0.2222	0.8484	0.2222	**sw39**	0	0.4999	0
sw11	0.2962	0.8484	0.2962	**sw40**	0	0.4999	0
sw12	0.2898	0.8636	0.2898	**sw41**	0	0.7333	0
sw13	0.2898	0.8636	0.2898	**sw42**	0	0.7857	0

sw14	0.2898	0.8636	0.2898	**sw43**	0	0.75	0
sw15	0.2898	0.8636	0.2898	**sw44**	0	0.75	0
sw16	0.2898	0.8636	0.2898	**sw45**	0.25	0.7037	0.25
sw17	0.2898	0.8636	0.2898	**sw46**	0.25	0.7037	0.25
sw18	0	0.4909	0	**sw47**	0.2215	0.025	0.1361
sw19	0	0.4909	0	**sw48**	0	0.5909	0
sw20	0	0.7142	0	**sw49**	0	0.9	0
sw21	0	0.7142	0	**sw50**	0	0.6666	1
sw22	0	0.7142	0	**sw51**	0	0.4444	1
sw23	0.3788	0.9013	0.3788	**sw52**	0	0.4444	0
sw24	0.2344	0.8975	0.2344	**sw53**	0	0.7788	0
sw25	0.3122	0.8981	0.3109	**sw54**	0	0.1666	0
sw26	0.3122	0.8981	0.3094	**sw55**	0	0	0
sw27	0.3122	0.8981	0.3109	**sw56**	0	0.7788	0
sw28	0	0.5069	0	**sw57**	0	0.1666	0
sw29	0	0.5069	0				

Tabla 7. Resultados obtenidos en la medición de acoplamiento (COF) de los Servicios Web que forman parte del conjunto de unidades de prueba para el experimento realizado

Núm. SW	COF (A1)	COF (A2)	COF (A3)	Núm. SW	COF (A1)	COF(A2)	COF(A3)
sw1	0.1	0	0.1	**sw30**	0.1	0	0.1
sw2	0.1	0	0.1	**sw31**	0.25	0	0.25
sw3	0.1	0	0.1	**sw32**	0.25	0	0.25
sw4	0.1666	0	0.1666	**sw33**	0.3333	0	0.3333
sw5	0.4166	0	0.4166	**sw34**	0.2	0	0.2

sw6	0.25	0	0.25	**sw35**	0.2	0	0.2
sw7	0.25	0	0.25	**sw36**	0.2	0	0.2
sw8	0.1666	0	0.1666	**sw37**	0.3333	0	0.3333
sw9	0.25	0	0.25	**sw38**	0.3333	0	0.3333
sw10	0.25	0	0.25	**sw39**	0.3333	0	0.3333
sw11	0.1666	0	0.1666	**sw40**	0.3333	0	0.3333
sw12	0.3333	0	0.3333	**sw41**	0.25	0	0.25
sw13	0.3333	0	0.3333	**sw42**	0.1666	0	0.1666
sw14	0.3333	0	0.3333	**sw43**	0.3333	0	0.3333
sw15	0.3333	0	0.3333	**sw44**	0.3333	0	0.3333
sw16	0.3333	0	0.3333	**sw45**	0.3333	0	0.3333
sw17	0.3333	0	0.3333	**sw46**	0.3333	0	0.3333
sw18	0.1616	0	0.1616	**sw47**	0.1333	0	0.1333
sw19	0.1616	0	0.1616	**sw48**	0.0909	0	0.0909
sw20	0.5	0	0.5	**sw49**	0.3333	0	0.3333
sw21	0.25	0	0.25	**sw50**	0.3333	0	0.3333
sw22	0.25	0	0.25	**sw51**	0.25	0	0.25
sw23	0.3333	0	0.3333	**sw52**	0.25	0	0.25
sw24	0.25	0	0.25	**sw53**	0.5	0	0.5
sw25	0.33	0	0.33	**sw54**	0.5	0	0.5
sw26	0.33	0	0.33	**sw55**	0.5	0	0.5
sw27	0.33	0	0.33	**sw56**	0.5	0	0.5
sw28	0.1	0	0.1	**sw57**	0.5	0	0.5
sw29	0.1	0	0.1				

5.5 Resumen del capítulo

En este capítulo se describió el proceso que se siguió para realizar la ejecución del experimento. En un principio se describe el desarrollo de la herramienta que se utilizó para la medición de las métricas estáticas. Posteriormente se presenta la lista de nombres de los servicios web que se utilizaron como unidades de prueba, así como la forma en que se organizaron las tres arquitecturas que se consideraron. Así mismo, se describen los recursos, el proceso y las herramientas que se utilizaron para las pruebas, tanto con las métricas estáticas, como para las métricas dinámicas. Por último, se muestran los resultados obtenidos en las pruebas con las diferentes métricas. Estos resultados fueron analizados con métodos estadísticos como se muestra en el siguiente capítulo.

Capítulo 6. Análisis estadístico e interpretación de resultados

6.1 Introducción

El uso de métodos estadísticos en la manufactura, las fuentes de energía, los productos farmacéuticos, el software para computadoras y muchas otras áreas, implican el acopio de información o datos científicos. Los datos deben ser recabados, resumidos, reportados y almacenados para su examen cuidadoso. Sin embargo, hay una diferencia profunda entre el acopio de información científica y la estadística inferencial (Walpole, et al., 2012). Esta última puede utilizar una gran cantidad de herramientas dentro de las cuales se encuentran los métodos estadísticos que han sido diseñados para contribuir al proceso de realizar juicios científicos frente a la incertidumbre y la variación.

En esta sección del presente trabajo de tesis se realizó el análisis de los datos para contar con la información necesaria que permita aceptar o rechazar las hipótesis planteadas en el diseño experimental y de igual forma obtener conclusiones acerca de los resultados obtenidos.

6.2 Análisis estadístico para los datos de tiempo de respuesta

6.2.1 Pruebas de normalidad

A partir de las pruebas realizadas para obtener el tiempo de respuesta de los Servicios Web y debido al número de iteraciones por cada prueba, se obtuvo un conjunto grande de datos. Estos datos por sí solos no proporcionan la suficiente información para realizar conclusiones acerca del fenómeno de estudio. Debido a esto, fue necesario realizar un análisis estadístico inferencial mediante la aplicación de técnicas conocidas para este fin.

La distribución de probabilidad continua más importante en todo el campo de la estadística es la distribución normal, la cual describe de manera aproximada muchos fenómenos que ocurren en la naturaleza, la industria y la investigación (Walpole, et al., 2012). Conocer la distribución de la densidad de los resultados experimentales de este trabajo es el primer paso para realizar el análisis estadístico. Se dice que los resultados experimentales son normales cuando su función de densidad tiene una forma acampanada y es simétrica con respecto a un parámetro estadístico determinado (Walpole, 1999).

Para verificar la normalidad de los datos obtenidos en las pruebas experimentales de este trabajo, se utilizó la prueba de normalidad *Kolmogorov-Smirnov*, la cual es una prueba no paramétrica ampliamente utilizada para evaluar el ajuste de cualquier distribución

hipotética, en este caso la distribución normal. Esta prueba se realizó mediante la herramienta *SPSS Statistics*, y con base en los resultados se determinó que la distribución de la prueba es una distribución normal.

6.2.2 Prueba paramétrica ANOVA de un factor

Debido a la naturaleza de los datos que se verificaron en la sección anterior, se procedió a realizar la prueba paramétrica ANOVA de un factor. Esta prueba es una generalización del contraste de igualdad de medias para dos muestras independientes y se utiliza para comparar varios grupos en una variable cuantitativa o factor, que para este caso es la estructura interna de cada Servicio Web. A continuación, en las Tablas 8 y 9 se presentan los resultados obtenidos mediante la herramienta utilizada para el análisis estadístico.

