UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM VALLE DE MÉXICO

Evaluación y análisis en la calidad en el diseño de sistemas de software orientados a objetos

TESIS

Que para obtener el Grado de

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

P r e s e n t a

Ing. Abraham Banda Madrid

Tutor Académico: Dra. en C. Leticia Dávila Nicanor
Tutores Adjuntos: Dra. en C. María de Lourdes López García CU UAEM Valle de Chalco
Dra. En C. Com. Maricela Quintana López CU UAEM Valle de México

Atizapán de Zaragoza, Edo. de Méx. Septiembre de 2020

CU UAEM
VALLE DE MÉXICO

ABSTRACT

In the area of Software Engineering, the development of test case prioritization models is mainly focused on the system testing stage, however, this work addresses a new way of establishing the prioritization of failures in Object-oriented-systems using a simple, adaptable and applied methodology in the early stages of the software development process.

The proposed methodology is mainly based on contemplating design attributes and software metrics that provide true and reliable information about the system design, class and sequence diagrams provide this type of information, since they represent its architecture and its operations, making them the essential means for the development of the model.

The application of the mathematical technique of graphs provides an ease of use of the methodology, this technique is capable of interpreting the class and sequence diagrams in a simple and precise way, together with the inclusion of rules or conditions for decision making, it makes the process to follow coherent and the results obtained are applied in later stages of development.

RESUMEN

En la Ingeniería de Software el desarrollo de los modelos de priorización de casos de prueba está enfocados principalmente en la etapa de pruebas del sistema, sin embargo, el presente trabajo aborda una nueva forma de establecer la priorización de fallos en los sistemas de software orientados a objetos utilizando una metodología simple, adaptable y aplicada en fases tempranas del proceso de desarrollo de software.

La metodología propuesta se basa principalmente el contemplar atributos de diseño y métricas de software que proporcionen información veraz y confiable sobre el diseño del sistema, los diagramas de clases y de secuencias proporcionan este tipo de información, dado que en ellos se representa su arquitectura y sus operaciones, haciéndolos los medios esenciales para el desarrollo del modelo.

La aplicación de la técnica matemática de grafos provee una facilidad de uso de la metodología, dicha técnica es capaz de interpretar los diagramas de clases y de secuencias de forma sencilla y precisa, junto con la inclusión de reglas o condiciones para la toma de decisiones, hace que el proceso a seguir sea coherente y los resultados obtenidos sean aplicados en fases posteriores de desarrollo.

Índice General

Índice de figuras

Índice de tablas

Índice de ilustraciones

1. Introducción

La priorización de los casos de prueba en los sistemas de software consiste en establecer un orden de ejecución de pruebas unitarias, de integración o de sistema, por medio del cual se alcance un objetivo en el proceso de desarrollo de software, este objetivo puede ser: la aceleración de detección de fallos, la aceleración de detección de fallos críticos, acelerar la cobertura del código o minimizar los costos asociados a las pruebas (Sánchez, 2013).

La cobertura en la fase de pruebas de los sistemas de software es necesaria para determinar que el producto de software resultante de todo el desarrollo cumpla con la funcionalidad, los estándares de calidad, los atributos de diseño especificados en su desarrollo y una ejecución exitosa para el uso del usuario final. En dado caso de que esta fase no sea llevada a cabo de forma adecuada o quede incompleta, el producto de software tendrá irregularidades y provocará pérdidas de recursos monetarios y/o humanos por parte de la organización para la cual fue desarrollada y los involucrados en la creación del software.

1.1. Antecedentes

A lo largo de la historia, los fallos en los sistemas de software han ocasionado grandes pérdidas económicas, desprestigio de los desarrolladores y en ocasiones incluso pérdidas humanas. Esto se debe a que, al desarrollar un sistema de software con una fase de pruebas inadecuada, no es posible identificar los fallos latentes a tiempo, lo que podría provocar un mal funcionamiento del sistema y/o no se cumpliría con la especificación para la cual fue desarrollado.

Cuando la fase de pruebas no es realizada de forma adecuada, no se garantiza que el producto desarrollado tenga óptimas condiciones de uso y se realice mantenimiento al mismo de forma constante. A continuación, mencionaremos algunos de los casos más representativos de esta índole:

- El Therac X-25 (Nancy Leveson, 1993), era un equipo de laboratorio médico desarrollado para obtener imágenes de rayos X y proporcionar terapia radioactiva, este equipo costó alrededor de un millón de dólares. Dicha máquina provocó quemaduras de diferente grado y en casos extremos la muerte a los pacientes que fueron tratados con ella. Los operadores que manipulaban la máquina verificaban que los datos introducidos eran correctos al momento de realizar un procedimiento y en caso contrario modificaban los datos para poder suministrar la dosis

correcta en los pacientes. Al investigar el porqué de los accidentes provocados por la máquina, se encontró que la interfaz de usuario de la máquina permitía al operador proporcionar dosis de radiación mortal. Otro fallo identificado fue que no se realizaba el cambio en los filtros para la aplicación de la radiación y que al realizar un cambio de datos en la dosis de radiación no se actualizaban los datos para la aplicación del procedimiento. Eso fue a causa de que los desarrolladores no previeron el error humano y tomar en cuenta un sistema de alertas al momento de ingresar los datos de las dosis de radiación. Por otra parte, en la documentación no se encontró registro de pruebas de actualización de datos y no existía un subsistema de recuperación de errores, en otras palabras, no se llevó a cabo una fase de pruebas adecuada para detectar fallas en el sistema.

- Un caso más actual, fue el de la sonda espacial Mars Climate (Valenzuela, 1999), la cual fue desarrollada a petición de la NASA por 2 compañías estadounidenses, con el propósito de estudiar la superficie del planeta Marte en búsqueda de señales de vida y agua. El Jet Propulsion Laboratory de Pasadena fue el encargado de programar los sistemas de navegación de la sonda y el Lockheed Martin Astronautics de Denver, diseñó y construyó la Mars Climate Observer. Sin embargo, la sonda espacial al llegar a su destino se estrelló en la superficie marciana, lo cual provocó una investigación de la causa del fallo en el sistema de software, al indagar en la causa del problema se encontró que las compañías desarrolladoras utilizaban sistemas numéricos diferentes, lo que ocasionó que el sistema de navegación de la sonda realizará cálculos erróneos al momento del aterrizaje; el Jet Propulsion Laboratory ocupaba el sistema métrico decimal y el Lockheed Martin Astronautics utilizaba el sistema anglosajón de medidas. Lo cual fue un error grave de comunicación y provocó pérdidas millonarias para los Estados Unidos de América.

1.2. Planteamiento del problema

En la actualidad, los modelos que establecen la priorización de los casos de prueba de los sistemas de software orientados a objetos tienen la premisa de tener un bajo costo computacional y una alta validez predictiva en contextos reales. Es necesario el desarrollo de un nuevo modelo que permita localizar y priorizar las rutas subyacentes de operación en la arquitectura del sistema evaluado, aplicado en fases tempranas del proceso de desarrollo de software, por medio del cual se pueda planificar una cobertura de pruebas adecuada para el sistema en desarrollo, esto

es debido a que los modelos existentes se desarrollan principalmente para evaluar productos específicos.

1.3. Motivación

En la mejora de la calidad de los sistemas de software, principalmente en los orientados a objetos, la mayor parte de los modelos de priorización de pruebas en el software están orientados a la fase de pruebas del sistema, es decir, operan con el sistema en ejecución, sin embargo, este enfoque tiene costos muy altos (I.Sommerville, 2011).

De acuerdo al estudio del SEI-Carneige and Mellon University (Park, Goethert, & Florac, 1996), indica que en un proceso de desarrollo de software, el origen de los fallos del sistema se concentra en un 40% en la fase de diseño, lo cual genera que estos se propaguen y sean detectados en fases posteriores en el proceso de desarrollo, por lo cual, atender esta etapa de forma temprana optimizaría el uso de los recursos y disminuiría el costo de prueba y mantenimiento del sistema.

Para realizar el análisis y detectar fallos en un sistema de software, es necesario aplicar una serie de métricas desde fases iniciales como la etapa de diseño, de tal manera que la arquitectura del sistema pueda ser analizada y con ello identificar zonas críticas y rutas de operación que tengan niveles de riesgo de desencadenar un fallo, provocando una mala ejecución de las operaciones del sistema en fases futuras.

1.4. Hipótesis

Mediante el análisis de los atributos de diseño: complejidad y acoplamiento, de un sistema de software orientado a objetos y métricas de software, será posible analizar la arquitectura del sistema, aplicando un modelo matemático para establecer la priorización de casos de pruebas, la cual será la base para estimar niveles de riesgo en rutas y zonas de operación de los casos de prueba durante la operación del sistema. Con ello se podrá optimizar los recursos dispuestos a la fase de pruebas y localizar fallos antes de la implementación del código.

1.5. Objetivo General

Mejorar la efectividad en la priorización de casos de prueba en los sistemas de software orientados a objetos, mediante el análisis de las propiedades de diseño: Complejidad y Acoplamiento a través de una metodología basada en la técnica de grafos, ponderados con métricas de software.

1.6. Objetivos Específicos

- Establecer las métricas de software más precisas para los atributos de diseño del software: complejidad y acoplamiento.
- Sistematizar los cálculos en las entradas y salidas del modelo mediante el análisis y evaluación de las técnicas y herramientas más usadas en el ámbito científico.
- Diseñar un modelo matemático con entradas naturales de la fase de diseño (diagramas, modelos, etc.) así como un análisis que permita estimar todo el conjunto de casos de prueba.
- Determinar un algoritmo que permita localizar todos los casos de prueba con un costo computacional justificado en relación con el sistema en estudio.
- Implementar el modelo desarrollado en dos casos de estudio independientes.
- Analizar los resultados obtenidos mediante el modelo desarrollado para establecer factores de riesgo.
- Determinar las secuencias de operación y zonas de la arquitectura del software subyacentes en el código, que posean un alto nivel de riesgo de fallo en su futura operación.

1.7. Metodología

Para el desarrollo de este trabajo se siguió el siguiente procedimiento:

I. Revisión del estado del arte sobre los atributos de complejidad y acoplamiento y sus métricas de implementación.
II. Sistematización de los cálculos de métricas establecidas: Investigación y experimentación de las herramientas de software existentes.
III. Desarrollo de la propuesta para generar un modelo matemático desde el enfoque de la cobertura, tomando en cuenta la arquitectura del sistema.
IV. Investigación de los algoritmos más adecuados para analizar el modelo propuesto.
V. Aplicación del modelo desarrollado en un primer caso de estudio de baja escala.
VI. Interpretación de los resultados del modelo en el primer caso de estudio.

VII. Aplicación del modelo desarrollado en un segundo caso de estudio de mediana escala.
VIII. Interpretación de los resultados del modelo en el segundo caso de estudio.
IX. Análisis de la efectividad del modelo en los casos de estudio.
X. Desarrollo de conclusiones generales y trabajo a futuro.

1.8. Organización del documento

El contenido del documento está distribuido de la siguiente forma:

- En el capítulo 2 se presenta el marco conceptual sobre el proceso de desarrollo de sistemas de software y su calidad.
- En el capítulo 3 se mencionan trabajos que han sobresalido en el estado del arte, así como, la comparación de los enfoques que se han propuesto para el análisis del diseño de los sistemas de software y su aplicación.
- En el capítulo 4, se presenta la metodología propuesta para el desarrollo de un nuevo modelo de priorización de casos de prueba, los experimentos y los resultados obtenidos.
- En el capítulo 5 se listan las conclusiones, el trabajo a futuro, las recomendaciones y las restricciones acerca de la investigación desarrollada.

2. Marco Teórico

En este capítulo se presenta el marco conceptual del desarrollo de sistemas. De tal manera que se tenga un conocimiento general de los conceptos básicos y particulares utilizados en este trabajo.

2.1. Ingeniería de software

La Ingeniería de software es una disciplina de ingeniería enfocada en el estudio de la producción del software, desde la primera etapa de especificación del sistema hasta la fase de mantenimiento. Dicha disciplina no abarca únicamente los procesos técnicos del desarrollo del software, considera también actividades como la administración del proyecto de software, el desarrollo de herramientas, métodos y teorías para el apoyo en la producción de software.

La Ingeniería de software es importante por dos razones (I.Sommerville, 2011):

- Actualmente la sociedad y los individuos se apoyan en sistemas de software, para lo cual se requiere una producción económica y rápida de sistemas confiables.
- El uso de métodos y técnicas de ingeniería de software resulta ser más barato a largo plazo, que solo diseñar los programas como si fuera un proyecto de programación personal. Debido a que los requerimientos pueden cambiar la función del software después de ponerlo en operación.

El desarrollo de un software sigue un proceso o una serie de actividades relacionadas entre sí que conducen a la elaboración de un producto de software, estas son cada una de las fases de desarrollo y están presentes en diversas formas de desarrollo de los productos de software las cuales se conocen como modelos de desarrollo.

2.2. Proceso de desarrollo de software

Existen diferentes procesos de software, que debe incluir las siguientes actividades que son fundamentales para la Ingeniería de software:

- Especificación del software: tiene que definirse la funcionalidad del software como las restricciones de superación.
- Diseño e implementación del software: debe desarrollarse el software para cumplir con la especificación de requerimientos.
- Validación del software: hay que validar el software para asegurarse de que cumple su especificación.

- Evolución del software: el software tiene que evolucionar para satisfacer las necesidades cambiantes del cliente.

Tales actividades forman parte de todos los procesos de software, sin embargo, en el aspecto práctico estas son actividades complejas en sí mismas ya que incluyen subactividades. También existen actividades de soporte al proceso, como la documentación y el manejo de la configuración del software.

2.3. Modelos de proceso de software

Un modelo de proceso de software es una representación simplificada de este proceso. Cada modelo del proceso es representado desde una particular perspectiva, por lo tanto, ofrece información parcial acerca de dicho proceso. A continuación, se examinan los modelos del proceso más representativos de la Ingeniería de software:

1. *El modelo en cascada*: este modelo toma las actividades fundamentales, luego las representa como fases separadas del proceso, tal como análisis y especificación de requerimientos, diseño del sistema, implementación, pruebas y mantenimiento, representado por la Figura 2.1 obtenida de (I.Sommerville, 2011).

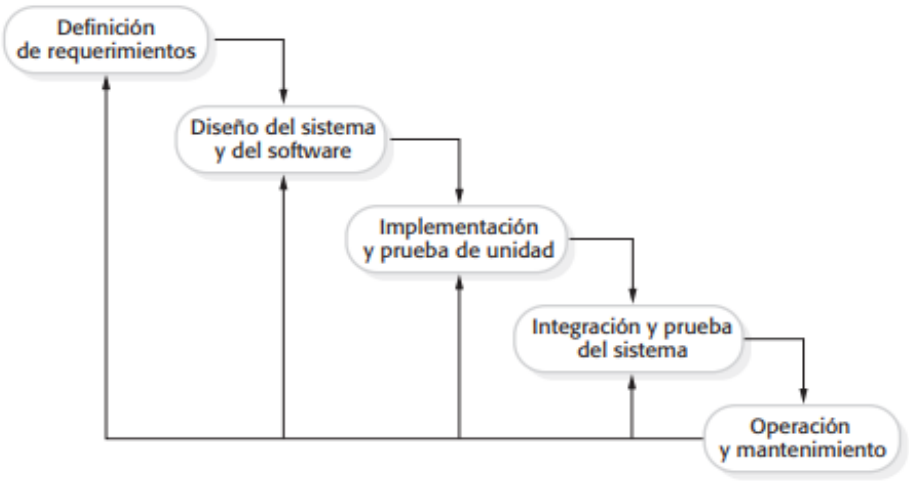

Figura 2.1Modelo en cascada

2. *Desarrollo incremental*: este enfoque vincula las actividades de especificación, desarrollo y validación. El sistema se desarrolla como una serie de versiones y cada versión añade funcionalidad a la versión anterior, la Figura 2.2 obtenida de (I.Sommerville, 2011) muestra de forma general los procesos para este modelo.

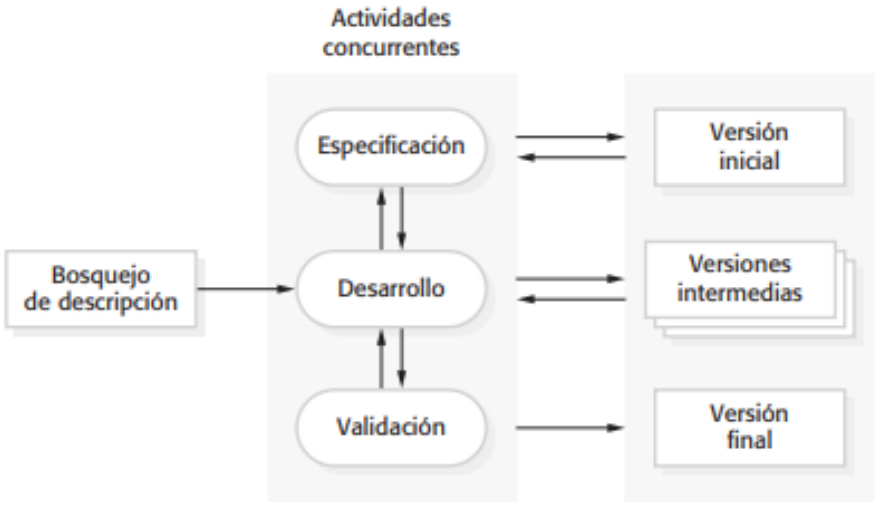

Figura 2.2 Modelo incremental

3. *Ingeniería de software orientada a la reutilización*: este enfoque se basa en la existencia de un número significativo de componentes reutilizables. El proceso de desarrollo del sistema se enfoca en la integración de estos componentes en un sistema, en lugar del desarrollo desde cero. La presenta la serie de pasos que se siguen para la reutilización. Los pasos por seguir son los representados en la Figura 2.3 obtenida de (I.Sommerville, 2011).