Tabla 8. Estadísticos descriptivos de los datos de tiempo de respuesta de los Servicios Web

Descriptivos

Tiempo de respuesta

	N	Media	Desv. Desviación	Desv. Error	95% del intervalo de confianza para la media		Mínimo	Máximo
					Límite inferior	Límite superior		
1.00	57	112.4912	84.60710	11.20648	90.0419	134.9405	11.00	350.00
2.00	57	95.0000	73.02079	9.67184	75.6250	114.3750	8.00	315.00
3.00	57	102.5965	80.65333	10.68279	81.1963	123.9967	9.00	313.00
Total	171	103.3626	79.42846	6.07405	91.3723	115.3528	8.00	350.00

Tabla 9. Resultados de la prueba ANOVA aplicada a los datos de tiempo de repuesta de los Servicios Web

ANOVA

Tiempo de respuesta

	Suma de cuadrados	gl	Media cuadrática	F	Sig.
Entre grupos	8769.556	2	4384.778	.693	.502
Dentro de grupos	1063739.965	168	6331.786		
Total	1072509.520	170			

Como se puede observar en la Tabla 9, el grado de significancia obtenido a partir de la prueba ANOVA es mayor a 0.5, por lo cual se determinó que no hay diferencia significativa entre las medias de los grupos de los datos obtenidos. Para el experimento realizado en este trabajo

los grupos consistieron de las tres arquitecturas diferentes en la estructura interna de los Servicios Web. Estos resultados aún no proporcionan información detallada acerca de la variación entre los datos, lo cual es muy importante conocer debido a la naturaleza del experimento. Por consecuencia, se procedió a utilizar una representación gráfica de datos, para obtener información más precisa y detallada.

6.2.3 Representación gráfica de datos

Con frecuencia el resumen gráfico de un conjunto de datos puede proporcionar información sobre el sistema del que se obtuvieron. Las gráficas pueden ilustrar información que permite que los resultados de la inferencia estadística formal se comuniquen mejor al científico o al ingeniero. Así también, las gráficas o el análisis exploratorio de los datos pueden enseñar al analista información que no se obtiene del análisis formal (Walpole et al., 2012). Por lo anterior, y después de haber aplicado los métodos estadísticos descritos en las secciones anteriores. A continuación, en las Figuras 11 y 12 se muestran las gráficas que corresponden al tiempo de respuesta promedio de cada uno de los Servicios Web que se probaron.

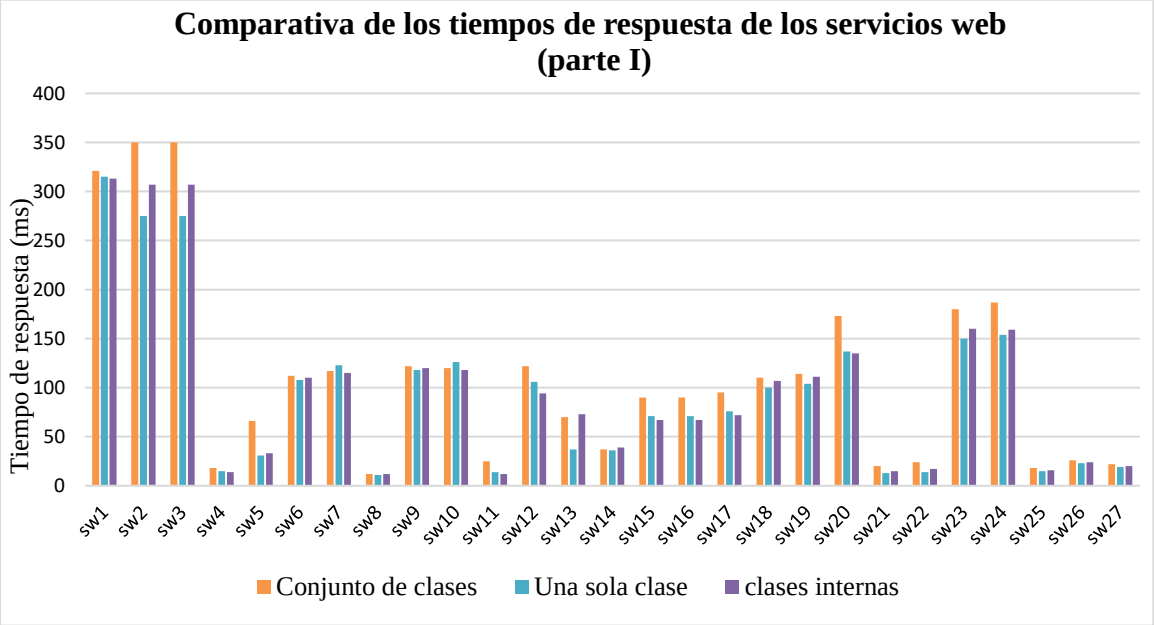

Figura 11. Gráfica comparativa del tiempo de respuesta promedio en los diferentes niveles de granulación

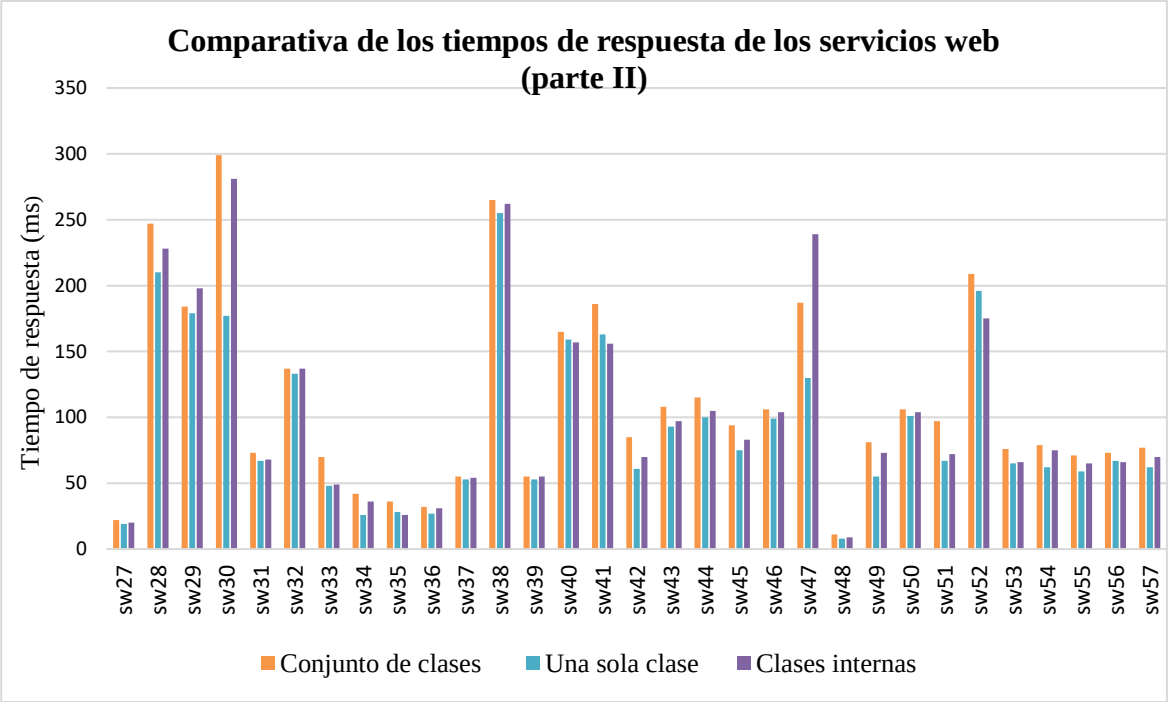

Figura 12. Gráfica comparativa del tiempo de respuesta promedio en los diferentes niveles de granulación

Con base en el análisis exploratorio de la gráfica anterior se observó que en la mayoría de los casos el menor tiempo de respuesta es el que corresponde a los Servicios Web con granulación de única clase y el mayor tiempo de respuesta es el que corresponde a los Servicios Web con granulación de conjunto de clases relacionadas. Cabe mencionar que los tiempos de respuesta fueron medidos en milisegundos, por esta razón es que la diferencia entre los tiempos de las diferentes arquitecturas no es significativa[2]. Esta diferencia hubiera sido más significativa si las mediciones se hubieran hecho en microsegundos o nanosegundos, no se hizo de esta manera por restricciones de la herramienta de medición utilizada.

6.3 Análisis estadístico para las métricas estáticas

En esta parte del análisis estadístico se realizó la representación gráfica y el análisis exploratorio de los resultados obtenidos en la medición de la coherencia, cohesión y acoplamiento en la estructura interna de los Servicios Web de prueba. Cabe mencionar que por el número de datos obtenidos no fue necesario aplicar otros métodos estadísticos para este análisis. A continuación, en las Figuras 13 y 14 se muestran las gráficas correspondientes a los resultados obtenidos para la métrica de coherencia de caso de uso.

[2] Con el fin de visualizar de manera más clara la diferencia entre los tiempos de respuesta para cada arquitectura, se realizaron algunas pruebas adicionales midiendo el tiempo de respuesta en nanosegundos. Los resultados de estas pruebas se presentan en el **Anexo A: Pruebas adicionales de tiempo de respuesta** de este documento.