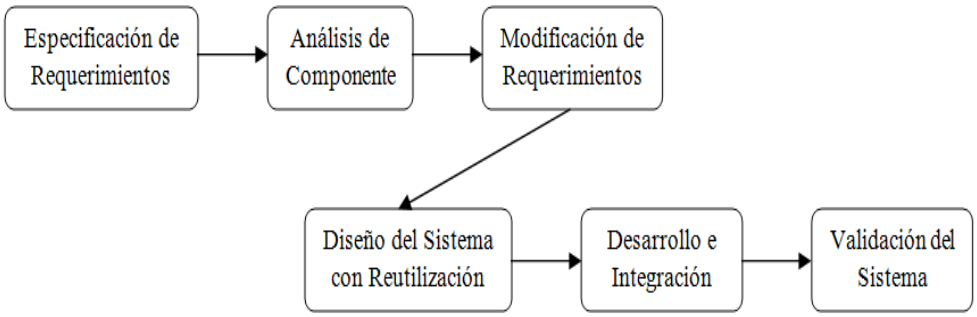

Figura 2.3 Ingeniería de software orientada a la reutilización

4. *Modelo en espiral*: el proceso del software se representa con una espiral, y no como una secuencia de actividades con cierto retroceso de una actividad a otra, cada ciclo en espiral representa una fase del proceso de software. Dicho modelo combina el evitar el cambio con la tolerancia al cambio, representado en la Figura 2.4 obtenida de (I.Sommerville, 2011).

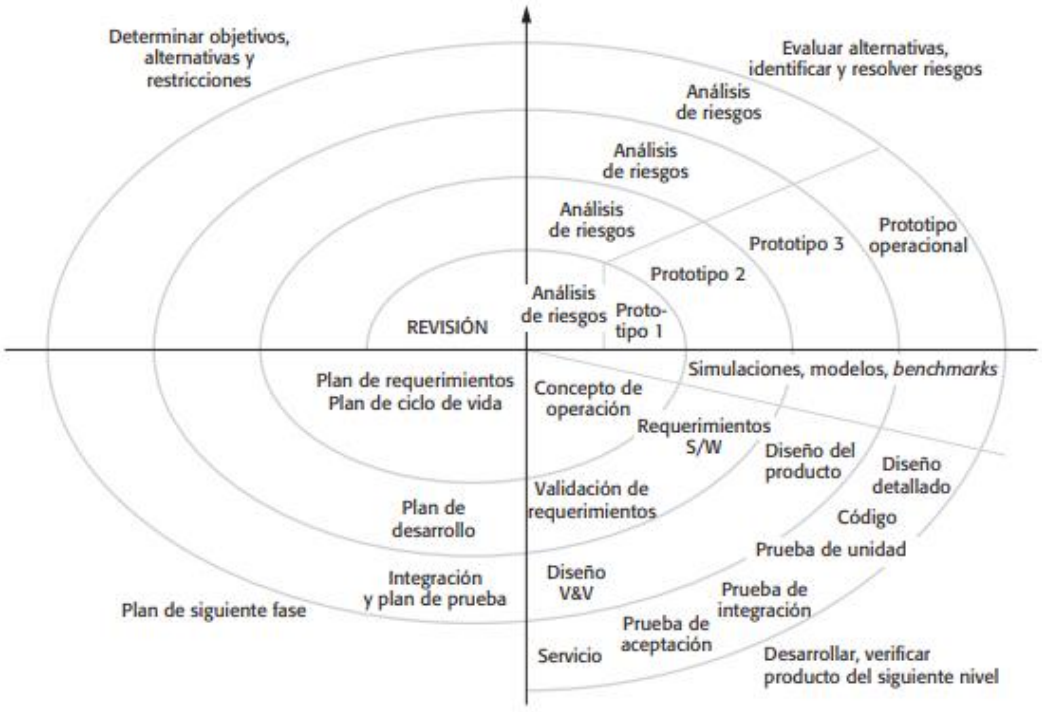

Figura 2.4 Modelo en espiral

2.4. Fase de diseño en el proceso de desarrollo de software

Para desarrollar la fase de diseño en el proceso de desarrollo de software, se utilizan herramientas de modelado para abstraer su naturaleza y establecer de manera general la propuesta de solución a los requerimientos del sistema, este proceso sirve de guía para la fase de implementación del sistema y en esta misma se establece la funcionalidad que será integrada al sistema en desarrollo.

La modelación es usada comúnmente en la ciencia e ingeniería para proveer abstracciones de un sistema en un nivel de precisión y detalle. De acuerdo con la OMG (Object Modeling Group), modelación es el diseño de aplicaciones de software antes de la codificación.

Un método de diseño de software (Gomaa, 2013), es un enfoque sistemático que describe la secuencia de pasos a conocer para crear un diseño, dados los requerimientos del software de la aplicación. Esto ayuda al diseñador o a un equipo de diseño de software a identificar qué debe hacerse para poder conocer la estructura del sistema que se vaya a desarrollar.

En los sistemas de software orientados a objetos, el entendimiento de los conceptos básicos del paradigma es de gran importancia para el análisis y diseño del software, porque a través de ellos se puede adaptar, modificar y evolucionar el

software. De acuerdo con la OMG, UML (Unified Modeling Language) es un estándar y es uno de los métodos de análisis y diseño orientados a objetos, UML fue desarrollado para proveer un lenguaje gráfico normalizado y con notaciones para describir un modelo orientado a objetos.

2.5. UML

La notación UML está dada por los diagramas de casos de uso, clases, colaboración, secuencia, actividades y estado, para el desarrollo de aplicaciones (Gomaa, 2013) (Jim Arlow, 2005) (J. Rumbaugh, 2000), esta se representa a continuación:

- Diagramas de casos de uso

Un caso de uso es definido como una secuencia de interacciones entre un actor y el sistema. Un actor inicializa un caso de uso y puede realizar diferentes tareas. El actor es representado mediante una figura en líneas y círculos de una persona, un caso de uso con una elipse, el sistema con una caja. Las relaciones entre casos de usos están definidas de dos tipos: include y extend. La notación para cada elemento de un diagrama de caso de uso se muestra en la Figura 2.5 obtenida de (J. Rumbaugh, 2000).

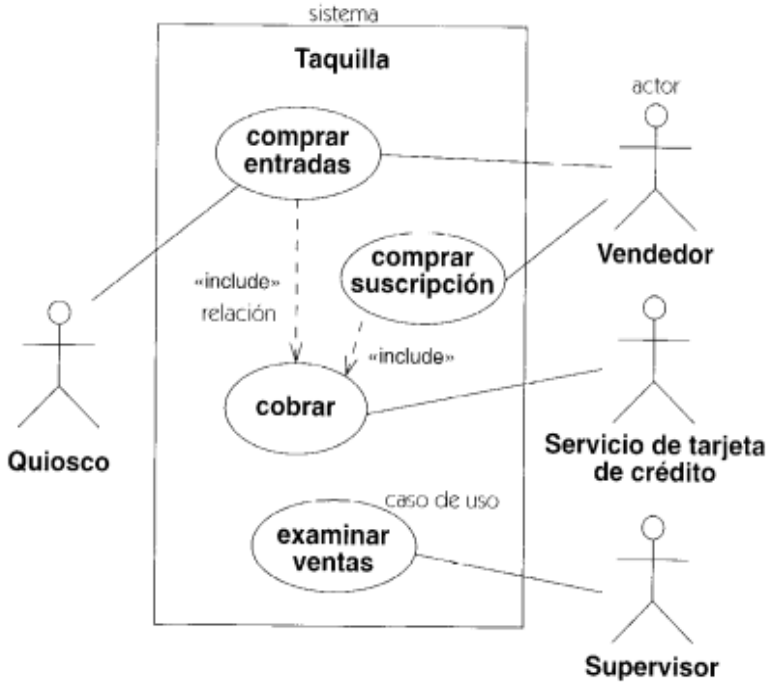

Figura 2.5. Ejemplo de un diagrama de casos de uso

- Diagramas de clases

En un diagrama de clases, las clases son representadas con una caja y las relaciones entre ellas son representadas mediante líneas de conectores. Existen tres principales tipos de relación entre clases: asociaciones, agregaciones/composiciones y generalizaciones/especializaciones.

Una asociación es una relación estructural entre dos o más clases. Una asociación entre dos clases, se le llama asociación binaria, la cual es representada mediante una línea. Una asociación tiene un nombre y opcionalmente una pequeña flecha que representa la dirección que toma. Al final de la línea de asociación en cada clase se describe la multiplicidad de asociación, la cual indica cuantas estancias de una clase están relacionadas con una instancia de la otra clase.

La jerarquía de agregaciones y composiciones son relaciones todo/parte. La relación de composición (representada como un diamante negro) es aquella con la cual se establece que una clase está compuesta de otra y esta no puede existir sin que exista un objeto de la clase con la que se relaciona. La relación de agregación (representada mediante un diamante sin color) es aquella con la cual se establece que una clase puede o no contener un objeto de otra clase y esta puede existir sin que exista un objeto de la clase con la que se realiza este tipo de relación.

Una generalización/especialización indica relaciones de herencia entre las clases. Es representada mediante un triángulo en la superclase (la clase general), y una línea simple a la/las clases que se derivan de ella. Todo lo anterior es denotado en la Figura 2.6 obtenida de (J. Rumbaugh, 2000).

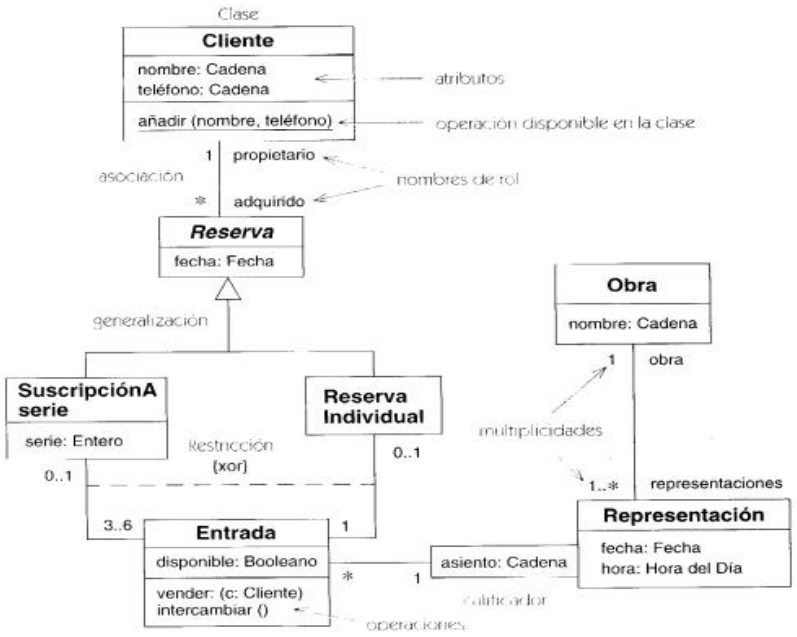

Figura 2.6 Ejemplo de un diagrama de clases

- Diagramas de comunicación

Este tipo de diagramas muestra la cooperación dinámica de los objetos con otros, mediante el envío y recepción de mensajes. El diagrama representa la organización estructural de los objetos que interactúan. Los objetos se representan mediante cajas y las líneas que unen las cajas representan interconexión de objetos. Una flecha y su etiqueta representan el nombre y la dirección del mensaje entre los objetos. La secuencia de mensajes entre objetos es numerada y se representan de acuerdo con la Figura 2.7 obtenida de (J. Rumbaugh, 2000).

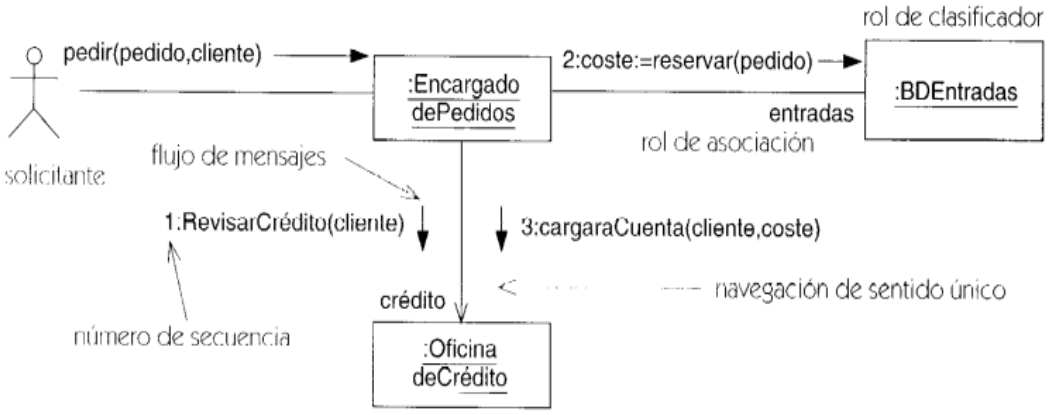

Figura 2.7 Ejemplo de un diagrama de comunicación

- Diagramas de secuencias

Un diagrama de secuencia es un diagrama en dos dimensiones en la que los objetos que participan en la interacción son representados horizontalmente y la dimensión vertical representa el tiempo. En cada cuadro de objeto hay una línea vertical discontinua, que se refiere a una línea de vida. Las flechas horizontales etiquetadas representan los mensajes, solo la fuente y el destino de la flecha son relevantes, ya que representa el objeto que envía el mensaje a otro objeto en el sistema. El espacio entre mensajes no es relevante. Los diagramas de secuencias se denotan como en la Figura 2.8 obtenida de (J. Rumbaugh, 2000).

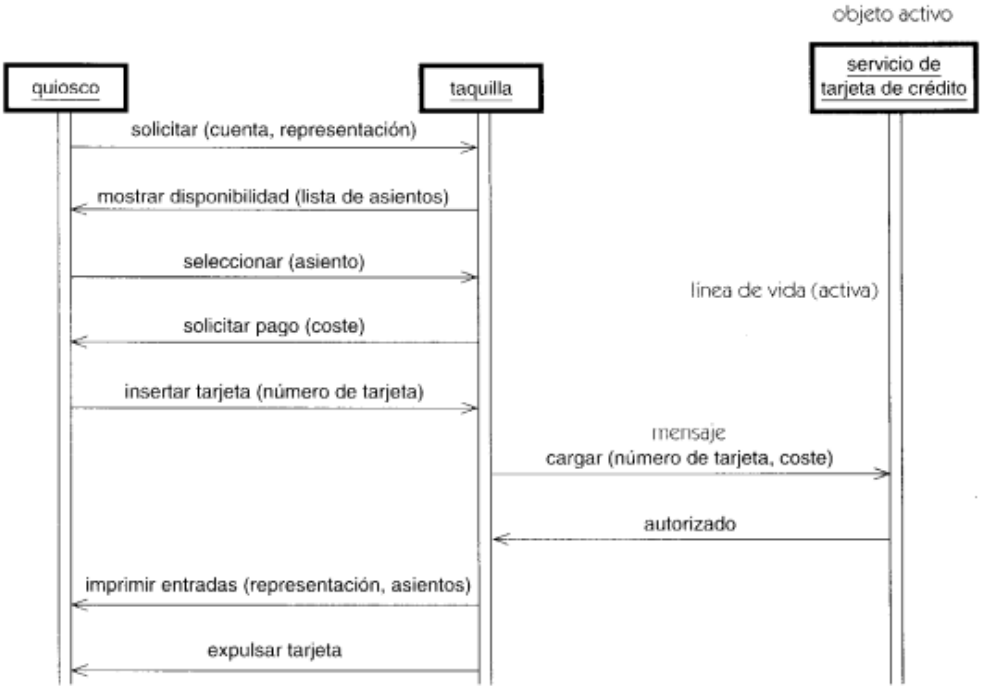

Figura 2.8 Ejemplo de un diagrama de secuencias

- Diagramas de actividades

Un diagrama de actividades representa el flujo de control, la secuencia de actividades, nodos de decisión, ciclos y las actividades concurrentes. Los diagramas de actividades son usualmente utilizados en el modelado del flujo de trabajo, por ejemplo, las aplicaciones orientadas al servicio. La notación de este tipo de diagramas se muestra en la Figura 2.9 obtenida de (J. Rumbaugh, 2000).

Figura 2.9 Ejemplo de un diagrama de actividades

- Diagramas de estado

Un diagrama de estado se refiere a un diagrama de máquinas de estados. La representación de un estado se denota por una caja redonda y las transiciones son modeladas por arcos que las conectan. El estado inicial es un círculo negro y el estado final es una circunferencia con un círculo negro al centro. Un estado puede descomponerse en sub-estados. Un evento causa la transición de un estado y la notación usada es Event [condition]/Action. La Figura 2.10 muestra un ejemplo de este tipo de diagrama obtenida de (J. Rumbaugh, 2000).

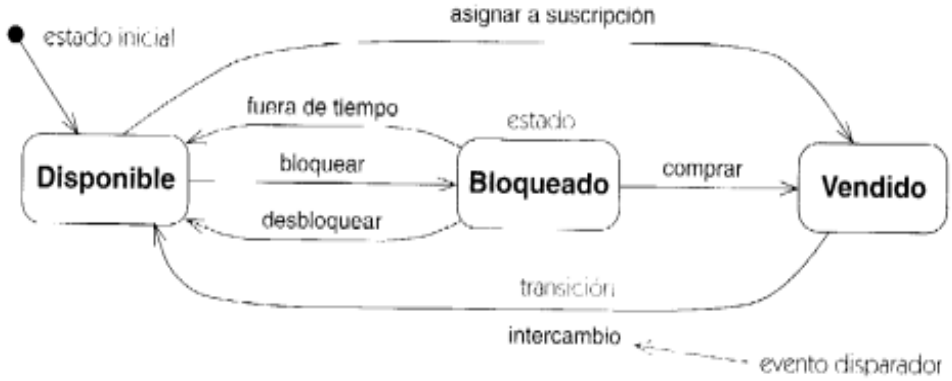

Figura 2.10. Ejemplo de un diagrama de estado

Una vez identificados los diagramas que modelan un sistema en la fase del diseño, y el estándar que se sigue a través de UML para los sistemas orientados a objetos, es necesario conocer las diferentes métricas que pueden aplicarse a estos diseños para determinar su calidad y/o obtener datos para ser utilizados en el desarrollo de modelos de confiabilidad.

2.6. Ingeniería de confiabilidad

La *Ingeniería de Confiabilidad de Software* (Musa, 2004), es una rama de la ingeniera de software que se encarga del estudio de las prácticas que permiten planear y guiar los procesos de pruebas de software de manera cuantitativa. Se divide en tres áreas:

- Tolerancia a fallos: Es esta área, se estudia en nivel de madurez del sistema y cómo reacciona ante los procesos que son mal ejecutados, es decir, a la capacidad del sistema de tolerar errores evitando un colapso total del sistema en ejecución.
- Evasión de fallos: En esta área, se estudia la forma en que un sistema pueda evadir una mala ejecución, para poder burlar la operación inadecuada y generar alternativas para recuperar la información que fue manipulada en la tarea que se realizaba.
- Detección y localización de fallos: Esta última área, se encarga del estudio de las diferentes formas de detección y localización de errores en el sistema antes de que estos mismos se produzcan, para poder generar estrategias para la corrección, evasión y eliminación de las fallas generadas en tiempo de ejecución.