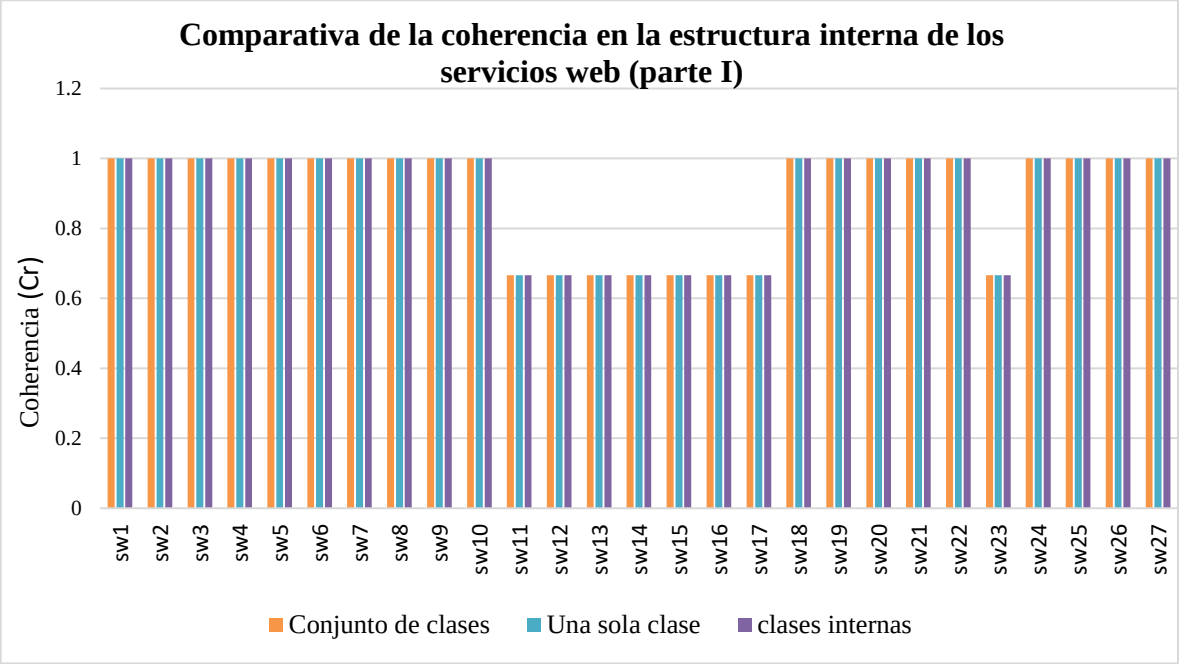

Figura 13. Gráfica comparativa de la coherencia de caso de uso en los diferentes niveles de granulación de los Servicios Web

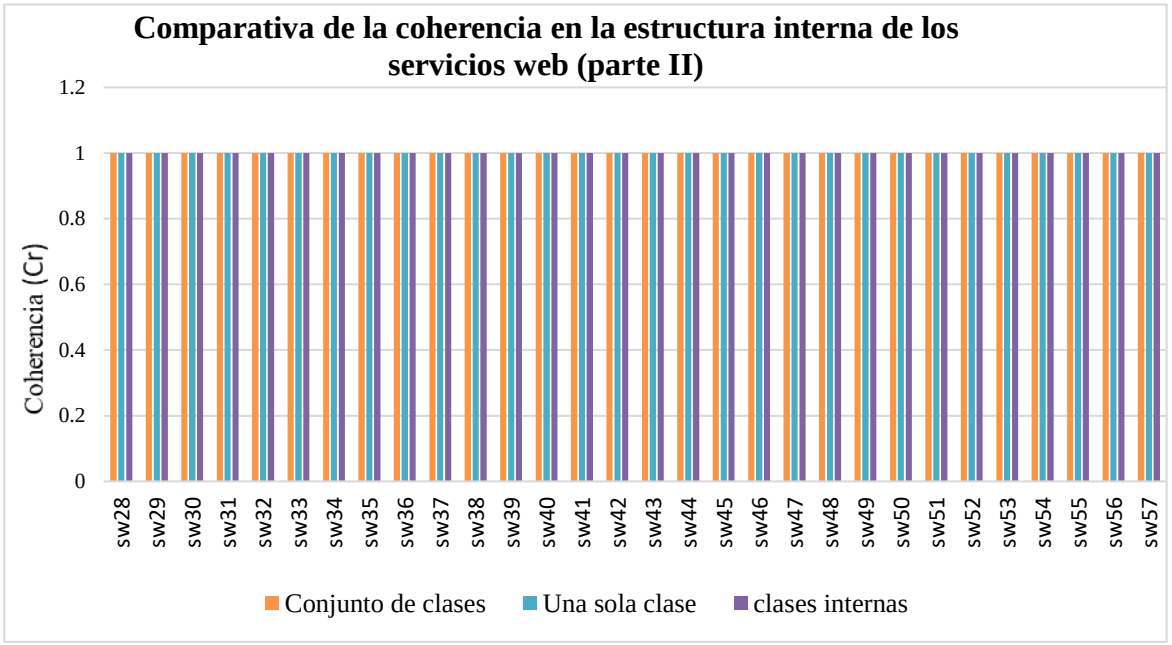

Figura 14. Gráfica comparativa de la coherencia de caso de uso en los diferentes niveles de granulación de los Servicios Web

De acuerdo a la información visual que proporcionan las gráficas, se puede notar que el grado de coherencia se mantiene igual para las tres diferentes arquitecturas, se observa que el comportamiento fue el mismo para los 57 servicios por lo que se puede concluir que independientemente del nivel de granulación, el grado de coherencia es el mismo y será bueno siempre y cuando el código esté programado de tal forma que se cumplan con las características que requiere la métrica de coherencia para alcanzar su valor óptimo. Cabe mencionar que debido a que el código se modificó como parte de este trabajo de tesis y que se retomaron las buenas prácticas de la programación orientada a objetos, los valores de coherencia son óptimos en la mayoría de los casos. Sin embargo, esto no sucede en muchos casos de la vida real en donde se presta poca atención a estas prácticas a la hora de programar, lo cual provoca que se incluyan fragmentos de código innecesario y a su vez aumente el número de líneas de código, que en arquitecturas grandes puede aumentar su complejidad y dificultar su entendimiento.

Para la medición de carencia de cohesión se utilizó la métrica LCOM*. A continuación, en las Figuras 15 y 16 se muestran las gráficas correspondientes a los resultados obtenidos para esta métrica.

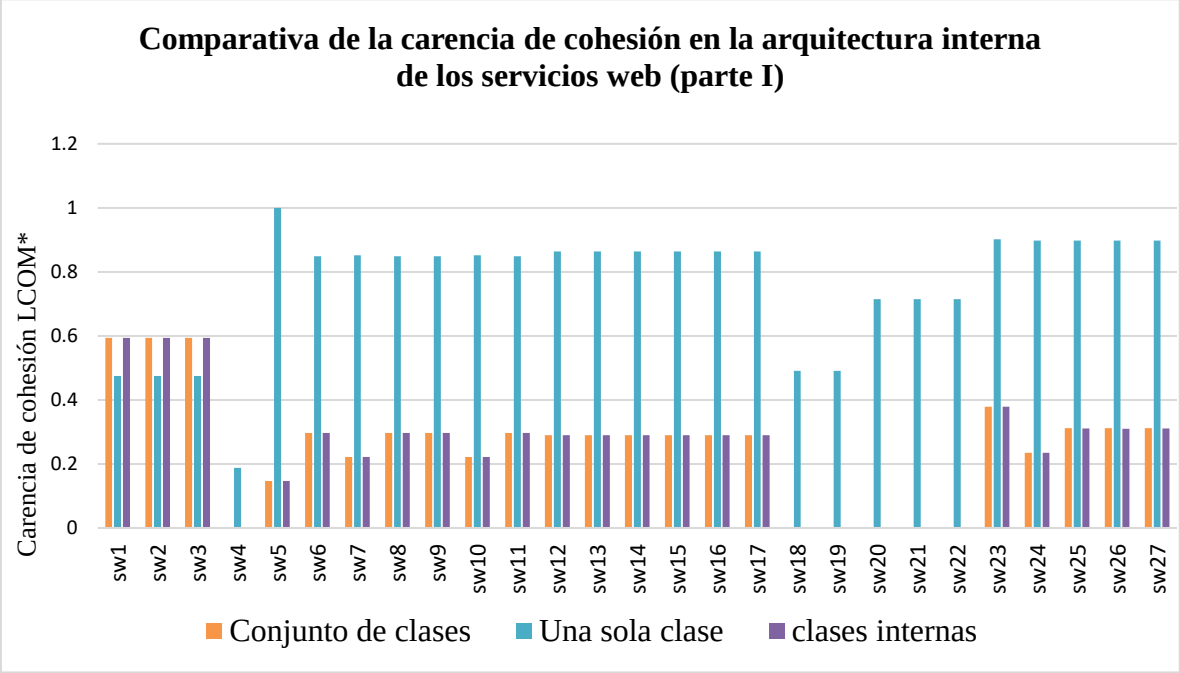

Figura 15. Gráfica comparativa de la carencia de cohesión (LCOM*) en los diferentes niveles de granulación de los Servicios Web