2.7. Modelo de confiabilidad de software

Un concepto importante por definir para entender más adecuadamente lo que es un modelo de confiabilidad de software es el término confiabilidad de software, el cual es definido por (Musa, 2004) como la probabilidad de que un sistema de software opere libre de fallos en un lapso de tiempo determinado bajo condiciones específicas de operación.

Un modelo de confiabilidad de software (Yamada, 2014), es un modelo de análisis matemático con el propósito de medir y asentar la confiabilidad/calidad de un software de forma cuantitativa.

Un modelo de confiabilidad de software (Kan, 2003), es una herramienta que es usada para asentar la confiabilidad de un producto de software o estimar un número de defectos latentes en ejecución. Dicha estimación es importante por dos razones:

- Para estimar la calidad del producto de software.
- Para planear adecuadamente la fase de pruebas y mantenimiento del software.

Estos modelos son clasificados en dos categorías: modelos estáticos y modelos dinámicos. Un modelo estático utiliza atributos de un proyecto o un módulo del programa para determinar un número de defectos en el software. Un modelo dinámico, usualmente basado en distribuciones estáticas, utiliza el patrón de defectos del desarrollo actual para estimar la confiabilidad final del producto de software.

Un modelo de confiabilidad de software es evaluado a través de un conjunto de criterios establecidos por un grupo de expertos (Kan, 2003), dichos criterios se mencionan a continuación:

- Validez predictiva: Es la capacidad del modelo para predecir el comportamiento de los fallos o el número de defectos por un periodo de tiempo específico basado en datos actuales en el modelo.
- Capacidad: La habilidad del modelo para estimar con precisión satisfactoria las necesidades de los desarrolladores de software, ingenieros y usuarios en la planificación y gestión de proyectos de desarrollo de software o controlar el cambio en la operación del sistema del software.
- Cláusulas de calidad: El modelo debe ser consistente y lógico, desde el punto de vista de los desarrolladores y la experiencia del ingeniero de software.
- Aplicabilidad: El grado en que el modelo pueda ser aplicado en diferentes productos de software.
- Simplicidad: El modelo debe ser simple en tres aspectos: que los datos del modelo sean simples de obtener, que no se requiera de un conocimiento matemático elevado para aplicar y comprender el modelo y que sea fácilmente implementado en programas de computadora.

Desde el punto de vista práctico y con recientes observaciones en el desarrollo de los modelos de confiabilidad de software, los criterios de mayor importancia son la validez predictiva, la simplicidad y las cláusulas de calidad, en ese orden de importancia. La capacidad y la aplicabilidad son menos significantes. Esto es debido a que el objetivo principal de dichos modelos es la predicción de fallos en el sistema, dado que un modelo con buena predictibilidad, pero con poca capacidad y

aplicabilidad es ciertamente superior a un modelo con buena capacidad y aplicabilidad, pero con poca predictibilidad.

Lo anterior está basado en qué dentro del estudio del estado del arte, dichos modelos son desarrollados para proyectos de software específicos, lo cual provoca que el modelo resultante tenga poca capacidad y aplicabilidad debido a su especificación de desarrollo.

2.8. Priorización de pruebas en sistemas

El realizar las pruebas a un sistema en desarrollo es un proceso que puede durar días, semanas, meses e incluso hasta años, dependiendo del tamaño del proyecto de software en evaluación, por lo cual, existe un gran número de factores para que durante la fase de pruebas se realicen todos los casos de prueba que sean necesarios en torno al sistema evaluado.

En muchos de los casos las pruebas no son prácticamente automáticas, estás deben planearse adecuadamente para alcanzar los objetivos establecidos por los desarrolladores, por lo cual, se debe especificar los lugares y rutas de operación que se deben probar para obtener un máximo rendimiento de los recursos utilizados para el desarrollo del software. En (Sánchez, 2013), se muestra una forma de priorización de casos de prueba, estos se establecen en relación con los objetivos y estrategias a utilizar y se clasifican siguiente manera:

- Acelerar la detección de fallos: El objetivo se alcanza cuando al realizar x cantidad de pruebas se detectan fallos en el sistema.
- Acelerar la detección de fallos críticos: El objetivo a alcanzar depende del tipo de software que se está desarrollando, pueden ser fallos de acceso, funcionalidad, seguridad, etc., para que al realizar las pruebas estas detecten de forma temprana los fallos con mayor nivel de gravedad en el sistema en desarrollo.
- Acelerar la cobertura del código: El objetivo se cumple cuando con las pruebas establecidas por los desarrolladores se alcanza a depurar el código completo de la aplicación.
- Minimizar los costos asociados a las pruebas: El objetivo esencial en estos casos es minimizar el uso de los recursos en el proceso de pruebas, considerando recursos como tiempo, dinero y esfuerzo.

El trabajo propuesto presenta una forma de cumplir con los objetivos de la priorización de fallos en los sistemas de software orientados a objetos.

2.9. Paradigma orientado a objetos

El POO (Paradigma Orientado a Objetos), es un paradigma que utiliza objetos y sus interacciones, para poder diseñar programas y aplicaciones de computadoras. Tiene como base las siguientes características (Weisfeld, 2013):

- Abstracción: Es la propiedad que permite representar las características generales de un objeto sin preocuparse de la implementación de este.
- Herencia: Es la propiedad que permite a una clase, heredar los atributos y métodos a todas las subclases derivadas de una clase padre. Generalmente se usa para la especialización y generalización de objetos en el sistema.
- Encapsulación: Se define por el hecho de que los objetos contienen tanto los atributos y comportamientos, la ocultación de datos es una parte importante de la encapsulación.
- Polimorfismo: Es la propiedad de un objeto de poder realizar una tarea de diversas formas, es decir, múltiples maneras de realizar una acción, siempre y cuando el objeto pueda responder al mensaje que se le envía.

Actualmente, el POO tiene el un amplio uso en el ámbito empresarial y de investigación, (Cass, 2014) lo confirma en su artículo de los lenguajes de programación utilizados para desarrollar aplicaciones de software.

2.10. Teoría de medición

De acuerdo con (Norman Fenton J. B., 2015), medición es el proceso por el cual números o símbolos son asignados a atributos de entidades en el mundo real de tal forma que puedan describirlos de acuerdo con reglas claramente establecidas. La medición captura información acerca de los atributos de una entidad.

En Ingeniería de software, aplicamos esta teoría para describir las actividades realizadas durante el proceso de desarrollo de software, por ejemplo, el número de horas que trabajo un programador, las líneas de código escritas en un programa, el costo del producto de software, etc. Para llevar a cabo el proceso de medición de un proyecto de software se utilizan métricas de software, las cuales han sido desarrolladas para proveer información acerca de las características que se desean conocer de un producto de software.

2.11. Métricas de software

En la Ingeniería de software una métrica es una medida o conjunto de medidas realizadas al software con el fin de conocer una característica acerca de un producto de software. Una métrica de software es un término que abarca muchas actividades

(Norman Fenton J. B., 2015), todas implican algún grado de medición de software, presentadas a continuación.

- Colección de datos
- Modelos y medidas de estimación de costos y esfuerzo
- Medidas y modelos de calidad
- Modelos de confiabilidad
- Métricas de seguridad
- Métricas de complejidad y estructura
- Evaluación de madurez de la capacidad
- Desarrollo por métricas
- Evaluación de métodos y herramientas

Cada una de estas actividades conlleva un análisis detallado, por medio del cual podemos describir o conocer características de un sistema de software.

2.12. Medidas y modelos

Existen diferentes tipos de modelos, estos pueden ser de estimación de costos, modelos de calidad, modelos de confiabilidad, etc. En general un modelo es una abstracción de la realidad, permitiéndonos conocer a detalle una entidad y analizar entidades desde una particular perspectiva. Por ejemplo, un modelo de costos permite examinar solamente los aspectos que contribuyen a determinar al costo final del proyecto.

Para este trabajo en particular, nos enfocaremos en la medición del diseño de un sistema de software orientado a objetos, utilizando métricas de software.

2.13. Atributos de calidad en el diseño de una aplicación orientada a objetos

En la fase de diseño se realizan diversas actividades por medio de las cuales se obtiene el modelo de la aplicación a desarrollar, en general, las aplicaciones orientadas a objetos utilizan el estándar UML para establecer el diseño del sistema, el cual puede ser analizado por los a través de atributos de diseño de software (Gomaa, 2013):

- *Acoplamiento* (Norman Fenton J. B., 2015): El acoplamiento es un atributo de un módulo individual y depende de los enlaces desde y hacia los elementos que son externos al módulo.

- *Cohesión* (Norman Fenton J. B., 2015): Es un atributo de un módulo individual y depende del grado en que los elementos internos están relacionados dentro de sí mismo.
- *Complejidad* (Norman Fenton J. B., 2015): La complejidad de un sistema depende del número de enlaces entre los elementos de un módulo.
- *Herencia* (Weisfeld, 2013): La herencia se basa en una relación jerárquica entre varias clases de tal forma que se pueden compartir atributos y operaciones en la subclase de la superclase de la que hereda ciertas propiedades y añade propiedades particulares.
- *Tamaño* (Norman Fenton J. B., 2015): Es un atributo el cual únicamente está basado en la longitud dimensional del sistema.

2.14. Métricas de software orientadas al diseño de una aplicación

Para poder obtener información acerca del diseño de una aplicación de software orientada a objetos se han desarrollado métricas de software, las cuales tienen el objetivo de describir de manera cuantitativa el diseño de una aplicación (Norman Fenton J. B., 2015). Entre las más destacadas para realizar dicho proceso se encuentran las siguientes:

- El conjunto de métricas propuestas por (Shyam R. Chidamber, 1994):
  - CBO (Coupling Between Objects): es el número de clases a las cuales una clase está ligada, sin tener con ella relaciones de herencia.
  - LCOM (Lack of Cohesion in Methods): número de grupos de métodos locales que no acceden a atributos comunes.
  - RFC (Response For a Class): es el cardinal del conjunto de todos los métodos que se pueden invocar como respuesta a un mensaje a un objeto de la clase o como respuesta a algún método en la clase.
  - WMC (Weighted Methods per Class): se define como el sumatorio de las complejidades de cada método de una clase.
  - DIT (Depth of Inheritance Tree): mide el máximo nivel en la jerarquía de herencia.
  - NOC (Number of Children): es el número de subclases subordinadas a una clase en la jerarquía, es decir, la cantidad de subclases que pertenecen a una clase.
- La métrica de complejidad propuesta por (McCabe, 1976).
  - CC (Ciclomatic Complexity): La complejidad ciclomática de un método es el número de caminos independientes a lo largo del método.
- El conjunto de métricas propuestas por (Abreu, 1996):
  - COF (Coupling Factor): es la proporción entre el número real de acoplamientos no imputables a herencia y el máximo número posible

de acoplamientos en el sistema. Es decir, indica la comunicación entre clases.

- MHF (Method Hiding Factor): es la proporción entre los métodos definidos como protegidos o privados y el número total de métodos.
- AHF (Attribute Hiding Factor): es la proporción entre los atributos definidos como protegidos o privados y el número total de atributos.
- MIF (Method Inheritance Factor): es la proporción entre la suma de todos los métodos heredados en todas las clases y el número total de métodos (localmente definidos más los heredados) en todas las clases.
- AIF (Attribute Inheritance Factor): es la proporción entre el número de atributos heredados y el número total de atributos.
- POF (Polymorphism Factor): es el número de métodos heredados redefinidos dividido entre el máximo número de situaciones polimorfas distintas posibles.

También existen las métricas propuestas por (Lorenz Mark, 1994), entre otras.

2.15. Teoría de grafos

La estructura matemática conocida como un grafo, es una herramienta valiosa que nos ayuda a visualizar, analizar y generalizar la situación de un problema y poder entender mejor las causas y generar una posible solución.

La formal definición formal de un grafo es de la siguiente manera de acuerdo con Benjamin (Arthur Benjamin, 2015). Un grafo ‘G’ es un conjunto finito de objetos llamados vértices denominado ‘V’, junto con un conjunto de elementos denotado como ‘E’ que consta de 2 subconjuntos de ‘V’. Cada elemento de ‘E’ es llamado arco de ‘G’. Los conjuntos ‘V’ y ‘E’ son llamados conjunto de vértices y conjunto de arcos, que se denotan de la siguiente manera G = (V, E). El número de vértices en un grafo ‘G’ es llamado grafo de orden ‘n’ y el número de arcos en ‘G’ determina el tamaño.

Para analizar los caminos representados en un grafo, se han desarrollado varios algoritmos de recorrido los cuales se utilizan de acuerdo con la forma del grafo y a la información que se requiera obtener. Entre los más populares se encuentran los siguientes:

- Algoritmo de Dijkstra
- Algoritmo de Bellman Ford
- Algoritmo de Floyd Warshall
- Algoritmo Kruskal
- Algoritmo de Prim

- Algoritmo Ford Fulkerson
- Entre otros

En la Ingeniería de software, los grafos son utilizados para modelar la estructura de los módulos del sistema, representar rutas de operación dentro de la estructura de una aplicación, etc. En el siguiente capítulo se menciona el estudio del estado de arte en relación con la problemática a resolver en este trabajo.

3. Trabajo relacionado

En este capítulo se analizará la investigación desarrollada por la comunidad científica en relación con el trabajo presentado en este documento.

En el manual de (Robert E. Park, 1996), del instituto de Ingeniería de software de la universidad de Carnegie Mellon establece la distribución y detección de fallos al largo del proceso de desarrollo de software representado en la Figura 3.1. Puede observarse que en las fases de diseño e implementación se concentran la mayor cantidad de fallas de origen en el sistema y en las fases posteriores se detectan la mayor cantidad de fallos en él. Un punto importante por mencionar es que entre más temprana sea la detección de los fallos es menor el costo para corregirlo, ya que este aumenta dada la maduración del sistema de software a desarrollar.

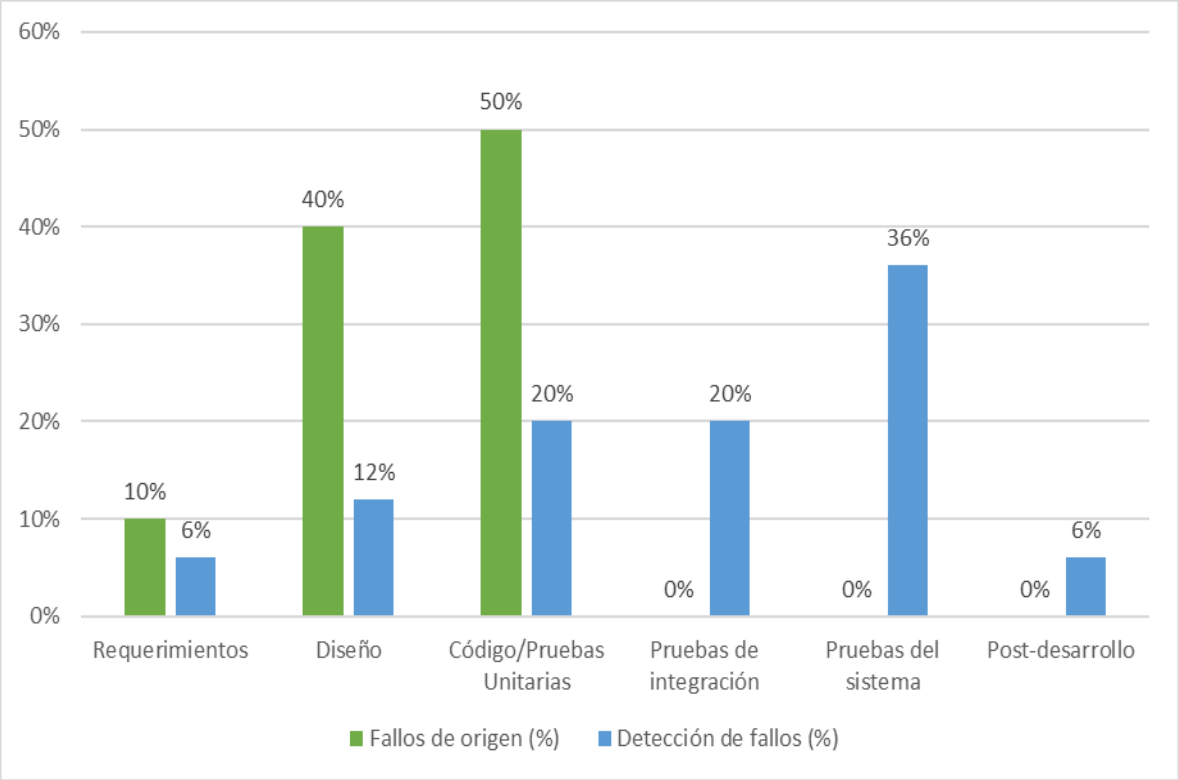

Figura 3.1 Gráfica de uso de recursos en el proceso de desarrollo de software

El IEEE Std 1061-1998 (IEEE, 2009), establece una metodología para establecer la calidad de un sistema de software en cada una de las fases del proceso de desarrollo, en la cual se dan definiciones sobre lo que es calidad, métricas, factores de calidad, etc. Se establece el proceso que debe de seguir para documentar las tareas y procesos de medición a realizar.

3.1. Justificación y categorización de las métricas de software orientadas a objetos

En los trabajos de (Abreu, 1996), (Lorenz Mark, 1994), (McCabe, 1976), (Shyam R. Chidamber, 1994), se proponen un número finito de métricas para obtener información acerca de los atributos del diseño de un software, definiendo cada una de ellas y estableciendo la forma en la cual se debe obtener su valor. Por otro lado, los trabajos de (Béla Újházi, 2010), (Da-wei, 2007), (Ladan Tahvildari, 2000), (Sukainah Husein, 2009), (Lalji Prasad, 2009), analizan y categorizan las métricas de software existentes, para establecer la utilidad específica de cada una de ellas, es decir, establecen a que atributo del diseño de un software debe ser enfocada cada una de las métricas para obtener de la misma información relevante a las mediciones realizadas.

En el trabajo de (Mukesh Bansal, 2014), se aborda el tema de métricas de software orientadas a objetos, tomando como punto inicial las suites de métricas propuestas, la CK suite (Chidamber, 1994), la MOOD suite (Melo, 1996), Lorentz and Kidd suite (Lorenz, 1994), etc. Dicho trabajo se enfoca en analizar cada una de las métricas de software propuestas, analizándolas con las propiedades que deben de cumplir de acuerdo con el estudio de Weyuker (Weyuker, 1998). Se analizaron las suites antes mencionadas y llegan a la conclusión de que la CK suite y la MOOD suite son adecuadas a utilizar para determinar la calidad de un sistema de software orientado a objetos, cada una de ellas con sus limitaciones y excepciones y descartando la Lorenz and Kidd suite dado que el autor menciona que dichas métricas tienen una mala reputación.

De acuerdo con (XIA, 1996), se analiza el acoplamiento como un módulo, el cual está presente en un sistema de software, es analizado desde diferentes puntos de vista y como esta afecta directamente la complejidad del sistema. El trabajo se enfoca en establecer que el acoplamiento es una métrica de diseño, la cual puede ser obtenida por diferentes procedimientos, determina que factores se deben considerar para obtener los valores de esta y la forma de realizar el procedimiento. El trabajo concluye que el acoplamiento es una métrica de diseño del software, la cual se relaciona directamente con la complejidad del sistema y es de mucha utilidad dada la información que proporciona sobre el sistema de software.

3.2. Uso y aplicación de métricas en sistemas de software orientados a objetos

El lenguaje de modelado unificado [UML] (Group, 2015), es un estándar utilizado para el análisis y diseño de los sistemas de software orientado a objetos. Dicho estándar ha servido como una herramienta para el desarrollo de trabajos

como el de Yadav (Vibhash Yadav, 2013), para determinar la calidad en el diseño de un sistema de software orientado a objetos o como el de Halim (Halim, 2013), para predecir la propagación de los fallos en el software.

De acuerdo con Yadav (Vibhash Yadav, 2013), se realiza un análisis de cuáles son las métricas más adecuadas a utilizar para establecer la calidad del diseño de una aplicación orientada a objetos utilizando diagramas UML, el cual selecciona un diagrama UML para analizar el problema en cuestión y un número de métricas tengan relación al atributo de diseño del software que se desee analizar, realiza un análisis en el cual establece que métricas son las más adecuadas a utilizar y que atributos de diseño evalúa cada una de ellas. Para demostrar los resultados de su estudio, utilizó diversos softwares de código abierto, de estos se conoce la calidad del diseño de la aplicación y se les aplicó el estudio. En el trabajo se concluye que la complejidad ciclomática, la falta de cohesión en los métodos, los métodos ponderados por clase y las líneas de código por método, son las métricas de software más adecuadas para determinar la calidad del diseño.

En el trabajo de (Halim, 2013), se utiliza la complejidad de los diagramas de clases para predecir la propagación de fallos en las clases del sistema, argumentando que la complejidad del diseño es un factor importante a analizar, para lo cual utilizó una combinación de una métrica de complejidad con dos algoritmos de clasificación los cuales son Naive Bayes y k-Nearest Neighbors, realizando un análisis de acuerdo a la clasificación obtenida por los algoritmos estableciendo falsos positivos, falsos negativos, verdaderos positivos, verdaderos negativos, como valores de salida para la propagación de los fallos en cada una de las clases del sistema, concluyendo que es posible predecir la propagación de fallos desde la fase de diseño del sistema, esto de acuerdo con la complejidad de los diagramas de clases.

En el trabajo de Rathore (Santosh Singh Rathore, 2012), se establecen las métricas de software que se deben analizar para predecir la propagación de fallos en módulos de software, en él se realiza un estudio de los atributos de diseño del software entre los cuales se encuentra la complejidad, el tamaño, el acoplamiento y la herencia, se establecen métricas de software que deben utilizar para medir cada atributo. Se construyen 15 diferentes modelos combinando los atributos de diseño, utiliza 4 algoritmos de clasificación, los cuales son: Naive-Bayes, Random Forest, Logistic Regression e IBK. Se concluye que los modelos que analizan los atributos acoplamiento y complejidad obtienen mejores resultados que los que analizan otros atributos, también establece que el modelo que contiene acoplamiento, complejidad y tamaño es una opción funcional, en caso de no utilizar el modelo óptimo que es el de complejidad y acoplamiento.

De acuerdo con Lionel (Lionel C. Briand, 1998), se puede utilizar métricas de diseño para predecir la propagación de fallos en las clases de un sistema de software orientado a objetos, los atributos de diseño analizados son: acoplamiento, cohesión y herencia, los cuales analiza con diferentes tipos de métricas. Los resultados obtenidos de cada una de las métricas son tratados en un modelo de regresión logística multivariable que determina la propagación de fallos en las clases del sistema. Concluye que durante la etapa de pruebas es posible utilizar medidas de diseño para crear un modelo que predice la propagación de fallos dentro de las clases del sistema de software.

En otro trabajo de Yadav (A. Yadav, 2010), se demuestra la relación existente entre los atributos de complejidad y acoplamiento, en dicho trabajo se analiza el acoplamiento desde diferentes puntos de vista y se establece que relación existe con la complejidad. Determina que a mayor nivel de acoplamiento aumenta de igual manera el nivel de complejidad, esto se debe a que los módulos de un software que poseen un alto acoplamiento no realizan por si mismos las tareas para las cuales fueron diseñados y necesitan utilizar otros componentes para cumplir con las operaciones para las cuales fueron realizados.

En otro trabajo de investigación realizado por Iyapparaja (M. Iyapparaja, 2012), se analiza la forma de utilizar métricas de software de diseño enfocadas al acoplamiento y cohesión con el fin de reducir el riesgo al reutilizar componentes en un sistema de software. Se realiza un análisis de los factores que se debe tomar en cuenta para reutilizar componentes de software orientados a objetos escritos en lenguaje Java, la finalidad de esto es disminuir la complejidad de adaptación de los componentes reutilizados en los sistemas de software, conociendo las métricas de acoplamiento y cohesión de los paquetes o módulos a reutilizar.

En el trabajo de Shatnawi (Shatnawi, 2014), se realiza un estudio para analizar cada una de las métricas de la CK suite con el fin de predecir fallos en sistemas de software de código abierto, se analizan diferentes proyectos de software de código libre y se aplica a cada uno de ellos las métricas de la CK suite, posteriormente se analiza el resultado de cada una de ellas para poder ser utilizarla en la predicción de fallos del sistema, excluyendo algunas de las métricas por su poca relevancia para llevar a cabo dicho proceso. Se llega a la conclusión que la CK suite es adecuada para poder realizar predicción de fallos en sistemas de software orientados a objetos, teniendo en cuenta el tipo de atributo que se quiera analizar.

De acuerdo con la investigación de Singh (Pradeep Singh, 2012), se analiza la CK suite para determinar su capacidad de predicción de fallos en sistemas de software orientados a objetos desarrollados en lenguaje de programación Java. Se utilizan sistemas de código abierto como casos de estudio, se determinan los valores de cada una de las métricas de software para posteriormente tratar los

resultados en algoritmos de clasificación, estos son el J48 y Naive Bayes. Se encontraron mejores resultados al implementar el algoritmo Naive Bayes y validan que la suite de métricas CK es adecuada para el desarrollo de modelos de predicción de fallos durante la fase de pruebas del sistema.

En el trabajo de Goyal (Puneet Kumar Goyal, 2014), se realiza un análisis especifico de la suite de métricas QMOOD para determinar la calidad de las aplicaciones en lenguaje de programación Java. En este trabajo se presenta una introducción a las MOOD y QMOOD metrics, analizando estas últimas de manera más extensa y mencionando que estás implican 4 niveles y tres linkeos. Los niveles son atributos de calidad en el diseño en la cual se especifica el atributo de calidad a ser analizado y los linkeos la relación existente entre cada uno de los niveles. El primer linkeo se realiza entre el nivel 3 y 4, el cual relaciona los componentes del diseño con las propiedades del diseño, el segundo linkeo establece la relación entre las métricas de software y la propiedad de diseño a analizar, y el último linkeo entre las propiedades del diseño con los atributos de calidad. En general se obtienen las métricas de software, se normalizan, se relaciona cada una con una propiedad en el diseño y sirven para obtener un valor para cada uno de los atributos de calidad en el diseño de una aplicación en Java. Se concluye que la suite de métricas QMOOD es adecuada para establecer la calidad de un programa escrito en lenguaje Java.

Existe gran cantidad de propuestas que utilizan métricas de software orientadas al diseño, como el trabajo de Atole (Chitra S. Atole, 2006) en el cual se analiza el acoplamiento y la cohesión en paquetes de software para predecir la calidad del diseño de un sistema de software orientado a objetos, otros trabajos como los de Yin (Meng-Lai Yin, 2004) y Honglei (Tu Honglei, 2009), analizan el software desde el punto de vista de la complejidad como un factor importante a analizar en el desarrollo de los sistemas de software orientados a objetos.

Cabe mencionar que los trabajos citados están enfocados en la fase de diseño del sistema, sin embargo, existen también trabajos que se aplican en la fase de pruebas del sistema. Por ejemplo, en el trabajo de Mahaweerawat (Atchara Mahaweerawat, 2004), utiliza las redes neuronales para predecir los fallos en un sistema de software orientado a objetos, utiliza una gran cantidad de datos, los cuales se obtienen al realizar la medición de métricas de software y sirven para la implementación de algoritmos de clasificación. Este tipo de análisis utiliza una alta cantidad de recursos dada la cantidad de datos necesarios para realizar el análisis, sin embargo, la predicción de fallos en el sistema es de alto nivel.

En el trabajo de Leticia (Davila, 2008), se realiza el estudio de la evaluación y análisis de la calidad de los sistemas en Internet, el cual basa su análisis en la complejidad de los módulos que componen el sistema web y con ellos poder

predecir la probabilidad de fallos de cada uno de los componentes del sistema, para llevar a cabo este proceso utilizó la teoría de grafos y los datos de una métrica de complejidad. Utilizó el algoritmo de Dijkstra para obtener pesos en los caminos de operación e implementó una función de riesgo para cada una de las operaciones del sistema, obteniendo en cada una de ellas un número de fallos en ejecución dada una cantidad de pruebas. Para este análisis se utilizó un sistema de generación de evaluadores virtuales y una matriz de prueba, esta sirve para comparar directamente la información proporcionada al sistema y la salida esperada a obtener.

Otro tipo de trabajo se basa no solo en el desarrollo de un modelo en alguna fase del proceso de desarrollo, de acuerdo a Fenton (Norman Fenton M. N., 2007), se puede predecir los defectos en el software durante el ciclo de vida del sistema, el cual se tiene como base la cantidad de líneas de código implementadas en cada una de las fases de desarrollo y utilizando un análisis bayesiano en el cual se obtienen los defectos encontrados, insertados, residuales y reparados. Estos sirven como datos de entrada en un análisis posterior y así poder maximizar la corrección y detección de defectos en el software.

Existe gran cantidad de trabajos, algunos de ellos están enfocados en análisis bayesianos como los trabajos de Okutan (Ahmet Okutan, 2014) y Wagner (Wagner, 2010) para predecir fallos en la operación del sistema, algunos otros se basan en redes de Petri como el de Arboleda (Arboleda Adrian, 2012), el cual generó un software de diagnóstico y predicción de fallos en sistemas de centrales de generación eléctrica basándose en modelos generados por redes de Petri.

Dada la información anterior, se pudo determinar que los elementos básicos de análisis en un sistema de software para desarrollar un modelo de análisis son los atributos de diseño del sistema: complejidad y acoplamiento, los cuales, son representados mediante métricas de software y su tratamiento depende del objetivo y la fase de desarrollo en la que el estudio se esté implementando, por lo cual, se establece que estos atributos son características importantes a analizar en los sistemas en desarrollo.

De acuerdo con el estudio del estado del arte, el tratamiento de las métricas de software orientadas a objetos en la fase de diseño es de gran utilidad para el desarrollo de modelos, establecer la calidad del sistema de software, realizar análisis de propagación de fallos, etc.

La Tabla 3.1 muestra en resumen el trabajo relacionado, las formas o métodos utilizados para el desarrollo de la investigación, las métricas de software y los atributos evaluados, en relación con la arquitectura del software.

Tabla 3.1 Concentrado de atributos y métricas por trabajo realizado

Nombre del trabajo	**Referencia**	**Atributos a evaluados**	**Métricas a evaluadas**
Investigating object-oriented design metrics to predict fault-proneness of software modules	(Santosh Singh Rathore, 2012)	Tamaño, cohesión, acoplamiento, complejidad y herencia.	*CBO*, RFC, *LCOM*, CAM, DIT, NOC, *LOC*, *WMC*, *CC*.
Prediction design quality of object-oriented software using UML diagrams	(Vibhash Yadav, 2013)	Tamaño, herencia, abstracción, acoplamiento, complejidad, cohesión, etc.	*CC*, *LCOM*, *WMC*, LOC.
Predict Fault-Prone Classes using the Complexity of UML Class Diagram	(Halim, 2013)	Complejidad	*CC*, RC, NC
Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults	(Yuming Zhou, 2006)	Acoplamiento, cohesión, complejidad, herencia, tamaño.	WMC, *CBO*, DIT, NOC, RFC, *LCOM*, LOC
Critical Analysis of Object-Oriented Metrics in Software Development	(Mukesh Bansal, 2014)	Acoplamiento, complejidad	CK Metrics, MOOD Metrics y Lorenz Kidd metric.
QMOOD metric sets to assess quality of java program	(Puneet Kumar Goyal, 2014)	Tamaño, cohesión, acoplamiento, herencia, abstracción, complejidad, etc.	QMOOD Metrics
Does Coupling Really Affect Complexity?	(A. Yadav, 2010)	Acoplamiento, complejidad	*CBO, CC*, MPC, DAC

Predicting Fault-Prone Classes with Design Measures in Object-Oriented Systems	(Lionel C. Briand, 1998)	Acoplamiento, cohesión, herencia	*CBO*, RFC, MPC, NIH, DAC
The software complexity model and metrics for object-oriented	(Da-wei, 2007)	Tamaño, herencia, acoplamiento, cohesión	*CC*, *LOC*, DIT, RFC
Coupling and Cohesion Metrics in Java for Adaptive Reusability Risk Reduction	(M. Iyapparaja, 2012)	Acoplamiento, cohesión	EV, IV

Se observa en la Tabla 3.1 que las métricas CBO y CC son las métricas con mayor nivel aplicación, seguidas de LCOM y LOC en los trabajos relacionados dado su frecuencia de uso. Así en los atributos de diseño sobresalen: la complejidad, el acoplamiento, la cohesión y la herencia.

3.3. Evaluación de las herramientas de software para sistematizar cálculos de métricas

En los trabajos de Winter (Winter Victor, 2013), Irrazába (Irrazába Emmanuel, 2010) y Ragab (Sahar R. Ragab, 2010) se analizan herramientas de software que permiten el cálculo automático de diversas métricas, las cuales pueden ser obtenidas a través del código fuente de la aplicación, proporcionando para cada una de ellas la información básica de la aplicación como puede ser la cantidad de métricas de software obtenidas, el lenguaje de programación que soporta, etc.

3.4. Complejidad de los algoritmos de recorrido en grafos

Para poder establecer la complejidad algorítmica de un algoritmo de recorrido en grafos, se tiene que considerar los recursos que utiliza a la hora de ejecutar sus procedimientos, estos son: memoria y tiempo. La memoria hace referencia a la cantidad máxima de memoria que utiliza el programa al mismo tiempo y el tiempo a cuanto tarda en correr un algoritmo. En grafos, existen diversos algoritmos de recorrido, los cuales presentan diferentes complejidades en su aplicación, las cuales se muestran en la siguiente lista:

- Algoritmo de Dijkstra: Posee complejidad $O(n^2)$.
- Algoritmo de Bellman-Ford: Posee complejidad O(VE)
- Algoritmo de Kruskal: Posee complejidad O(log(m))
- Algoritmo de Floyd-Warshall: Posee complejidad $O(n^3)$
- Algoritmo de Prim: Posee complejidad $O(n^2)$
- Algoritmo de Ford-Fulkerson: O(QN)

La Figura 3.2, obtenida de (Perez, 2019) muestra la comparación de la complejidad algorítmica.

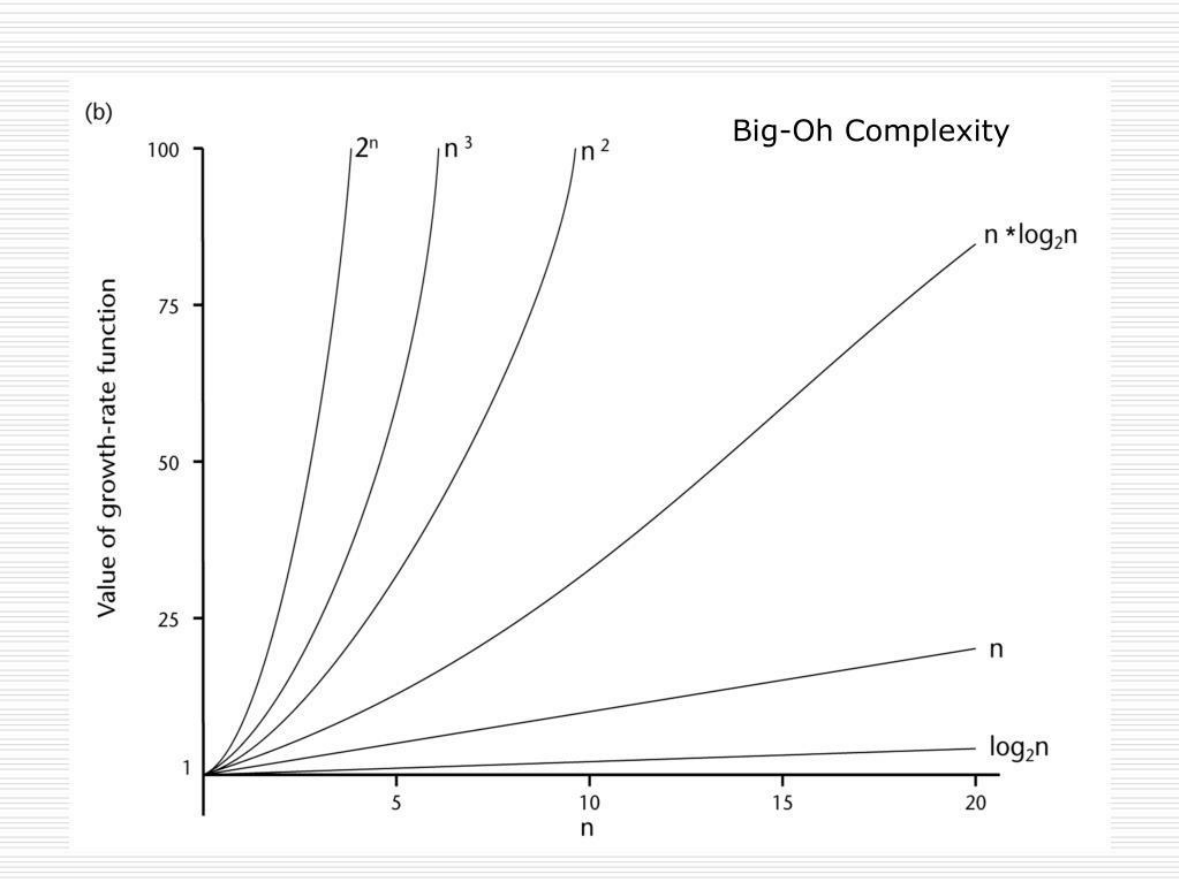

Figura 3.2 Comparación de complejidad algorítmica

A continuación, procedemos a explicar una forma por medio de la cual se puede obtener los diagramas de clases de una aplicación de software en dado caso que está no se encuentre disponible, se utiliza un programa que a través del código fuente de la aplicación puede realizar el modelado de aplicaciones al importar este a la aplicación, obteniendo como resultado diagramas UML.