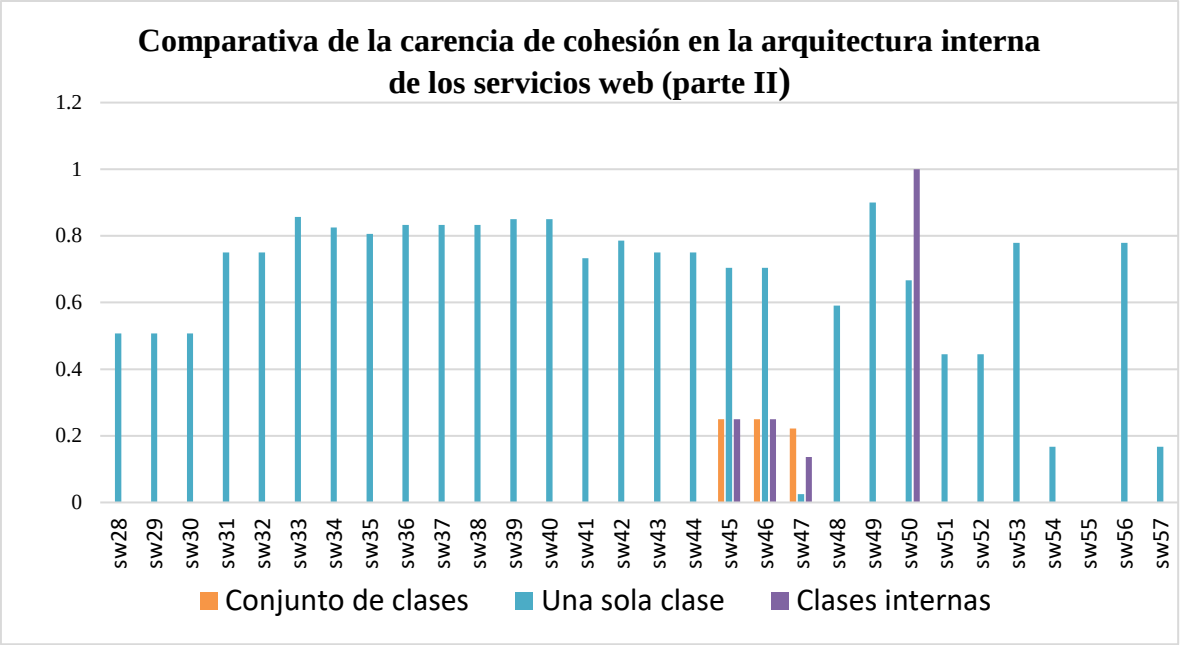

Figura 16. Gráfica comparativa de la carencia de cohesión (LCOM*) en los diferentes niveles de granulación de los Servicios Web

Debido a que la métrica LCOM* mide la carencia de cohesión, el peor valor se tiene a medida que el resultado aumenta, es decir, mientras más se acerque a uno, menos cohesión se tiene en el código. Como se observa en las gráficas, los valores más altos los tiene la arquitectura de una sola clase, es decir, es la arquitectura que menos cohesión tiene. Así mismo, se observa que en las otras dos arquitecturas que corresponden a la arquitectura de única clase y clases internas, los resultados son iguales, es decir, cuando la funcionalidad se agrupa en un conjunto de clases relacionadas, el grado de cohesión aumenta a diferencia de cuando toda la funcionalidad está en una única clase. Es importante mencionar que para determinar la calidad del código es necesario tomar en cuenta otras métricas. Aunque para esta métrica la mejor arquitectura es la de un conjunto de clases, para otras métricas puede ser que los mejores valores se presenten en una arquitectura diferente.

A continuación, en las Figuras 17 y 18 se presentan las gráficas correspondientes a los resultados obtenidos en la medición del factor de acoplamiento que es otra de las métricas que se tomaron en cuenta para este trabajo de tesis.

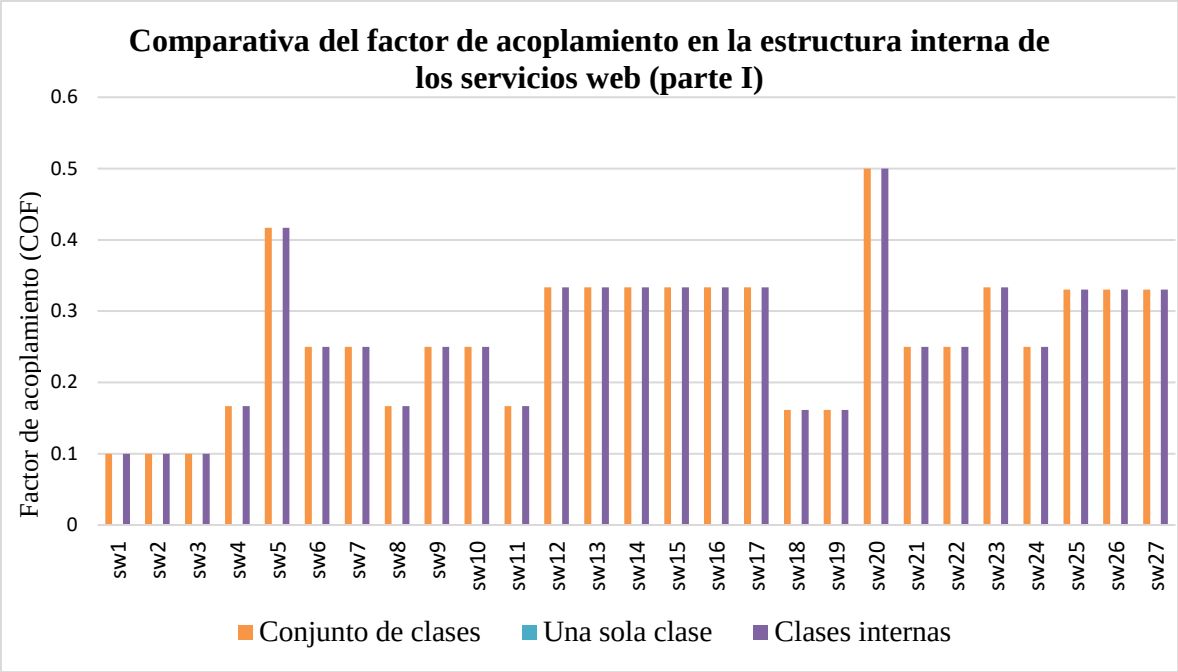

Figura 17. Gráfica comparativa del factor de acoplamiento (COF) en los diferentes niveles de granulación de los Servicios Web

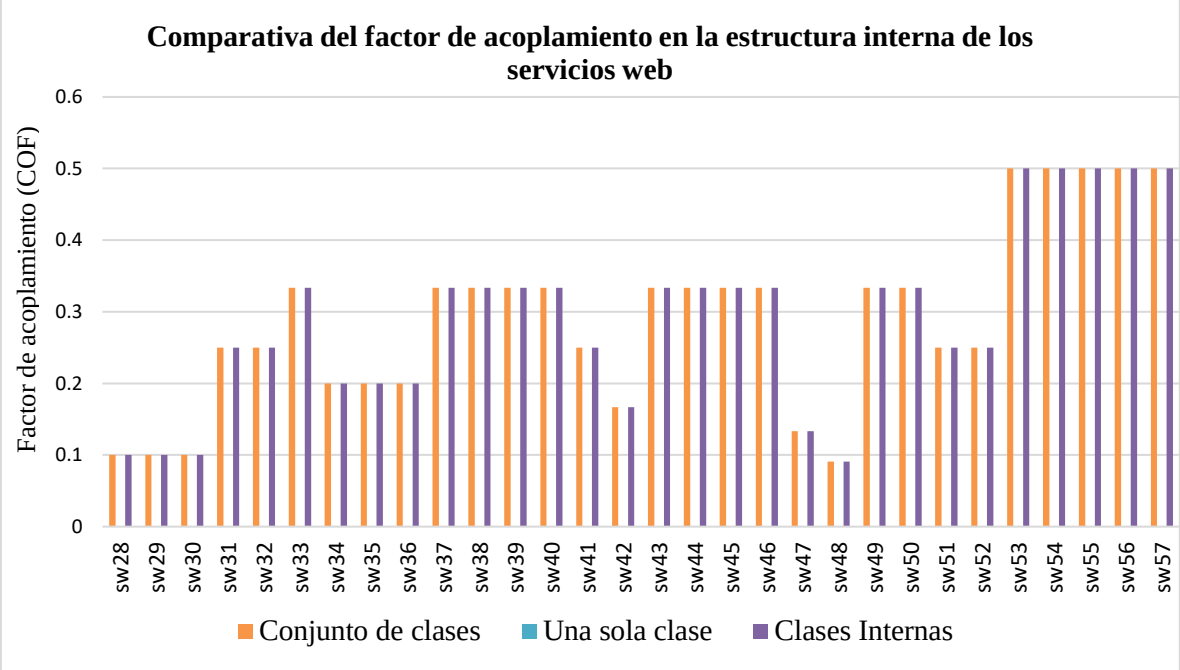

Figura 18. Gráfica comparativa del factor de acoplamiento (COF) en los diferentes niveles de granulación de los Servicios Web

En la gráfica anterior se muestra la comparativa de los resultados obtenidos con la métrica COF (factor de acoplamiento). Se observa que, aunque se comparan las tres arquitecturas, únicamente se visualizan las barras que corresponden al conjunto de clases y a las de clases internas, esto es debido a que, en el caso de la arquitectura de una sola clase, todos los resultados fueron de cero. La métrica utilizada realiza las mediciones con base en los canales de comunicación entre clases, es por ello que en arquitecturas de una única clase el valor siempre será cero. Es decir, cuando en la arquitectura se tiene una única clase esta no se acopla con ninguna otra. Además, se observa que los valores para las otras dos arquitecturas son iguales en todos los casos. Por lo anterior, puede concluirse que cuando las clases están envueltas en una clase contenedora y por consiguiente en un solo archivo, el acoplamiento es el mismo que cuando están en archivos separados como un conjunto de clases.

6.4 Contraste de hipótesis

La verdad o falsedad de una hipótesis estadística no se sabe con absoluta certeza, a menos que se examine toda la población, lo cual, por supuesto, sería poco práctico en la mayoría de las situaciones. Debido a esto se toma una muestra aleatoria de la población de interés y se utilizan los datos contenidos en ella para proporcionar evidencia que respalde o no la hipótesis. Para el experimento realizado en esta tesis se trabajó con una muestra, como fue definida en el diseño experimental. Posteriormente, se realizó el experimento y se obtuvieron los resultados, mismos que fueron tratados y analizados en las secciones anteriores. Con base en los resultados y en el análisis estadístico, se obtuvo la información suficiente que permitió realizar el contraste de las hipótesis planteadas para el experimento.

A continuación, se presentan las hipótesis definidas en el diseño experimental y que se contrastarán en esta sección.

Hipótesis para la experimentación con Servicios Web con granularidad de clase única.

Hipótesis nula (H_0):

$$(TRuc < (TRcc \vee TRci)) \wedge (COFuc < (COFcc \wedge COFci)) \\ \wedge (COHuc = (COHcc \vee COHci)) \wedge (LCOMuc > (LCOMcc \vee LCOMci))$$

Hipótesis alternativa (H_1):

$$(TRuc < (TRcc \vee TRci)) \wedge (COFuc < (COFcc \vee COFci)) \\ \wedge (COHuc > (COHcc \vee COHci)) \wedge (LCOMuc > (LCOMcc \vee LCOMci))$$

A partir del análisis estadístico, se observó que esta arquitectura es la que se lleva el menor tiempo de respuesta y de igual forma el valor más bajo para el factor de acoplamiento. Para la métrica de coherencia se observó que los resultados fueron iguales a los de las otras

arquitecturas. Por lo anterior, se determinó aceptar la hipótesis nula (H_0) y rechazar la hipótesis alternativa (H_1) que se planteó para esta arquitectura.

Hipótesis para la experimentación con Servicios Web con granularidad de conjunto de clases relacionadas.

Hipótesis nula (H_0):

$$\left(COFcc > (COFuc \vee COFci)\right) \wedge \left(TRcc > (TRuc \vee TRci)\right)$$

Hipótesis alternativa (H_1):

$$\left(COFcc < (COFuc \vee COFci)\right) \wedge \left(TRcc < (TRcc \vee TRci)\right)$$

Durante las pruebas con la arquitectura de conjunto de clases se obtuvieron datos suficientes que después de ser analizados permiten determinar la aceptación de la hipótesis nula (H_0) y el rechazo de la hipótesis alternativa (H_1). Esto se determinó teniendo en cuenta que, durante las pruebas, en todos los casos la arquitectura de conjunto de clases obtuvo el mayor tiempo de respuesta y el mayor valor de acoplamiento.

Hipótesis para la experimentación con Servicios Web con granularidad de clases internas.

Hipótesis nula (H_0):

$$\left(COHci > (COHuc \vee COHcc)\right) \wedge \left(LCOMci < (LCOMuc \vee LCOMcc)\right) \wedge \left(COFci < (COFuc \vee COFcc)\right) \wedge (TRci < (TRcc)) \wedge (TRci > (TRuc))$$

Hipótesis alternativa 1 (H_1):

$$\left(COHci = (COHuc \vee COHcc)\right) \wedge \left(LCOMci = (LCOMcc)\right) \wedge (COFci = (COFcc)) \wedge ((TRci > (TRuc)) \wedge TRci < (TRcc))$$

La última arquitectura con la que se realizaron las pruebas fue la de clases internas. Luego de analizar los resultados se pudo notar que el comportamiento de las métricas de coherencia, cohesión y acoplamiento es el mismo que para la arquitectura de conjunto de clases, por lo cual se rechaza la hipótesis nula (H_0) y se acepta la hipótesis alternativa (H_1).

6.5 Comparación entre arquitecturas

En esta sección se realizó la comparación de las tres arquitecturas correspondientes a los niveles de granulación con los que se trabajó en esta tesis: “conjunto de clases relacionadas”, “única clase” y “clases internas”. A continuación, en las Figuras 19, 20 y 21 se muestran de manera gráfica los valores promedio de las métricas estáticas y el tiempo de respuesta para cada arquitectura.

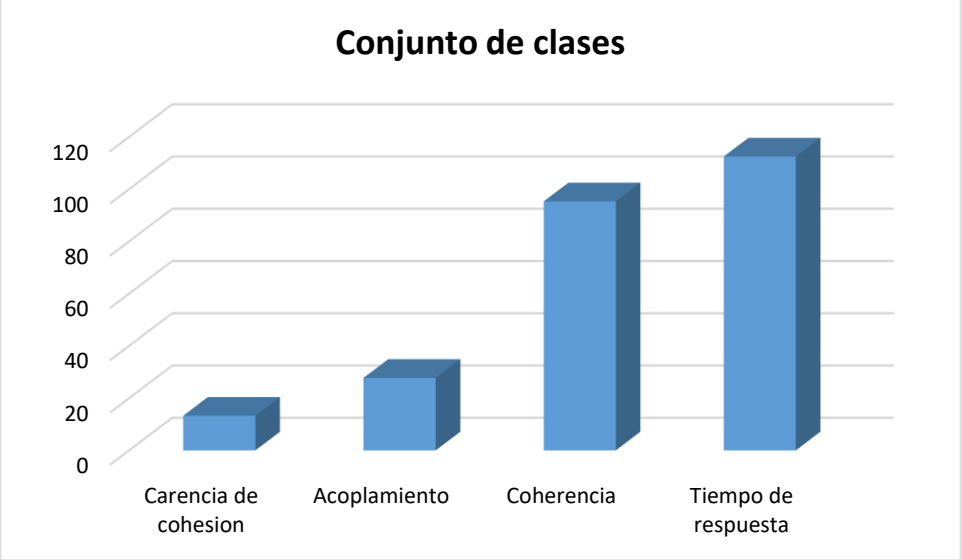

Figura 19. Gráfica con los promedios de las métricas para la arquitectura de conjunto de clases