3.5. Obtener el diagrama de clases a través del código fuente

Tomando en cuenta que en la industria las organizaciones no cuentan con una buena documentación sobre el diseño del sistema, dadas las malas prácticas de los programadores, realizan la codificación sin tomar en cuenta el modelado del sistema, por lo cual, se desconoce la relación que tiene cada módulo del sistema con los componentes de este, sin embargo, es posible utilizar el código fuente para obtener los diagramas del modelado del sistema, se realiza por medio de una herramienta de ingeniería inversa que permite extraer el diagrama de clases del sistema.

A continuación, se enlistan los pasos a seguir para obtener el diagrama de clases utilizando el código fuente de la aplicación, para este caso se utilizó la herramienta Umbrello (KDE, 2014), con el sistema operativo XUBUNTU, cabe mencionar que el uso de esta aplicación no está restringida a este sistema operativo, puede trabajar con ambientes Windows y Linux. Los pasos que se deben seguir son:

1) Al iniciar la aplicación se mostrará la pantalla de inicio.
2) Seleccionar el lenguaje de programación con el cual está desarrollada la aplicación, este proceso se realiza desde el menú código, lenguaje activo y se selecciona el lenguaje correcto.
3) Se debe importar el código a la aplicación, para ello se ingresa al menú código, posteriormente al asistente de importación de código.
4) Seleccionar la ruta donde se encuentre el código fuente, al finalizar darle clic en siguiente.
5) En la pantalla siguiente dar clic en iniciar importación.
6) Al finalizar se mostrará una nueva ventana, dar clic en Finalizar.
7) En la vista de árbol, en el apartado de vista lógica, se podrán visualizar los paquetes del proyecto, así como las clases que contiene cada uno.
8) En la pestaña diagrama de clases, se debe arrastrar cada una de las clases del sistema para que se genere de forma automática su representación y las relaciones que posee con cada una de las clases existentes en el sistema.
9) Se guarda el proyecto en Umbrello desde el menú Archivo, Guardar/Guardar como.
10) Seleccionar la ruta del archivo, el nombre y dar clic en Guardar.
11) Al finalizar el proceso anterior y haber generado el diagrama de clases, se puede importar dicho diagrama a una imagen, para ello vamos al menú Diagrama, posteriormente a la opción Exportar como imagen.
12) Seleccionar el tipo de imagen que se desea obtener, la ruta del archivo, el nombre y se da clic en guardar.

13) Se cierra la aplicación y se termina el proceso de reingeniería.

Las imágenes del proceso antes mencionado se encuentran en el Anexo C de este documento.

Cabe mencionar que este proceso se realiza de igual manera en cada uno de los sistemas operativos en donde se encuentre instalada la aplicación. En el siguiente apartado aplicaremos la metodología para desarrollar el modelo de confiabilidad ya con los requerimientos iniciales para su implementación.

4. Metodología propuesta

La metodología que se propone para el desarrollo de un nuevo modelo de priorización de casos de prueba se describe a continuación en relación de sus fases para su aplicación. Los atributos seleccionados son la complejidad y el acoplamiento, de acuerdo con el estudio del estado del arte, estas propiedades son esenciales para realizar análisis en el diseño de los sistemas de software orientados a objetos y poseen métricas de software confiables en la construcción de modelos de priorización de pruebas. Las métricas de software utilizadas para analizar los atributos de diseño de complejidad y acoplamiento son las siguientes:

- En el análisis del atributo de diseño de complejidad se optó por utilizar la Complejidad Ciclomática Media (CCM).
- Para analizar el atributo de diseño de acoplamiento se utilizó la métrica de Acoplamiento entre Objetos (CBO).

En el análisis para obtener la priorización de casos de prueba a través del diseño, se debe seguir las siguientes fases:

1. Fase 1. Realizar la proyección del Grafo de Complejidad Arquitectónica (GCA)[1] con base en el diseño del sistema evaluado.

   Donde:

   - Los subconjuntos {a, b} representan las relaciones de herencia, composición y agregación del sistema.
   - Los vértices que derivan del primero se relacionan con el diagrama de clases de la arquitectura del sistema.
   - Las aristas → que enlazan los vértices nos indican las relaciones existentes entre las clases del sistema.

   Este procedimiento se realiza utilizando los diagramas de clases del sistema, para lo cual se deben considerar los siguientes puntos:

   - El número de vértices representados en el grafo es X, que representa el total de clases en el sistema.
   - El GCA está compuesto por vértices (v_i) que representan cada una de las clases del sistema y los arcos representan las relaciones existentes en relación con el diagrama de clases.

[1] *Definición 1: Sea un grafo dirigido G definido como un conjunto no vacío de vértices $V= \{V_1, V_2, \ldots, V_n\}$ y un conjunto de aristas $A= \{A_1, A_2, \ldots, A_n\}$. Donde una arista de G es un subconjunto de $\{a, b\} \epsilon V$, con $a \neq b$.*

- Las interfaces no están contempladas en el diseño del GCA. Esto es debido a que su naturaleza como interfaz es abstracta, no está definida en su estructura un procedimiento y/o método, no es posible calcular su complejidad ciclomática de operación.
- La clase que contiene el método main no debe incluirse en el GCA. Esto es debido a que la clase main se crea o construye al momento de la implementación.
- Los arcos del grafo serán ponderados con los datos de la métrica de CCM, la cual establece la complejidad ciclomática media de la clase.

Al término de la creación del GCA se debe implementar un algoritmo de recorrido en grafos que proporcione como salida todos los posibles caminos existentes de recorrido en el grafo, para poder seleccionar los casos de prueba a utilizar para la determinación del nivel de riesgo de las operaciones. El algoritmo implementado para analizar el GCA en este trabajo es el de Flyod-Warshall, el cual obtiene todos los caminos de recorrido en el grafo.

2. Fase 2. Seleccionar los casos de prueba y determinar nivel de riesgo por el atributo de complejidad.
   En esta fase serán contratados los caminos (secuencias) localizadas mediante el análisis de Grafo de Complejidad Ciclomática. La contrastación es en relación de los diagramas de secuencia del sistema de software a evaluar. En esta selección de casos de prueba, se ha tomado en cuenta las siguientes consideraciones:
   - De acuerdo con el resultado obtenido por el algoritmo de recorrido en el GCA, se realizará una selección de información en el cual, se contrastará el camino obtenido en los diagramas de secuencia con los caminos obtenidos con el algoritmo. Al realizar la selección de los casos de prueba será calculado el nivel de riesgo de las operaciones.
   - Los caminos que no estén contemplados en los diagramas de secuencia no son seleccionados como casos de prueba, sin embargo, esta información puede ser utilizada para la realización de otro trabajo de investigación.

   Al realizar la selección de los casos de prueba, se debe determinar los niveles de riesgo en las operaciones del sistema, siguiendo los siguientes criterios:

   - Para determinar el nivel de riesgo de la operación, se debe obtener la complejidad total de la ruta de operación, la cual es ponderada por los pesos de las aristas del grafo involucradas con la operación.
   - Al obtener los datos de las aristas utilizadas para completar la secuencia de operación, estos se deben sumar para obtener la

complejidad total de la ruta de operación y repetir el proceso hasta cubrir todas las rutas de operación en el sistema.

- Aplicar la siguiente función a cada una de las operaciones del sistema:

$$NR(op) = \frac{CCR_i}{\sum_{j=1}^{n} CCR_j}$$

Donde:

NR(op) = Valor del nivel de riesgo de la ruta de operación.

CCR_i = Complejidad Ciclomática de la Ruta analizada.

CCR_j = Complejidad Ciclomática de la Ruta j.

- El nivel de riesgo por complejidad (NRC) se establece de acuerdo con los siguientes criterios, se toma como referencia el resultado de la formula anterior:
  - NRC bajo ϵ ($0 \leq CCR_i < 0.2$)
  - NRC medio ϵ ($0.2 \leq CCR_i < 0.4$)
  - NRC alto ϵ ($0.4 \leq CCR_i < 0.6$)
  - NRC muy alto ϵ ($0.6 \leq CCR_i \leq 1$)

3. Fase 3. Selección de casos de prueba y determinación del nivel de riesgo por el atributo de acoplamiento.

   El nivel de riesgo de la zona[2] (NRA) está dado por el nivel de acoplamiento, el cual indica que entre más alto sea el valor, aumenta la complejidad de su ejecución, así como de prueba y mantenimiento del sistema (A. Yadav, 2010). Para determinar las zonas de riesgo en la arquitectura del sistema será utilizada la métrica CBO, por medio de la cual se realizará un análisis en función del nivel de acoplamiento de la clase en el sistema. Se deben considerar los siguientes aspectos:

   - $NRA_{max} = X$
     - Donde X es el número total de clases del sistema.
   - NRA ϵ ($0 \leq V_i \leq NRA_{max}$)
     - Donde V_i es el resultado de la medición del atributo acoplamiento.

Para determinar el nivel de riesgo en las zonas del sistema se determinará en relación con los siguientes puntos:

R1: Si $V_i \rightarrow 0$ se determina un nivel de riesgo de la zona bajo.

R2: Si $V_i \rightarrow$ NRAmax se determina un nivel de riesgo de la zona alto.

[2] Una zona Z es un conjunto de clases $\{v_i...v_j\}$ relacionadas de forma directa con una clase determinada vi para poder realizar cualquier operación en el sistema.

R3: Si V_i → NRAmax /2 se determina un nivel de riesgo de la zona medio.

4. Fase 4. Análisis de resultados de los datos proporcionados por la metodología y establecer priorización de casos de prueba. Para realizar este proceso es necesario considerar NRC Y NRA. El producto cartesiano de ambos criterios es la base para establecer los criterios de prioridad.
    - De acuerdo con el *NRC* se considera que las operaciones con nivel de riesgo alto son prioritarias en los casos de prueba, seguidas de las operaciones con riesgo medio y al final las de riesgo bajo, este análisis se realiza de con relación al atributo de complejidad.
    - De acuerdo con el *NRA* se considera que la zona más compleja es la de mayor cobertura o alcance, de acuerdo con el resultado del CBO. Los niveles de riesgo por acoplamiento están en función del alcance de las clases con sus derivados.
    - La priorización se establece en relación de NRC y NRA, es el producto cartesiano de ambos conjuntos de casos de prueba. La clasificación de los niveles de prioridad (NP) son:
        - NP 1→ Si NRA´ es alto && NRC´ es muy alto
        - NP 2→ Si NRA´ es alto && NRC´ es alto
        - NP 3→ Si NRA´ es alto && NRC´ medio
        - NP 4→ Si NRA´ es medio && NRC´ es muy alto
        - NP 5→ Si NRA´ es medio && NRC´ es alto
        - NP 6→ Si NRA´ es medio && NRC´ es medio
        - NP 7→ Si NRA´ es bajo && NRC´ es muy alto
        - NP 8→ Si NRA´ es bajo && NRC´ es alto
        - NP 9→ Si NRA´ es bajo && NRC´ es medio
        - NP 10→ Si NRA´ es alto && NRC´ es bajo
        - NP 11→ Si NRA´ es medio && NRC´ es bajo
        - NP 12→ Si NRA´ es bajo && NRC´ es bajo
    - Los niveles NP enlistados son la guía para establecer la priorización de los casos de prueba. Es posible que en los sistemas analizados no siempre estén presentes todos los niveles, por lo tanto, estos niveles se toman de acuerdo con la naturaleza de cada uno de los casos de estudio utilizados.

Las fases anteriores fueron traducidas a un diagrama que representa la metodología a implementar en los casos de estudio, está representa la secuencia que se debe seguir. El diagrama de la metodología propuesta se muestra en la Figura 4.1.

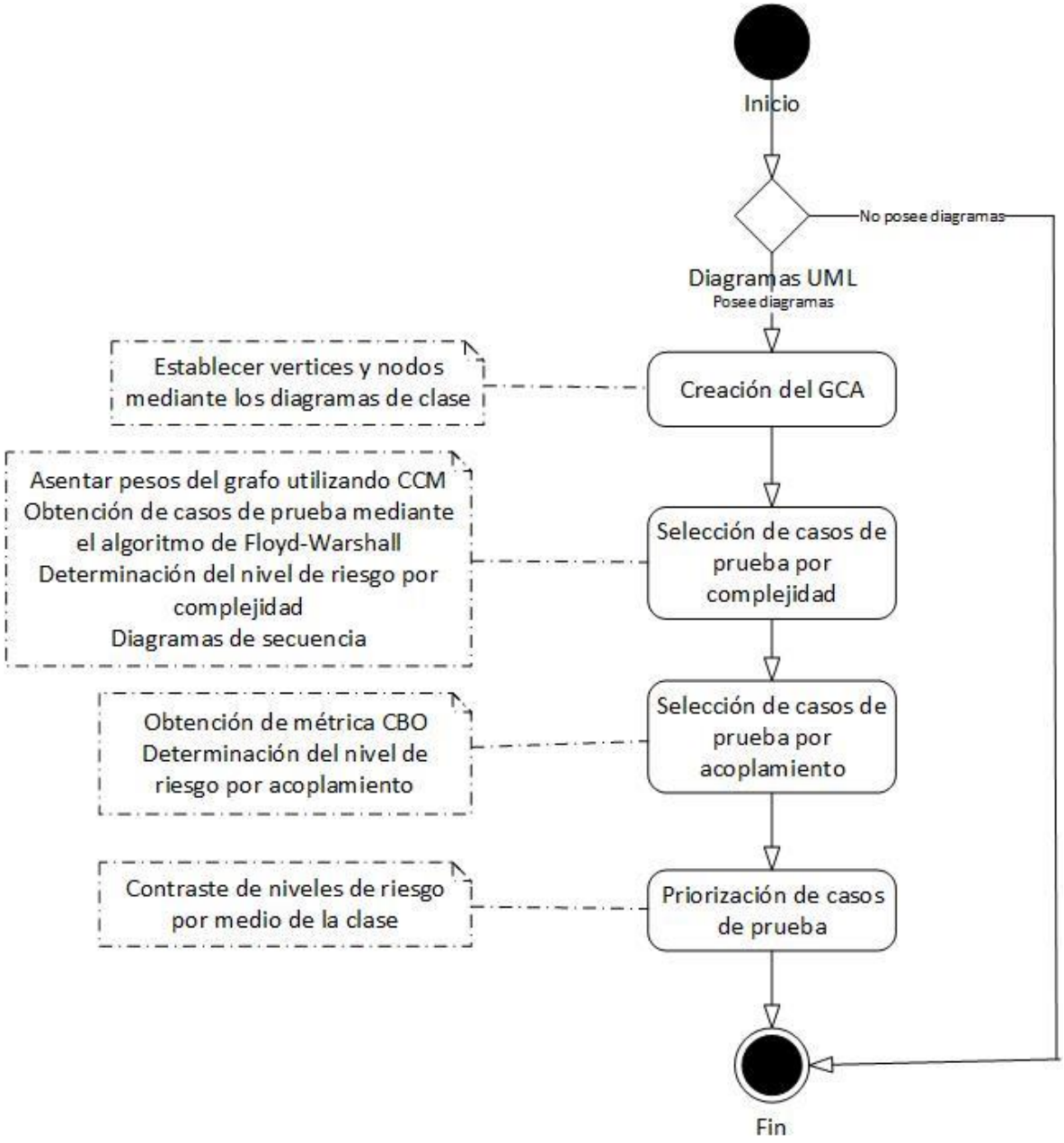

Figura 4.1 Representación de la metodología

Los datos de entrada para la aplicación de la metodología son el diagrama de clases del sistema y los diagramas de secuencias de cada una de las operaciones del sistema.

5. Aplicación de la metodología en el caso de estudio: Sistema Bank

En el contenido de este capítulo, se implementará la metodología propuesta en este trabajo al caso de estudio del sistema Bank.

5.1. Sistema Bank

El sistema Bank, es una aplicación de software orientada a objetos, la cual está destinada a realizar la actividad de un sistema bancario en relación con las actividades más comunes que se realizan, las cuales constan de la creación de cuentas para clientes en el banco, así como la generación de un respaldo de información de los datos de cada uno de los clientes. La Figura 5.1 muestra el diagrama de clases de la arquitectura del sistema.