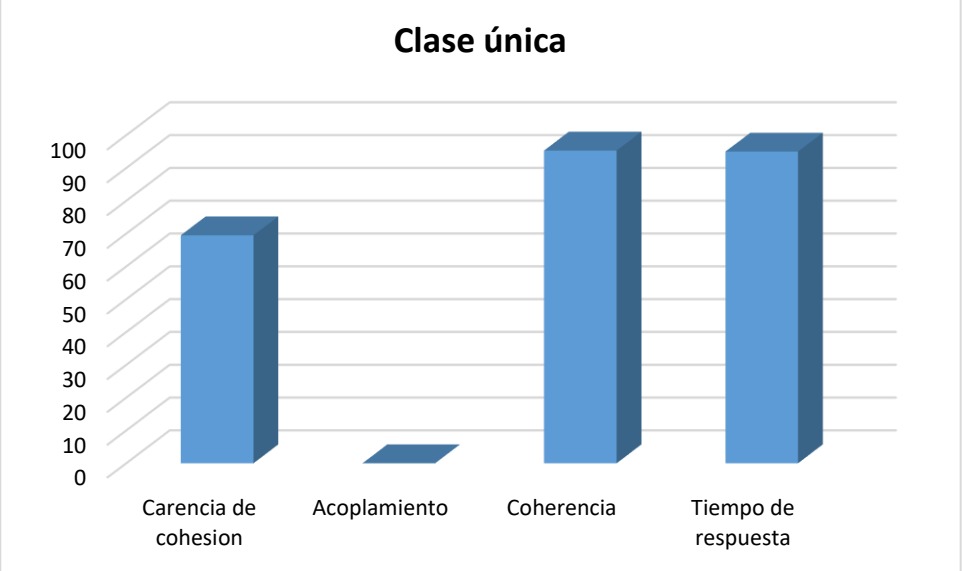

Figura 20. Gráfica con los promedios de las métricas para la arquitectura de clase única

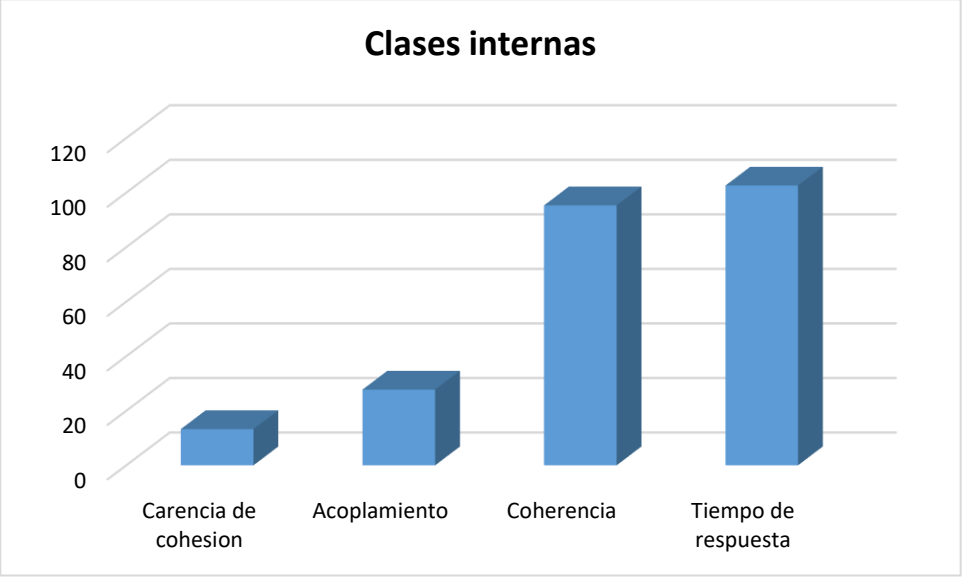

Figura 21. Gráfica con los promedios de las métricas para la arquitectura de clases internas

En la Tabla 10 se presenta una comparativa con base en los valores promedio para cada una de las métricas en las diferentes arquitecturas que se evaluaron.

Tabla 10. Comparativa de las métricas evaluadas en las diferentes arquitecturas que se consideraron en el experimento

Arquitectura	Coherencia	Carencia de Cohesión	Acoplamiento	Tiempo de respuesta
Conjunto de clases	0.953216374	0.132757326	0.277545029	112.4912281
Clase única	0.953216374	0.694834093	0	95
Clases internas	0.953216374	0.132757326	0.277545029	102.5964912

Con base en los datos mostrados, y tomando en cuenta los valores óptimos para cada métrica, se puede observar que la arquitectura de conjunto de clases tiene buenas características en cuanto a los atributos que se miden en las métricas de coherencia y carencia de cohesión. Sin embargo, existe cierto grado de acoplamiento debido a los canales de comunicación y al intercambio de información entre clases. Esto no es deseable en las arquitecturas de software, sobre todo cuando el grado de acoplamiento es alto ya que afectaría su movilidad y reuso, ya que para reutilizarse tendrían que considerarse todas las clases que estén relacionadas. Así

mismo, debido a estos canales de comunicación, el tiempo de respuesta es mayor que en las otras arquitecturas, aunque la diferencia no es significativa.

Para la arquitectura de clase única se observa que existe un nivel alto de carencia de cohesión, lo que puede producir complejidad en la arquitectura limitando su reuso y facilidad de mantenimiento. Así mismo se viola el principio de única responsabilidad. Se observa también que, al ser una única clase, no hay acoplamiento y en consecuencia el tiempo de respuesta es menor que en las otras dos arquitecturas.

En lo que se refiere a las clases con arquitectura de clases internas, se pudo comprobar mediante las pruebas realizadas, que sus características son similares a la arquitectura de conjunto de clases relacionadas. Para las métricas de coherencia, cohesión y acoplamiento los valores resultaron iguales. La diferencia que se encontró aparte de que las clases internas están en un solo archivo, fue con respecto al tiempo de respuesta, ya que en la arquitectura de conjunto de clases el tiempo fue mayor al de las clases internas. Cabe mencionar que, durante las pruebas, la coherencia en las tres diferentes arquitecturas fue la misma, lo cual indica que independientemente de la arquitectura, si se utilizan todos los métodos que se incluyen para alcanzar una meta de valor u objetivo, se tendrá un buen nivel de coherencia. A continuación, en la Tabla 11 se presenta de manera resumida las características antes descritas.

Tabla 11. Comparativa de ventajas y desventajas de las arquitecturas analizadas

Arquitectura	Ventajas	Desventajas
Conjunto de clases	• Alta cohesión • Facilidad de reuso • Fácil de mantener • Principio de única responsabilidad. • Encapsulamiento	• Existe acoplamiento • Mayor tiempo de respuesta
Clase única	• Menor tiempo de respuesta.	• Baja cohesión • Arquitectura compleja • Difícil de mantener y reusar • Más de una responsabilidad • Carencia de protección modular
Clases internas	• Alta cohesión • Facilidad de reuso • Fácil de mantener • Encapsulamiento • Principio de única responsabilidad.	• Existe acoplamiento

6.6 Resumen del capítulo

En este capítulo se presentó el análisis estadístico que se aplicó a los resultados obtenidos en el capítulo anterior. Para los resultados de las métricas dinámicas, debido al gran número de datos, se aplicaron las pruebas de normalidad, la prueba paramétrica ANOVA de un factor y la representación gráfica de los datos. Para los resultados de las métricas estáticas, se realizó la representación gráfica de los datos y el análisis exploratorio. Posteriormente se presentaron los resultados de contrastar las hipótesis. Y finalmente una comparación entre las arquitecturas de acuerdo a los resultados obtenidos, mostrando las ventajas y desventajas que brindan cada arquitectura. Esta comparación junto con el contraste de las hipótesis, permitieron formular conclusiones del trabajo, mismas que se presentan en el capítulo siguiente.

Capítulo 7. Conclusiones

7.1 Conclusiones generales

A partir del experimento realizado como parte de esta investigación se comprobaron algunas hipótesis planteadas en trabajos anteriores realizados por el grupo de ingeniería de software de esta institución, así como las que se plantearon propiamente con base en el conocimiento previo y a la investigación realizada en los inicios de este trabajo de tesis. Algunas hipótesis se rechazaron y otras se aceptaron de acuerdo a la información obtenida a partir de los resultados. Se realizó también un análisis del comportamiento de las métricas de coherencia, cohesión y acoplamiento como parte de las métricas estáticas y del tiempo de respuesta como parte de las métricas dinámicas de software. Con base en lo anterior se presentan las siguientes conclusiones:

- La arquitectura con menos ventajas en cuanto a métricas estáticas fue la de única clase, por lo cual es una opción que no debería de considerarse, o al menos no en una arquitectura compleja o con muchas funcionalidades.
- El grado de coherencia dependerá en gran parte de qué tanto se retomen las buenas prácticas de programación y no necesariamente por la arquitectura de clases que se utilice.
- La arquitectura de conjunto de clases puede presentar cierto grado de acoplamiento. Sin embargo, siempre y cuando sea un acoplamiento débil, la arquitectura seguirá contando con las ventajas propias que esta proporciona. Aprovechar estas ventajas dependerá en gran parte del estilo y la experiencia del desarrollador.
- Las clases internas además de contar con las mismas ventajas de la arquitectura de conjunto de clases en archivos separados, en la mayoría de las pruebas se observó un menor tiempo de respuesta con respecto a esa arquitectura.
- La arquitectura de única clase podría ser una opción en Servicios Web que tengan una responsabilidad de grano fino, de tal forma que toda la funcionalidad pueda ser alojada en una única clase. De esta forma se puede aprovechar la ventaja de la rapidez en cuanto al tiempo de respuesta que presenta esta arquitectura.
- A medida que la cohesión aumenta, también puede aumentar el acoplamiento, pero no es algo que aplique en todos los casos. Siempre y cuando se apliquen los principios de programación orientada a objetos, se mantendrá un acoplamiento débil en la arquitectura.
- Los resultados obtenidos a partir del experimento realizado en este trabajo de tesis son preliminares, por lo tanto, no es posible determinar contundentemente, cuál arquitectura es la mejor, al menos de las estudiadas en este trabajo. Esto debido a que

la calidad del código dependerá mucho de la experiencia y habilidades del desarrollador, y del contexto en que se requiera un diseño de software.

Este trabajo proporciona conocimiento acerca del comportamiento de las métricas tanto estáticas como dinámicas ante diferentes niveles de granulación de estructuras internas de Servicios Web. Así mismo, hace evidente que, adicionalmente a la experiencia y habilidades del desarrollador, es necesario tomar en cuenta el conocimiento obtenido en este trabajo para la toma de decisiones técnicas en la etapa de diseño de las estructuras internas de Servicios Web.

7.2 Trabajos futuros

En este apartado se puntualizan algunos trabajos futuros que se pueden realizar con el objetivo de seguir obteniendo conocimiento acerca de los temas tratados en este trabajo de tesis.

- Realizar experimentación con Servicios Web con arquitecturas más grandes, con un número de líneas de código mayor a los que se utilizaron en este trabajo.
- Realizar experimentación alojando los Servicios Web en un servidor dedicado o en un equipo con mayores capacidades que el utilizado en este trabajo, para así realizar las pruebas con un mayor número de peticiones simultáneas.
- Incluir otras métricas dinámicas para la medición de los Servicios Web en ejecución.
- Diseñar un mayor número de escenarios o contextos en las diferentes arquitecturas, para observar el comportamiento de las métricas y determinar la arquitectura con mejores factores de comportamiento ante los escenarios.
- Incluir otras métricas de calidad estáticas al Marco Orientado a Objetos para la Medición de Calidad de Arquitecturas de Software.

Referencias

Al-Masri, E., & Mahmoud, Q. H. (2007). QoS-based discovery and ranking of Web services. *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, 529–534. https://doi.org/10.1109/ICCCN.2007.4317873

Alonso, G., Casati, F., Kuno, H., Machirajuu, V., & Verlag, S. (2004). Web Services. *Web Services - Concepts, Architectures and Applications*, (Chapter 1), 124–151. https://doi.org/10.1007/978-1-4302-0797-9_18

Amarjeet, & Chhabra, J. K. (2017). Improving modular structure of software system using structural and lexical dependency. *Information and Software Technology*, *82*, 96–120. https://doi.org/10.1016/j.infsof.2016.09.011

Arcelli, D., Cortellessa, V., & Di Pompeo, D. (2016). Automated translation among EPSILON languages for performance-driven UML software model refactoring. *Proceedings of the 1st International Workshop on Software Refactoring - IWoR 2016*, 25–32. https://doi.org/10.1145/2975945.2975951

Arcelli, D., Cortellessa, V., & Di Pompeo, D. (2018). Performance-driven software model refactoring. *Information and Software Technology*, *95*(September 2017), 366–397. https://doi.org/10.1016/j.infsof.2017.09.006

Arcelli, D., & Di Pompeo, D. (2017). Applying Design Patterns to Remove Software Performance Antipatterns: A Preliminary Approach. *Procedia Computer Science*, *109*(2016), 521–528. https://doi.org/10.1016/j.procs.2017.05.330

Barnes, D. J., & Kölling, M. (2007). *Programación orientada a objetos con Java Una introducción práctica usando BlueJ* (03 Edition). Madrid: Pearson Prentice Hall.

Brito e Abreu, F., & Melo, W. (2002). Evaluating the impact of object-oriented design on software quality. *Software Metrics Symposium, 1996., Proceedings of the 3rd International*, 90–99. https://doi.org/10.1109/metric.1996.492446

Candela, I., Bavota, G., Russo, B., & Oliveto, R. (2016). *Using Cohesion and Coupling for Software Remodularization : Is It Enough ? 25*(3), 1–28.

Cerami, E. (2002). *Web Services Essentials* (First Edit). O'Reilly.

Chhabra, J., & Gupta, V. (2010). A survey of dynamic software metrics. *Journal of Computer Science and Technology*, *25*(5), 1016–1029. https://doi.org/10.1007/s11390-010-1080-9

Clements, P. C. (1995). Component-based software engineering. *American Programmer*, *8*.

Cortellessa, V., Di Marco, A., & Trubiani, C. (2014). An approach for modeling and detecting software performance antipatterns based on first-order logics. *Software and Systems Modeling*, *13*(1), 391–432. https://doi.org/10.1007/s10270-012-0246-z

Deitel, P., & Deitel, H. M. (2008). *Como programar en Java* (Pearson Ed; L. Mi. Cruz, Ed.). México.

Diaz, A. (2009). *Diseño estadístico de experimentos* (2º Edición).

Dudney, B., Asbury, S., Krozak, J., & Wittkopf, K. (2003). *J2EE AntiPatterns* (Wiley Publ;

R. Elliot, Ed.). Indianapolis, Indiana: Robert Ipsen.

Fenton, N. E., & Ohlsson, N. (2000). Software Measurement: A Necessary Scientific Basis. *IEEE Transactions on Software Engineering*, *26*(8), 797–814.

Foundation, A. S. (2019). Apache JMeter. Retrieved May 2, 2019, from https://jmeter.apache.org/

Fowler, M. (1997). A Survey of Object Oriented Analysis and Design Methods. *Software Engineering, 1997., Proceedings of the 1997 International Conference on*. https://doi.org/10.1145/253228.253783

Fuggetta, A. (2000). Software Process: A Roadmap. *Proceedings of the Conference on The Future of Software*, *11*(1), 25–34.

Gallardo, S. (2018). *Generador de Servicios Web desde Marcos Orientados a Objetos con el Enfoque de Clases Internas*. Centro Nacional de Inverstigación y Desarrollo Tecnológico.

Garavan, H., & Murphy, K. (2016). Experimental design. *Neuromethods*, *119*, 137–153. https://doi.org/10.1007/978-1-4939-5611-1_5

Garlan, D., & Shaw, M. (2008). Introduction to software architecture. *Advanced Topics in Science and Technology in China*, (January), 1–33. https://doi.org/10.1007/978-3-540-74343-9_1

Guadarrama Rogel, L. C. (2013). *Estudio Experimental para determinar la relación entre la estructura interna de Servicios Web y valores de QoS*. CENIDET.

Gunther, N. J. (1998). *The Practical Performance Analyst* (iUniverse).