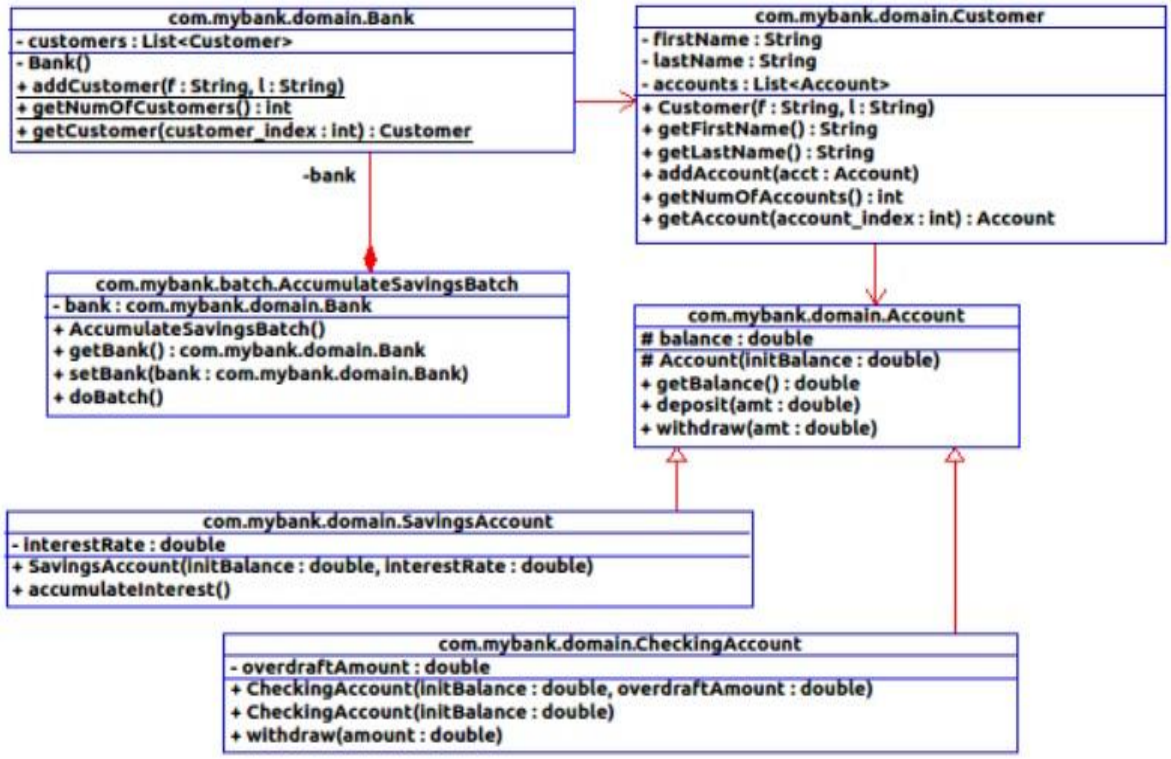

Figura 5.1 Diagrama de clases del sistema Bank

Los diagramas de secuencias que representan cada una de las operaciones del sistema Bank, están representados en las figuras del Anexo A.

5.2. Implementación de la metodología al sistema Bank

Como se menciona en el capítulo 4 la metodología consta de 4 pasos, los atributos seleccionados a utilizar en el análisis del sistema Bank son la complejidad y el acoplamiento para obtener el modelo de priorización de casos de prueba.

- Paso 1. Realizar la proyección del Grafo de Complejidad Arquitectónica (GCA)

Tomando como base la Figura 5.2, se diseñó el GCA que representa el diagrama de clases del sistema BANK. La interpolación de los vértices se hace mediante las clases existentes en el sistema y los arcos están de acuerdo con las relaciones existentes entre cada una de las clases del sistema.

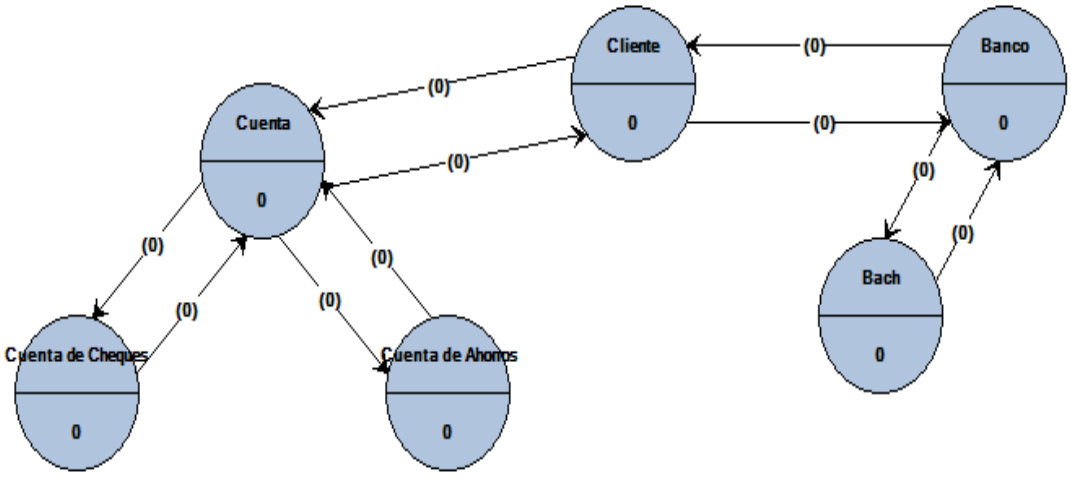

Figura 5.2 Grafo del sistema Bank

- Paso 2: Seleccionar los casos de prueba y determinar nivel de riesgo por el atributo de complejidad

Para analizar la complejidad del sistema se eligió la métrica de software Complejidad Ciclomática Media (CCM) y en el análisis del acoplamiento se optó por utilizar la métrica Acoplamiento entre Objetos (CBO). En el cálculo de las métricas de software, se utilizó un plug-in llamado CodePro Analityx (Google, 2012) desarrollado por Google como complemento para el IDE de programación Eclipse, por medio de la cual se obtuvo la métrica de CCM del sistema y la herramienta Understand (Toolworks, 2014) desarrollada por Scientific Toolworks Coorporation la cual se utilizó para obtener la métrica CBO.

La Tabla 5.1 Métricas de software del sistema Bank muestra los datos obtenidos de la métrica de complejidad ciclomática media (CCM) y la métrica de acoplamiento entre objetos (CBO), de cada una de las clases del sistema Bank.

Tabla 5.1 Métricas de software del sistema Bank

Clase	**Nombre del vértice**	**CCM**	**CBO**
Bank	Banco	1	4
Account	Cuenta	1.25	1
Customer	Cliente	1	4
Savingaccount	Cuenta de Ahorros	1	0
Checkingaccount	Cuenta de Cheques	1.66	1
Batch	Bach	1.75	5

Como es posible observar los resultados en las métricas, la CCM en el sistema Bank posee valores entre [1,2], indicando que la complejidad de operaciones internas en las clases es baja, en relación con la cantidad de métodos definidos en cada una de las clases. Por otro lado, el acoplamiento será representado por valores enteros, los cuales oscilaran entre (0, X), donde X es el número total de clases en el sistema, el resultado de acoplamiento del sistema Bank en general es alto, dado que algunas clases poseen valores muy cercanos a X, lo que determina la autonomía y/o dependencia de cada una de las clases para realizar sus operaciones internas.

Posteriormente, para establecer el peso en los arcos del sistema se utilizó la métrica de CCM, la cual pondera el arco con el resultado de clase de acuerdo con la clase precedente en el sistema. En la Figura 5.3, se puede observar el grafo terminado.

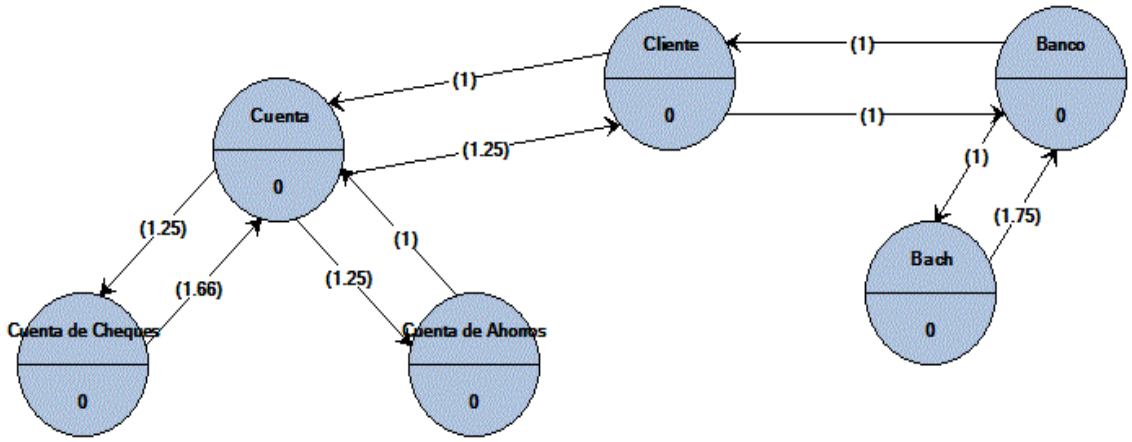

Figura 5.3 Grafo del sistema Bank terminado

Posteriormente se implementó un algoritmo de recorrido en grafos, este proceso se realizó utilizando la herramienta Grafos (Villalobos, 2012), en esta herramienta se puede realizar la implementación de diversos algoritmos de recorrido en grafos, en este caso se utilizó el algoritmo de Floyd-Warshall, el cual reportó los resultados representados en la Tabla 5.2. En esta tabla, la primera columna representa el identificador del caso de prueba o Camino, en la columna 2 se encuentra el vértice inicial, el peso de la secuencia o camino en la columna 3, en la columna 4 se muestra el vértifce final del camino.

Tabla 5.2 Resultados de la implementación del algoritmo de Floyd-Warshall

Camino	Vértice Inicial	Peso	Vértice Final
1.	Banco	0	Banco
2.	Banco	1	Cliente
3.	Banco	2	Cuenta
4.	Banco	3.25	Cuenta de Cheques
5.	Banco	3.25	Cuenta de Ahorros
6.	Banco	1	Bach
7.	Cliente	1	Banco
8.	Cliente	0	Cliente
9.	Cliente	1	Cuenta
10.	Cliente	2.25	Cuenta de Cheques
11.	Cliente	2.25	Cuenta de Ahorros
12.	Cliente	2	Bach
13.	Cuenta	2.25	Banco

14.	Cuenta	1.25	Cliente
15.	Cuenta	0	Cuenta
16.	Cuenta	1.25	Cuenta de Cheques
17.	Cuenta	1.25	Cuenta de Ahorros
18.	Cuenta	3.25	Bach
19.	Cuenta de Cheques	3.91	Banco
20.	Cuenta de Cheques	2.91	Cliente
21.	Cuenta de Cheques	1.66	Cuenta
22.	Cuenta de Cheques	0	Cuenta de Cheques
23.	Cuenta de Cheques	2.91	Cuenta de Ahorros
24.	Cuenta de Cheques	4.91	Bach
25.	Cuenta de Ahorros	3.25	Banco
26.	Cuenta de Ahorros	2.25	Cliente
27.	Cuenta de Ahorros	1	Cuenta
28.	Cuenta de Ahorros	2.25	Cuenta de Cheques
29.	Cuenta de Ahorros	0	Cuenta de Ahorros
30.	Cuenta de Ahorros	4.25	Bach
31.	Bach	1.75	Banco
32.	Bach	2.75	Cliente
33.	Bach	3.75	Cuenta
34.	Bach	5	Cuenta de Cheques
35.	Bach	5	Cuenta de Ahorros
36.	Bach	0	Bach

Al utilizar los diagramas de secuencias se obtiene una lista de caminos en la cual se muestra específicamente las clases que participan para realizar la operación y el camino que debe seguir para concluirla, la Tabla 5.3 representa las operaciones del sistema y el camino a seguir para llevar a cabo dicha tarea.

Tabla 5.3 Operaciones del Sistema Bank

	Nombre de la operación	**Ruta de operación**
1.	Dar de alta a un nuevo cliente.	Banco – Cliente – Banco
2.	Crear una cuenta de cheques.	Banco – Cliente – Cuenta – Cuenta de Cheques – Cuenta – Cliente - Banco
3.	Crear una cuenta de ahorros.	Banco – Cliente – Cuenta – Cuenta de Ahorros – Cuenta – Cliente - Banco
4.	Realizar respaldo de información.	Banco – Bach – Banco
5.	Disposición de efectivo de un cliente.	Cliente – Cuenta – Cuenta de Cheques – Cuenta – Cliente

Para determinar los casos de prueba a analizar específicamente en el grafo son tomados de los diagramas de secuencias, los cuales determinan las rutas de operación real del sistema, dado que las rutas encontradas por el algoritmo de Floyd-Warshall son mayores, estas son seleccionadas y se obtienen los resultados mostrados en la Tabla 5.4.

Tabla 5.4 Selección de casos de prueba en relación con las secuencias de operación

Operación	**Secuencia de operación**	**Caminos del grafo utilizado para realizar el análisis**
Dar de alta a un nuevo cliente.	Banco – Cliente – Banco	Camino 2 Camino 7
Crear una cuenta de cheques.	Banco – Cliente – Cuenta – Cuenta de Cheques – Cuenta – Cliente - Banco	Camino 4 Camino 19

Crear una cuenta de ahorros.	Banco – Cliente – Cuenta – Cuenta de Ahorros – Cuenta – Cliente - Banco	Camino 5 Camino 25
Realizar respaldo de información.	Banco – Bach – Banco	Camino 6 Camino 31
Disposición de efectivo de un cliente.	Cliente – Cuenta – Cuenta de Cheques – Cuenta – Cliente	Camino 10 Camino 20

Al realizar la selección de los casos de prueba, se encontró que 10 de los casos encontrados en el GCA son las relaciones que marcan los diagramas de secuencia en su operación. La selección se utiliza para el análisis y por medio de este establecer el nivel de riesgo de cada una de las operaciones del sistema.

El siguiente procedimiento es el determinar el nivel de riesgo de la operación de acuerdo con la complejidad del camino en cuestión, este análisis se realizará utilizando la función marcada por la metodología propuesta en el capítulo 4. La Tabla 5.5, muestra los resultados de aplicar la función a cada una de las secuencias de operación, para lo cual se obtendrá un valor numérico, que será utilizado para establecer el nivel de riesgo de generarse un fallo en la futura operación del sistema.

Tabla 5.5 Resultados de aplicar la función de riesgo a cada una de las secuencias de operación

Nombre de la Operación	**Casos de prueba utilizados**	**Complejidad total de la ruta**	**Resultado de la función para obtener nivel de riesgo**
Dar de alta a un nuevo cliente.	Camino 2 Camino 7	1+1=2	0.084853627
Crear una cuenta de cheques.	Camino 4 Camino 19	3.25+3.91=7.16	0.303775986
Crear una cuenta de ahorros.	Camino 5 Camino 25	3.25+3.25=6.5	0.275774289
Realizar respaldo de información.	Camino 6 Camino 31	1+1.75=2.75	0.116673738

Disposición de efectivo de un cliente.	Camino 10 Camino 20	2.25+2.91=5.16	0.218922359

Al obtener los resultados de aplicar la función NR(op) se puede establecer el nivel de riesgo por complejidad (NRC) de la ruta de operación, estos se determinan en relación con lo establecido en el paso 6 de la metodología. La Tabla 5.6 muestra los resultados obtenidos.

Tabla 5.6 Niveles de riesgo de las operaciones del sistema Bank

Operación	**Nivel de riesgo**
Dar de alta a un nuevo cliente.	Bajo
Crear una cuenta de cheques.	Medio
Crear una cuenta de ahorros.	Medio
Realizar respaldo de información.	Bajo
Disposición de efectivo de un cliente.	Medio

- Paso 3: Selección de casos de prueba y determinar nivel de riesgo por el atributo de acoplamiento

Al utilizar la información del atributo de diseño de acoplamiento es posible determinar zonas de riesgo en la operación del sistema.

En esta etapa se utiliza la métrica de software enfocada al acoplamiento del sistema para determinar las zonas de riesgo, este análisis se realiza en relación con los resultados de la métrica CBO. La métrica proporciona datos numéricos enteros, los cuales estarán en el rango de (0-X), donde X es el número total de clases en el sistema.

Para determinar zonas de riesgo, es decir, las clases del sistema en las cuales se incrementa la complejidad de prueba en su futura operación, es mediante el valor de la métrica CBO, si el valor de la métrica es 0 indica que para la realización de pruebas unitarias, a la clase se le puede realizar este proceso sin necesidad de incluir o depender del funcionamiento de alguna otra clase del sistema, sin embargo, si el valor es mayor a 0 este indicará el número de clases adicionales con la cual se debe probar la clase en cuestión. Por ejemplo, si el valor del CBO de una clase X es 2, indica que la clase X tiene relación directa con 2 clases en el sistema, por lo que en la fase de pruebas esta debe probarse en conjunto con las 2 clases con las cuales tiene relación o en su defecto emular los resultados obtenidos por las clases

relacionadas, esto es para analizar si la clase X realiza de manera adecuada las tareas u operaciones para las cuales fue desarrollada.

El resultado del valor del CBO oscila entre los valores [0, X], indicando la cantidad de clases con las cuales tiene relación directa, para realizar el análisis de estos resultados se toma como base un área de cobertura en la cual, este se determina por medio de la intersección de conjuntos en relación con las operaciones

En la **¡Error! No se encuentra el origen de la referencia.**, se muestra el r esultado de este análisis y se establece el nivel de riesgo en función del valor de la métrica.

Tabla 5.7 Resultados del análisis del acoplamiento del sistema Bank

Clase	**CBO**	**Nivel de riesgo**
Bank	3	Alto
Account	0	Bajo
Customer	3	Alto
Savingaccount	0	Bajo
Checkingaccount	0	Bajo
Batch	4	Alto

- Paso 4. Análisis de resultados del sistema Bank y establecer priorización de casos de prueba

En la última fase de la metodología, el análisis de los resultados del caso de estudio del Sistema Bank muestra lo siguiente:

- Como se puede observar en la Tabla 5.5, la complejidad ciclomática de las rutas de operación es variable lo que indica diferentes niveles de riesgo. Las operaciones: *crear una cuenta de cheques* y *crear una cuenta de ahorros*, son consideradas en primer orden en la priorización de los casos de prueba del sistema en relación con las otras operaciones del sistema, esto para determinar su buen funcionamiento debido a la complejidad que presenta la operación. Las operaciones: *dar de alta un cliente* y *realizar respaldo de información*, no presentan un alto nivel de riesgo, lo que indica que su priorización como caso de prueba está en último lugar. En cuanto a la operación: *disposición de efectivo*, tiene un nivel de riesgo medio, la priorización de esta operación es en segundo nivel en relación con las demás operaciones del sistema.

- Las zonas de riesgo se presentan en la clase *Bank, Bach y Customer*, esto es por el resultado del análisis de la métrica CBO, a estas clases se les debe priorizar como caso de prueba de primer nivel en comparación con las demás clases, esto es debido a que las clases aumentan su complejidad de prueba dado que su nivel de acoplamiento es alto, proporcionando una cobertura más amplia de operación en relación con los métodos a realizar por cada una de ellas.
- Las clases *Account, Checkingaccount y Savingaccount*, pueden ser analizadas de manera individual, no presentan relación con ninguna otra clase en la función o ejecución de sus métodos.
- El diseño del sistema es adecuado en relación con el número de operaciones que se realizan en él.
- Las zonas de más alto riesgo están presentes en las clases *Bank y Customer*, esto es debido a que están relacionadas con la mayoría de las operaciones del sistema y muestran un nivel de acoplamiento elevado en relación con las clases del sistema.

En relación con la calidad del diseño del sistema podría ser considerada media en relación con lo expresado en el trabajo (Vibhash Yadav, 2013), debido a que la complejidad es baja y el acoplamiento es alto en algunas clases del sistema. Tomando en cuenta solo los atributos de diseño de complejidad y acoplamiento.

6. Aplicación de la metodología en el caso de estudio: Sistema Entorno

En el contenido de este capítulo se analizará otro caso de estudio llamado sistema Entorno, al cual se le implementará la metodología propuesta en este documento.

6.1. Sistema Entorno

El sistema entorno es una aplicación de software que está destinada a realizar tareas en la fase de pruebas de un sistema en desarrollo, por medio de la cual podrá obtenerse información acerca del funcionamiento real del sistema y generar bitácoras con los resultados de las pruebas generadas por dicha fase del proceso de desarrollo. El diseño del sistema Entorno se muestra en la Figura 6.1, la cual representa el diagrama de clases de la aplicación en cuestión.

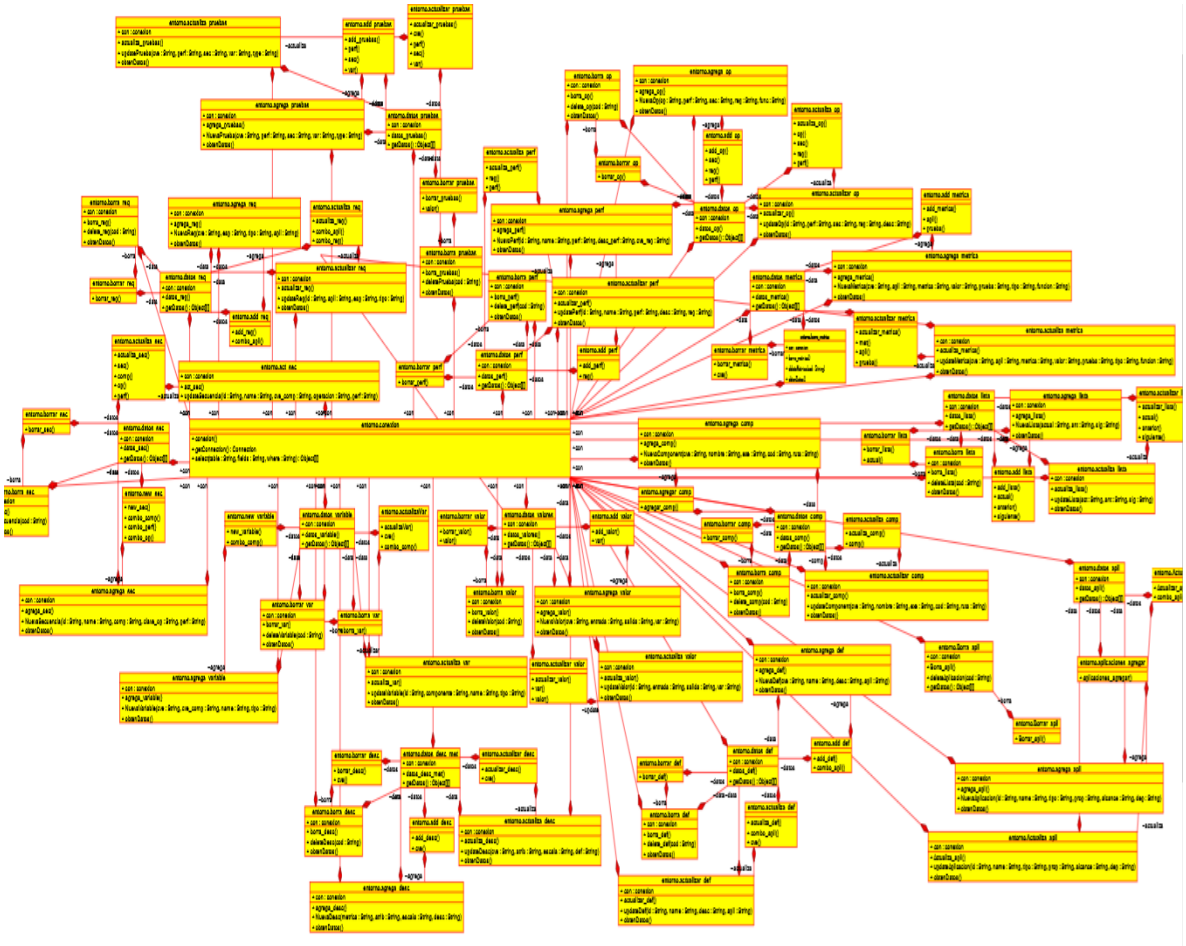

Figura 6.1 Diagrama de clases del sistema Entorno

Los diagramas de secuencias que representan cada una de las operaciones del sistema Entorno, están representados en las figuras del Anexo B”

6.2. Implementación de la metodología al sistema Entorno

Para la realización de esta investigación se tomó un segundo caso de estudio, el cual se le aplicara la metodología propuesta para analizar su priorización de casos de prueba en el sistema.

Los atributos por evaluar en el sistema Entorno para determinar la priorización de los casos de prueba serán las mencionados por la metodología, los cuales son la complejidad y el acoplamiento.

- Paso 1. Realizar la proyección del Grafo de Complejidad Arquitectónica (GCA)

Tomando como base la Figura 6.2, se diseñó el GCA que representa el diagrama de clases del sistema. La interpolación de los vértices se hace mediante las clases existentes en el sistema y los arcos están de acuerdo con las relaciones existentes entre cada una de las clases del sistema.

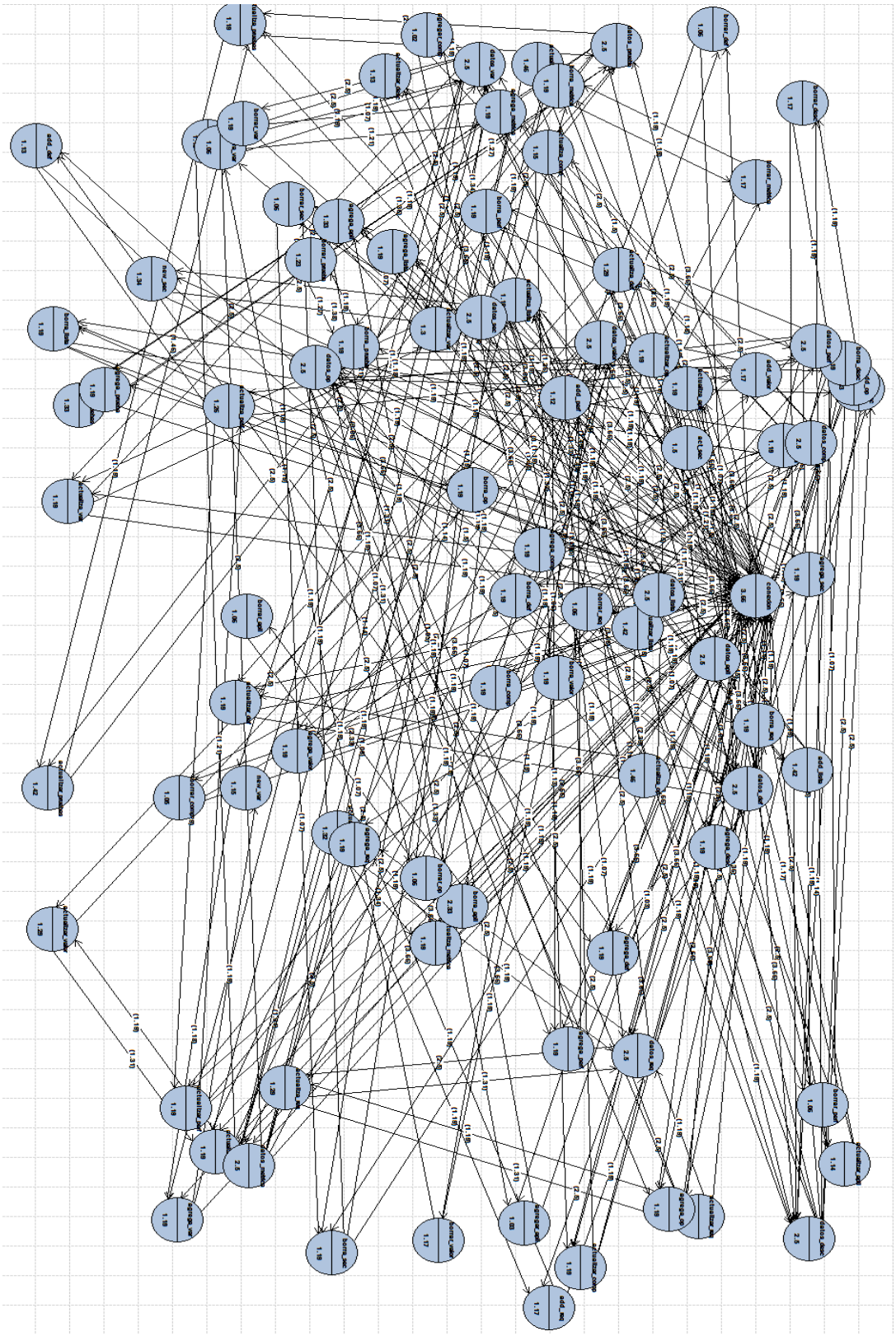

Figura 6.2 Representación del GCA del sistema Entorno

- Paso 2: Seleccionar los casos de prueba y determinar nivel de riesgo por el atributo de complejidad

Las métricas de software utilizadas serán el CBO para el acoplamiento y la CCM para la complejidad, se realizó el mismo procedimiento utilizado en el caso de prueba del sistema Bank, para el cálculo de los valores de las métricas. Los resultados de las métricas de software CCM Y CBO se muestran en la **¡Error! No s e encuentra el origen de la referencia.**.

Tabla 6.1 Resultados de las métricas del sistema Entorno

Clase	CCM	CBO
act_sec	1.5	1
actualiza_apli	1.19	2
actualiza_comp	1.15	3
actualiza_def	1.29	3
actualiza desc	1.19	2
actualiza_lista	1.19	2
actualiza_metrica	1.19	2
actualiza_op	1.46	3
actualiza_perf	1.26	3
actualiza_pruebas	1.19	2
actualiza_req	1.29	3
actualiza_sec	1.46	3
actualiza_valor	1.19	2
actualiza_var	1.19	2
actualizar_apli	1.14	3
actualizar_comp	1.19	2
actualizar_def	1.19	2
actualizar_desc	1.13	3
actualizar_lista	1.42	3
actualizar_metrica	1.32	3
actualizar_op	1.19	2
actualizar_perf	1.19	2
actualizar_pruebas	1.42	3
actualizar_req	1.19	2
actualizar_valor	1.29	3
actualizar_var	1.3	3
add_def	1.13	3
add_desc	1.13	3
add_lista	1.42	3
add_metrica	1.19	3
add_op	1.33	3
add_perf	1.12	3
add_pruebas	1.33	3
add_req	1.17	3
add_valor	1.17	3
agrega_apli	1.33	2
agrega_comp	1.19	2
agrega_def	1.19	2
agrega_desc	1.19	2
agrega_lista	1.19	2
agrega_metrica	1.19	2
agrega_op	1.19	2
agrega_perf	1.19	2
agrega_prueba	1.19	2

agrega_req	1.19	2
agrega_sec	1.19	2
agrega_valor	1.19	2
agrega_var	1.19	2
agregar_comp	1.02	2
agregar_apli	1.03	2
borra_apli	2.33	1
borra_comp	1.19	2
borra_def	1.19	2
borra_desc	1.19	2
borra_lista	1.19	2
borra_metrica	1.19	2
borra_op	1.19	2
borra_perf	1.19	2
borra_prueba	1.19	2
borra_req	1.19	2
borra_sec	1.19	2
borra_valor	1.19	2
borra_var	1.06	2
borrar_apli	1.06	1
borrar_comp	1.05	2
borrar_def	1.06	2
borrar_desc	1.17	3
borrar_lista	1.19	3

borrar_metrica	1.17	3
borrar_op	1.06	2
borrar_perf	1.06	2
borrar_prueba	1.23	3
borrar_req	1.06	2
borrar_sec	1.06	2
borrar_valor	1.17	3
borrar_var	1.19	2
Conexión	3.66	0
datos_apli	2.5	1
datos_comp	2.5	1
datos_def	2.5	1
datos_desc	2.5	1
datos_lista	2.5	1
datos_metrica	2.5	1
datos_op	2.5	1
datos_perf	2.5	1
datos_prueba	2.5	1
datos_req	2.5	1
datos_sec	2.5	1
datos_valor	2.5	1
datos_var	2.5	1
new_sec	1.34	3
new_var	1.15	3

Para establecer los pesos de los arcos del GCA, se han ponderado de acuerdo con el valor de la métrica CCM, el resultado de esta fase está representado en la Figura 6.3, en la que se puede observar que el GCA creado tiene una mayor complejidad de análisis que el mostrado en el sistema Bank.

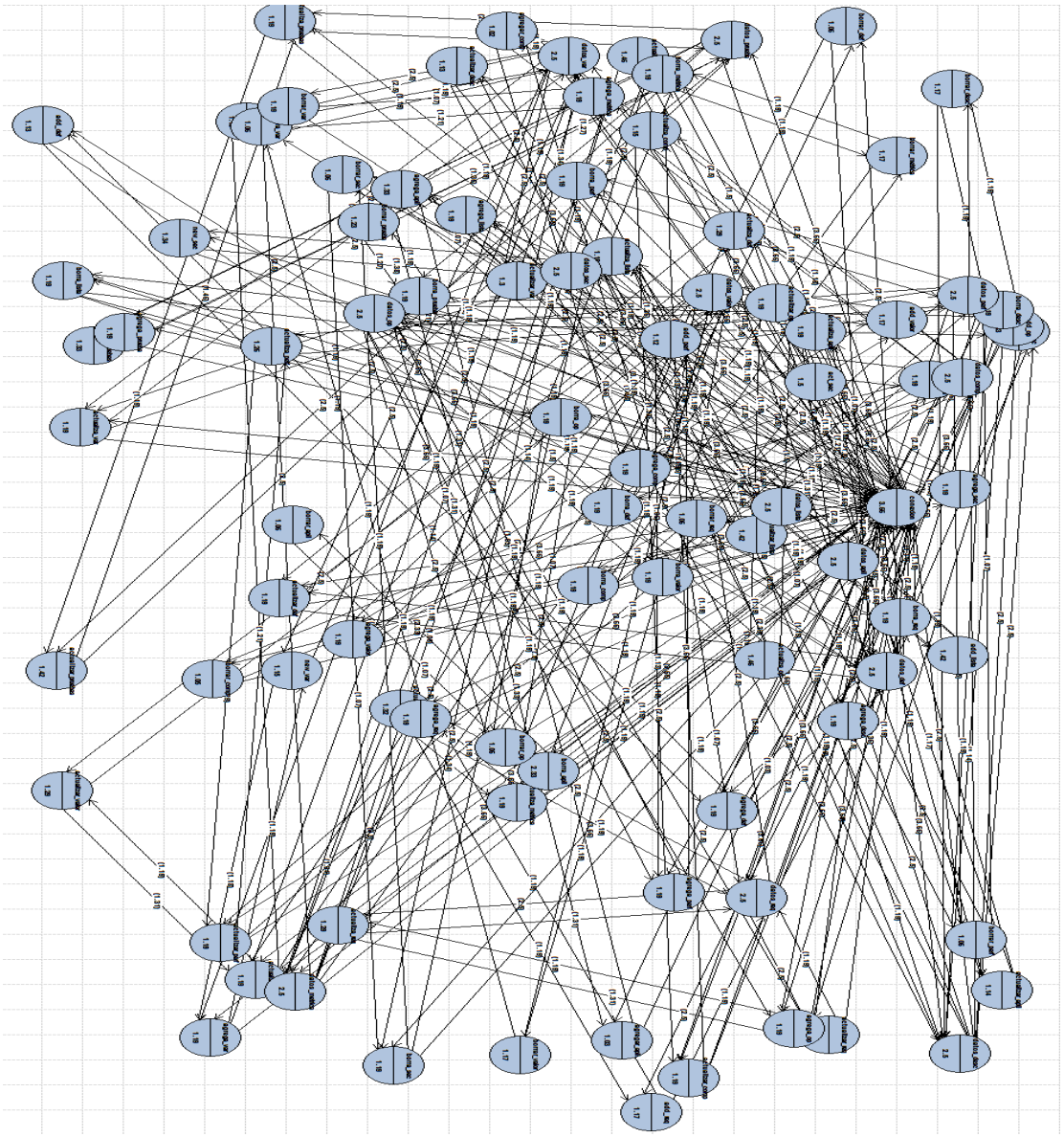

Figura 6.3 Grafo terminado del sistema Entorno

Posteriormente se implementó el algoritmo de Floyd-Warshall en la aplicación Grafos, los resultados se presentan en la Tabla 6.2. Esta muestra en la primera columna el identificador del caso de prueba, en la columna 2 se encuentra el vértice inicial, el peso de la secuencia o camino en la columna número 3 y en la columna número 4 muestra el vértice final del camino.

Tabla 6.2 Resultados de la aplicación del algoritmo de recorrido

Camino	Vértice Inicial	Peso	Vértice Final
1.	act_sec	0	act_sec

2.	act_sec	5.16	actualiza_apli
3.	act_sec	6.35	actualiza_comp
4.	act_sec	6.35	actualiza_def
5.	act_sec	5.16	actualiza
6.	act_sec	5.16	actualiza_lista
7.	act_sec	5.16	actualiza_metrica
8.	act_sec	6.35	actualiza_op
9.	act_sec	6.35	actualiza_perf
10.	act_sec	5.16	actualiza_pruebas
11.	act_sec	6.35	actualiza_req
12.	act_sec	1.5	actualiza_sec
13.	act_sec	5.16	actualiza_valor
14.	act_sec	5.16	actualiza_var
15.	act_sec	6.35	actualizar_apli
...	**…**	**…**	**…**
8450.	new_var	6.03	datos_apli
8451.	new_var	6.03	datos_comp
8452.	new_var	6.03	datos_def
8453.	new_var	6.03	datos_desc
8454.	new_var	6.03	datos_lista
8455.	new_var	6.03	datos_metrica
8456.	new_var	6.03	datos_op
8457.	new_var	6.03	datos_perf
8458.	new_var	6.03	datos_prueba
8459.	new_var	6.03	datos_req
8460.	new_var	6.03	datos_sec
8461.	new_var	6.03	datos_valor
8462.	new_var	1.18	datos_var
8463.	new_var	7.22	new_sec
8464.	new_var	0	new_var

El total de caminos obtenidos por el algoritmo fue de 8464, sin embargo, no son mostrados en su totalidad en la Tabla 6.2 debido al tamaño.