Henderson-Sellers, B. (1996). *Object-Oriented Metrics Measures of Complexity.*

Ieee. (1990). IEEE Standard Glossary of Software Engineering Terminology. *Office*, *121990*(1), 1. https://doi.org/10.1109/IEEESTD.1990.101064

Král, J., & Žemlička, M. (2009). Popular SOA antipatterns. *Computation World: Future Computing, Service Computation, Adaptive, Content, Cognitive, Patterns, ComputationWorld 2009*, 271–276. https://doi.org/10.1109/ComputationWorld.2009.80

Legorreta, R. N. E. (2017). *Transformación de Marcos de Aplicaciones Orientados a Objetos hacia Marcos con Arquitectura Orientada a Servicios*. Tesis de maestría - CENIDET.

León, F. (2009). *Generación de Servicios Web desde Marcos Orientados a Objetos a partir de Clases Colaborativas en Casos de Uso (MOOaSW)*. Tesis de maestría - CENIDET.

Mansoor, U., Kessentini, M., Wimmer, M., & Deb, K. (2017). Multi-view refactoring of class and activity diagrams using a multi-objective evolutionary algorithm. *Software Quality Journal*, *25*(2), 473–501. https://doi.org/10.1007/s11219-015-9284-4

Maria, F., & Brosig, K. (2014). *escartes Architecture-Level Software Performance Models for Online Performance Prediction*. *90*, 71–92.

Mišić, V. B. (2000). Coherence equals cohesion - Or does it? *Proceedings - Asia-Pacific Software Engineering Conference, APSEC, 2000–Janua*, 465–469. https://doi.org/10.1109/APSEC.2000.896735

Parr, T. (2014). ANTLR.

Pfleeger, S. L. (1995). Experimental Design and Analysis in Software Engineering Part 5: Analyzing the Data. *ACM SIGSOFT Software Engineering Notes*, *20*(5), 14–17. https://doi.org/10.1007/BF02249052

Rathee, A., & Chhabra, J. K. (2018). Improving Cohesion of a Software System by Performing Usage Pattern Based Clustering. *Procedia Computer Science*, *125*, 740–746. https://doi.org/10.1016/j.procs.2017.12.095

Red Hat, I. (2017). WildFly. Retrieved May 2, 2019, from https://wildfly.org/

Rodríguez, E. A. (2005). *Metodología de la investigación* (Universida). México.

Serrano, R. (2003). *Introducción al análisis de datos experimentales* (Universita).

Trubiani, C., Bran, A., van Hoorn, A., Avritzer, A., & Knoche, H. (2018). Exploiting load testing and profiling for Performance Antipattern Detection. *Information and Software Technology*, *95*(March 2017), 329–345. https://doi.org/10.1016/j.infsof.2017.11.016

Trubiani, C., Koziolek, A., Cortellessa, V., & Reussner, R. (2014). Guilt-based handling of software performance antipatterns in palladio architectural models. *Journal of Systems and Software*, *95*, 141–165. https://doi.org/10.1016/j.jss.2014.03.081

Vivanco, M. (2005). *Muestreo Estadístico Diseño y Aplicaciones* (Editorial). Santiago de Chile.

Vivona, I. (2011). *Java Domine el lenguaje líder en aplicaciones cliente-servidor*. Buenos Aires: Fox Andina.

Walpole, R. E. (1999). *Probabilidad y Estadística para Ingenieros* (6a, Ed.). México: Prentice-Hall Hispanoamericana.

Walpole, R. E., Myers, R. H., & Myers, S. L. (2012). *Probabildad y estadística para ingeniería y ciencias* (Pearson No; G. Lopez, Ed.). México.

Wang, H., Kessentini, M., Hassouna, T., & Ouni, A. (2017). On the Value of Quality of Service Attributes for Detecting Bad Design Practices. *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017*, 341–348. https://doi.org/10.1109/ICWS.2017.126

Xu, J., Zheng, Z., & Lyu, M. R. (2016). Web Service Personalized Quality of Service Prediction via Reputation-Based Matrix Factorization. *IEEE Transactions on Reliability*, *65*(1), 28–37. https://doi.org/10.1109/TR.2015.2464075

Zubillaga, A. (2006). *Análisis estadístico de los datos*. (1), 1–27. https://doi.org/10.1016/j.ces.2007.08.075

Anexos

Anexo A: Pruebas adicionales de tiempo de respuesta

Adicionalmente a las pruebas con el tiempo de respuesta de los Servicios Web, cuyas mediciones se dieron en milisegundos, se realizaron pruebas midiendo los tiempos en nanosegundos, con el propósito de observar más claramente las diferencias de los tiempos en las diferentes arquitecturas que se consideraron en este trabajo.

A continuación, se muestran los datos obtenidos en estas pruebas.

Arquitecturas que se utilizaron en las pruebas	
A1	Conjunto de clases relacionadas
A2	Una sola clase
A3	Clases internas

1.- Tablas de multiplicar		
A1	A2	A3
6674498	205108	6336763
8036230	205108	6215960
6623092	226185	6286900
6708425	379373	6375831
6556265	398907	7333003
7209630	273991	6482241
6645196	216932	7616248
6785019	295068	6232923
7850657	299694	6324939
10653801	332594	6540843
73742813	2832960	65745651

2.- Parámetros de Regresión		
A1	A2	A3
661545303	607794624	616713489
642436315	618536330	624354925
654002565	601882471	633458334
658767346	600361381	635414316
661062604	623451215	623962186
667563865	612960883	626495966
651579307	618573342	631061293
662985686	606391253	629850179
651711420	613272915	615934180
651711420	612187229	625972657
6563365831	6115411643	6263217525

3.- Identifica Palabras		
A1	A2	A3
35817336	34696694	37732707
62864883	45410640	43182723
39320624	32453870	37980481
37372868	32867170	40711659
39235291	33572969	38284802
38161429	38018007	43506579
36899422	35758219	38336208
60024212	31701293	39867579
39590504	35882107	38856432
40587257	39032239	38099743
429873826	359393208	396558913

4.- Agrupa Palabras		
A1	A2	A3
25586602	25213912	42596186
28285908	24138508	26716496
29531978	25285365	26579757
28698694	25839517	35982347
25504867	22139345	39441941
27943547	25315180	28958806
29041055	25991163	28800991
33767282	23975037	27592446
25943870	21979474	30005937
27295323	27239290	38038570
281599126	247116791	324713477

5.- Operaciones Aritméticas		
A1	A2	A3
4971945	3088446	5105086
4789970	3446229	5300941
5255190	3128543	5171913
4858339	3295096	5022323
4882500	3294069	4945729
5290145	3085875	5052138
5077327	3352156	4858339
4936475	3156301	5015640
5076813	3124430	5186821
4950869	3448799	4914885
50089573	32419944	50573815

6.- Medidas Cono		
A1	A2	A3
5262387	3130085	5312251
5264443	3284301	5580073
5414033	3546469	5389358
5599093	3120317	5400154
5497310	3204622	5236684
5269070	3085876	5358001
5655125	3128542	5358001
5345663	3717136	5361086
5274724	3215932	5286033
5362628	3499176	5493711
53944476	32932456	53775352

7.- Consulta Restaurantes de BD		
A1	A2	A3
175707124	177351077	184569974
187707261	184157701	181401843
194023474	189514170	180496075
188928145	186131678	187977655
185403260	188287117	189966540
185187870	183400495	185394007
185439757	183413346	188419230
187581317	182832462	190874364
182072687	184740126	178717953
185920915	184841396	181114485
1857971810	1844669568	1848932126

8.- Consulta Personas de BD		
A1	A2	A3
154524906	151773676	153227427
156145213	153511186	155998707
162398711	156630996	156315880
160405714	162881925	152677387
154743894	153323042	154039123
155924168	152832632	155785887
157559382	155242015	157220619
151478607	153310704	156104088
158747368	151741804	150536855
158747368	152179780	157927961
1570675331	1543427760	1549833934

9.- Inserta Personas a BD		
A1	A2	A3
223986807	212274542	245831129
213535522	223041457	217596055
238731493	219881551	210577641
213815169	208781012	227393460
260479172	207522602	214724535
237700296	226673268	231047375
223590984	219642000	232932935
224283418	234732647	226665557
236188969	212848742	224530679
231382025	222318694	211821143
2303693855	2187716515	2243120509

10.- Desviación estándar		
A1	A2	A3
1618764	509430	1705126
1818218	631261	1622877
1604371	417414	1635728
1724146	474988	1676339
1567358	399422	1699471
1607455	396851	1704098
1547825	495037	1518523
1520579	404562	1758588
1468146	399936	1741624
1560675	463165	1531374
16037537	4592066	16593748