El paso siguiente es realizar la selección de las rutas encontradas por el algoritmo de recorrido en grafos en función de los caminos obtenidos por las operaciones del sistema, los resultados de esta fase se muestran en la Tabla 6.3,

la cual permite visualizar el camino a seguir por la operación en contraste con las rutas obtenidas en el GCA.

Tabla 6.3 Depurado de caminos encontrados en el GCA

Operación	Secuencia de operación	Caminos del grafo utilizado para realizar el análisis
Actualizar aplicación	conexión – datos_apli – conexión – actualiza_apli – actualizar_apli – actualiza_apli – conexión	Camino 7070 Camino 7161 Camino 7007 Camino 1365
Actualizar métricas	conexión – datos_metrica – conexión – actualiza_metrica – actualizar_metrica – actualiza_metrica – conexión	Camino 7075 Camino 7621 Camino 7012 Camino 1825
Actualizar pruebas	conexión – datos_pruebas – conexión – actualiza_pruebas – actualizar_pruebas – actualiza_pruebas – conexión	Camino 7078 Camino 7897 Camino 7015 Camino 2101
Actualizar requerimientos	conexión – datos_req – conexión – actualizar_req – actualiza_req – actualizar_req – conexión	Camino 7079 Camino 7989 Camino 7003 Camino 997
Añadir aplicación	conexión – datos_apli – conexión – agrega_apli – aplicaciones_agregar – agrega_apli – conexión	Camino 7070 Camino 7161 Camino 7042 Camino 4585

Añadir métrica	conexión – datos_metrica – conexión – agrega_metrica – add_metrica – agrega_metrica – conexión	Camino 7075 Camino 7621 Camino 7022 Camino 2745
Añadir pruebas	conexión – datos_pruebas – conexión – agrega_pruebas – add_pruebas – agrega_pruebas – conexión	Camino 7078 Camino 7897 Camino 7025 Camino 3021
Añadir requerimiento	conexión – datos_req – conexión – agrega_req – add_req – agrega_req – conexión	Camino 7079 Camino 7989 Camino 7026 Camino 3113
Borrar métricas	conexión – datos_metrica – conexión – borra_metrica – borrar_metrica – borra_metrica – conexión	Camino 7075 Camino 7621 Camino 7061 Camino 6333
Borrar pruebas	conexión – datos_pruebas – conexión – borra_pruebas – borrar_pruebas – borra_pruebas – conexión	Camino 7078 Camino 7897 Camino 7064 Camino 6609
Borrar aplicación	conexión – datos_apli – conexión – borra_apli – borrar_apli –borra_apli – conexión	Camino 7070 Camino 7161 Camino 7056 Camino 5873

Borrar requerimiento	conexión – datos_req – conexión – borra_req – borrar_req – borra_req – conexión	Camino 7079 Camino 7989 Camino 7065 Camino 6701

De todo el conjunto de secuencias encontradas por el algoritmo, solo se utilizaron 48 casos de prueba con los cuales se da solución a las operaciones del sistema. Esta selección se utiliza para el análisis y por medio de este establecer el nivel de riesgo de cada una de las operaciones del sistema.

El paso siguiente es aplicar la función de riesgo a cada una de las operaciones del sistema y determinar el nivel del riesgo en cada una de ellas, estos resultados se muestran en la Tabla 6.4.

Tabla 6.4 Resultados del análisis de riesgo de las operaciones del sistema Entorno

	Caminos utilizados	Complejidad total de la ruta	Resultado de la función para obtener nivel de riesgo	Nivel de riesgo de la operación
Actualizar aplicación	Camino 7070 Camino 7161 Camino 7007 Camino 1365	13.37	0.081708733	Bajo
Actualizar métricas	Camino 7075 Camino 7621 Camino 7012	13.54	0.082747662	Bajo

	Camino 1825			
Actualizar pruebas	Camino 7078 Camino 7897 Camino 7015 Camino 2101	13.66	0.083481024	Bajo
Actualizar requerimientos	Camino 7079 Camino 7989 Camino 7003 Camino 997	13.51	0.082564322	Bajo
Añadir aplicación	Camino 7070 Camino 7161 Camino 7042 Camino 4585	13.51	0.082564322	Bajo

Añadir métrica	Camino 7075 Camino 7621 Camino 7022 Camino 2745	13.41	0.081953187	Bajo
Añadir pruebas	Camino 7078 Camino 7897 Camino 7025 Camino 3021	13.56	0.082869889	Bajo
Añadir requerimiento	Camino 7079 Camino 7989 Camino 7026 Camino 3113	13.39	0.08183096	Bajo

Borrar métricas	Camino 7075 Camino 7621 Camino 7061 Camino 6333	13.39	0.08183096	Bajo
Borrar pruebas	Camino 7078 Camino 7897 Camino 7064 Camino 6609	13.47	0.082319868	Bajo
Borrar aplicación	Camino 7070 Camino 7161 Camino 7056 Camino 5873	15.55	0.095031473	Bajo

Borrar requerimiento	Camino 7079 Camino 7989 Camino 7065 Camino 6701	13.27	0.081097598	Bajo

- Paso 3: Selección de casos de prueba y determinar nivel de riesgo por el atributo de acoplamiento

A continuación, seguimos con el análisis del acoplamiento del sistema, los resultados se muestran en la Tabla 6.5.

Tabla 6.5 Resultados de la métrica de acoplamiento

Clase	CBO	Nivel de riesgo
act_sec	1	Bajo
actualiza_apli	2	Bajo
actualiza_comp	3	Bajo
actualiza_def	3	Bajo
actualiza desc	2	Bajo
actualiza_lista	2	Bajo
actualiza_metrica	2	Bajo
actualiza_op	3	Bajo
actualiza_perf	3	Bajo
actualiza_pruebas	2	Bajo
actualiza_req	3	Bajo
actualiza_sec	3	Bajo
actualiza_valor	2	Bajo
actualiza_var	2	Bajo
actualizar_apli	3	Bajo

actualizar_comp	2	Bajo
actualizar_def	2	Bajo
actualizar_desc	3	Bajo
actualizar_lista	3	Bajo
actualizar_metrica	3	Bajo
actualizar_op	2	Bajo
actualizar_perf	2	Bajo
actualizar_pruebas	3	Bajo
actualizar_req	2	Bajo
actualizar_valor	3	Bajo
actualizar_var	3	Bajo
add_def	3	Bajo
add_desc	3	Bajo
add_lista	3	Bajo
add_metrica	3	Bajo
add_op	3	Bajo
add_perf	3	Bajo
add_pruebas	3	Bajo
add_req	3	Bajo
add_valor	3	Bajo
agrega_apli	2	Bajo
agrega_comp	2	Bajo
agrega_def	2	Bajo
agrega_desc	2	Bajo
agrega_lista	2	Bajo
agrega_metrica	2	Bajo
agrega_op	2	Bajo
agrega_perf	2	Bajo
agrega_prueba	2	Bajo
agrega_req	2	Bajo

agrega_sec	2	Bajo
agrega_valor	2	Bajo
agrega_var	2	Bajo
agregar_comp	2	Bajo
agregar_apli	2	Bajo
borra_apli	1	Bajo
borra_comp	2	Bajo
borra_def	2	Bajo
borra_desc	2	Bajo
borra_lista	2	Bajo
borra_metrica	2	Bajo
borra_op	2	Bajo
borra_perf	2	Bajo
borra_prueba	2	Bajo
borra_req	2	Bajo
borra_sec	2	Bajo
borra_valor	2	Bajo
borra_var	2	Bajo
borrar_apli	1	Bajo
borrar_comp	2	Bajo
borrar_def	2	Bajo
borrar_desc	3	Bajo
borrar_lista	3	Bajo
borrar_metrica	3	Bajo
borrar_op	2	Bajo
borrar_perf	2	Bajo
borrar_prueba	3	Bajo
borrar_req	2	Bajo
borrar_sec	2	Bajo
borrar_valor	3	Bajo

borrar_var	2	Bajo
conexión	0	Bajo
datos_apli	1	Bajo
datos_comp	1	Bajo
datos_def	1	Bajo
datos_desc	1	Bajo
datos_lista	1	Bajo
datos_metrica	1	Bajo
datos_op	1	Bajo
datos_perf	1	Bajo
datos_prueba	1	Bajo
datos_req	1	Bajo
datos_sec	1	Bajo
datos_valor	1	Bajo
datos_var	1	Bajo
new_sec	3	Bajo
new_var	3	Bajo

- Paso 4. Análisis de resultados del sistema Entorno y establecer priorización de casos de prueba

Los resultados de la aplicación de la metodología del sistema entorno se describen en los siguientes puntos:

- En relación con las rutas de operación del sistema, puede observarse que las operaciones manejan valores muy similares en relación de la complejidad que se presenta en las mismas, lo que determina un nivel de riesgo bajo en todas las operaciones.
- En función del acoplamiento, se observa que los valores de la métrica CBO no exceden el valor 4, esto permite que la fase principal de las pruebas de cada una de las clases sea de igual magnitud en cuanto a la cantidad de pruebas realizadas a cada una de ellas.
- La clase conexión tiene un acoplamiento de 0, lo que permite analizar esta clase de manera unitaria, haciendo mención que es la única en la que puede realizarse de esta manera.

- En este análisis de resultados se encontró que los resultados otorgados por NRC son muy similares, es decir, no existe evidencia visible de la diferencia entre la complejidad de las operaciones. En este caso, el resultado de NRA nos sirvió de guía para establecer la priorización de los casos de prueba, es decir, este resultado apoya o fortalece los resultados encontrados por NRC, esto es debido a que ambos atienden de forma directa la complejidad del sistema (A. Yadav, 2010), por lo tanto, los resultados obtenidos se basan primordialmente en el resultado obtenido por NRA.
- En cuanto a la calidad el diseño, se puede observar que en relación con las métricas de software y lo establecido en el trabajo de (Vibhash Yadav, 2013), la calidad del diseño de esta aplicación es alta, debido a su bajo acoplamiento y su baja complejidad.
- Por otro lado, el diagrama de clases del sistema muestra un patrón en relación con las clases del sistema, se recomienda conjuntarlas utilizando algún patrón de diseño que se adecue al funcionamiento que se le pretende dar al sistema.

Se cree que la información proporcionada por el desarrollador está incompleta.

7. Conclusiones y trabajo futuro

Considerando la relación existente en la priorización de casos de prueba se tiene regresión (Regresión Test Selection) cuando se busca descartar casos de prueba, suites de pruebas (Test Suite Minimization) cuando se minimiza la tasa de fallos detectados y priorización (Test Case Prioritization) cuando se contempla todos los casos de prueba, por consiguiente, al realizar el proceso de priorización en casos de prueba (TCP) el costo de los recursos es alto porque se tiene la condición de contemplar todas las secuencias de operación del sistema que se evalúa. De ahí obtenemos que la técnica seleccionada para el desarrollo del modelo y los algoritmos a implementar tienen que cumplir con esta restricción, teniendo una especial atención en el costo de los procesos. Nuestra propuesta cumple con la condición del TCP y es posible observar que el consumo de recursos está bien definido y este representa costos justificados para su aplicación.

Por otro lado, la priorización de casos de prueba se realiza principalmente en la etapa de pruebas del sistema, sin embargo, nuestra metodología se aplica en etapas tempranas del proceso de desarrollo, en específico en la fase de diseño, donde de acuerdo con el estudio del arte este proceso ha sido menormente explorado.

En esta propuesta se ha evaluado el diseño mediante la arquitectura a nivel de diagramas de clase y secuencia. De una diversidad muy grande de atributos relacionados con el diseño de un sistema de software orientado a objetos, se han seleccionado la complejidad y el acoplamiento como factores importantes para su desarrollo. En la fase de experimentación se presentan resultados confiables en la aplicación de la creación de un modelo de priorización, el uso de las métricas de software relacionadas con los atributos de diseño del software, demostraron que a mayores aspectos evaluados se genera una mayor precisión en los resultados del modelo, sin embargo, entre más datos se inserten a un modelo este será más costoso y complejo al implementarse.

El uso de la técnica de grafos para establecer notaciones formales para el análisis de la arquitectura de software es ampliamente utilizado en el ámbito de priorización de casos de prueba, la interpolación de la arquitectura del sistema representada mediante un grafo, a través de los diagramas de clases, facilita la obtención de zonas, rutas y/o secuencias de operación, es decir, los casos de prueba. Dicha técnica es adecuada porque el costo de los algoritmos que analizan los recorridos es menor que en otras técnicas, lo cual fortalece la sistematización del análisis de datos y ayuda a la interpretación de los resultados.

En la etapa de análisis de los resultados de la metodología se genera un nivel de riesgo en operaciones del sistema y zonas de riesgo de operación en relación con los atributos evaluados: complejidad y acoplamiento. La priorización de los casos de prueba en relación con las operaciones y las zonas del sistema que se evalúa, se determinan por niveles de riesgo que son el resultado de la implementación del modelo y las métricas de software que representan cada uno de los atributos de diseño, estos sirven de guía para establecer una cobertura de pruebas, también proporcionan información para realizar una adecuada distribución de los recursos destinados a la fase de pruebas del proyecto, permitiendo a los desarrolladores y/o líderes del proyecto tomar decisiones en relación al curso que debe tomar el proyecto.

El análisis particular sobre la metodología ha permitido evaluar las ventajas y desventajas que posee, entre las cuales destacan como puntos favorables la aplicabilidad, la simplicidad, la escalabilidad y la validez de los resultados, sin embargo, también se encontraron áreas de oportunidad en las cuales se debe mejorar y refinar, entre las cuales se identifica la inclusión de un atributo de diseño más para fortalecer el modelo, poderlo implementar en sistemas de mayor tamaño, así mismo, realizar un análisis para el desarrollo de una herramienta que permita realizar estos procesos de forma automática y sistematizada.

Como resultado del trabajo de investigación se presentaron 2 artículos para publicación en revistas, el primero abordo la elaboración de la metodología y su implementación en el primer caso de estudio como primera fase, este fue presentado en el CIINDET 2015 (Banda Madrid Abraham D. N., 2015) y el segundo incluyo la implementación del segundo caso de estudio con algunas adecuaciones en la metodología, este fue presentado en el CIM Orizaba 2017 (Banda Madrid Abraham L. D., 2017), de las cuales se recibieron criticas favorables y puntos de vista para puntualizar la actividad que se está desarrollando.

El trabajo a futuro se centra en la inclusión de un atributo de diseño adicional con su métrica de software representativa, para fortalecer el modelo y proporcionar resultados más precisos, así como la presentación de un análisis de correlación entre las métricas utilizadas para proporcionar a la metodología una demostración matemática más profunda en relación con los resultados que proporciona.

8. Anexo A

Diagramas de secuencias de las operaciones del sistema Bank.

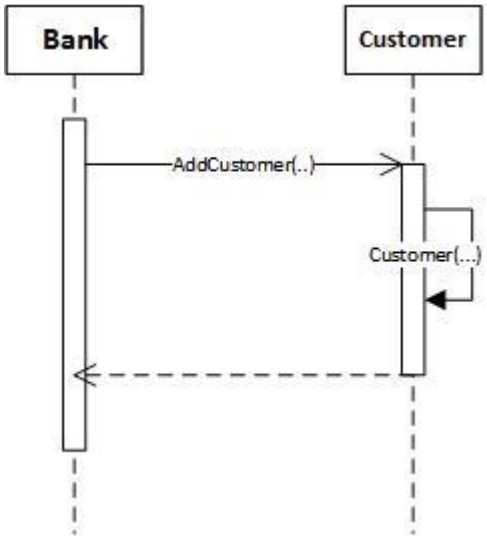

Ilustración 1 Dar de alta un nuevo cliente

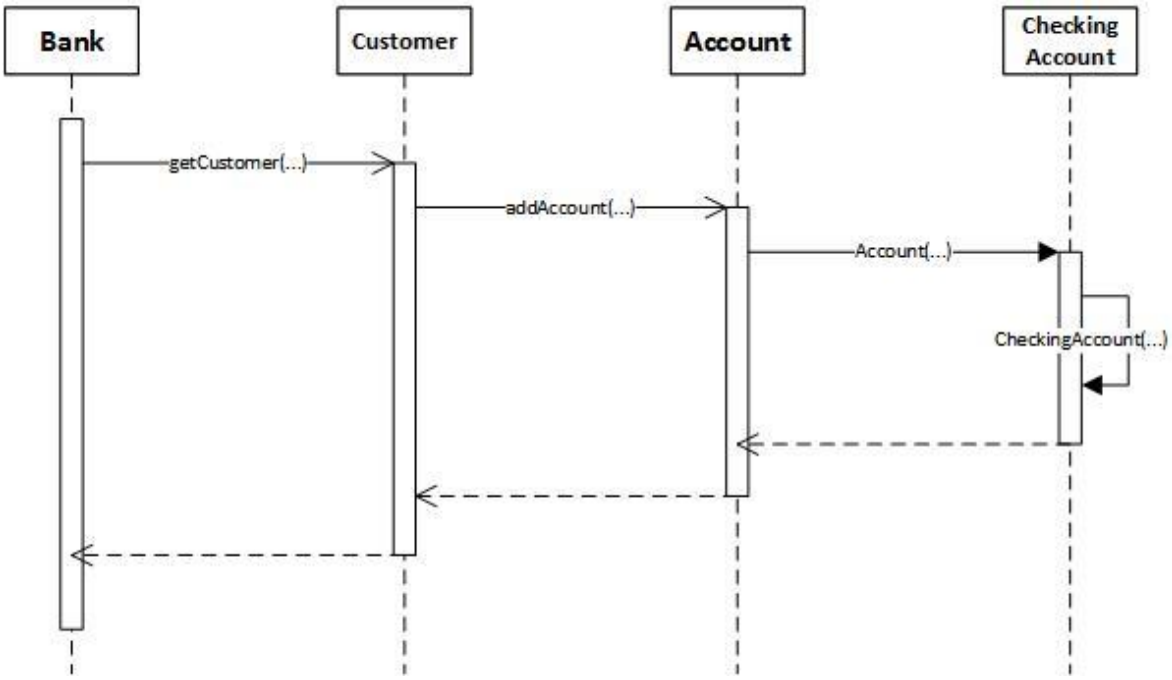

Ilustración 2 Crear una cuenta de cheques.

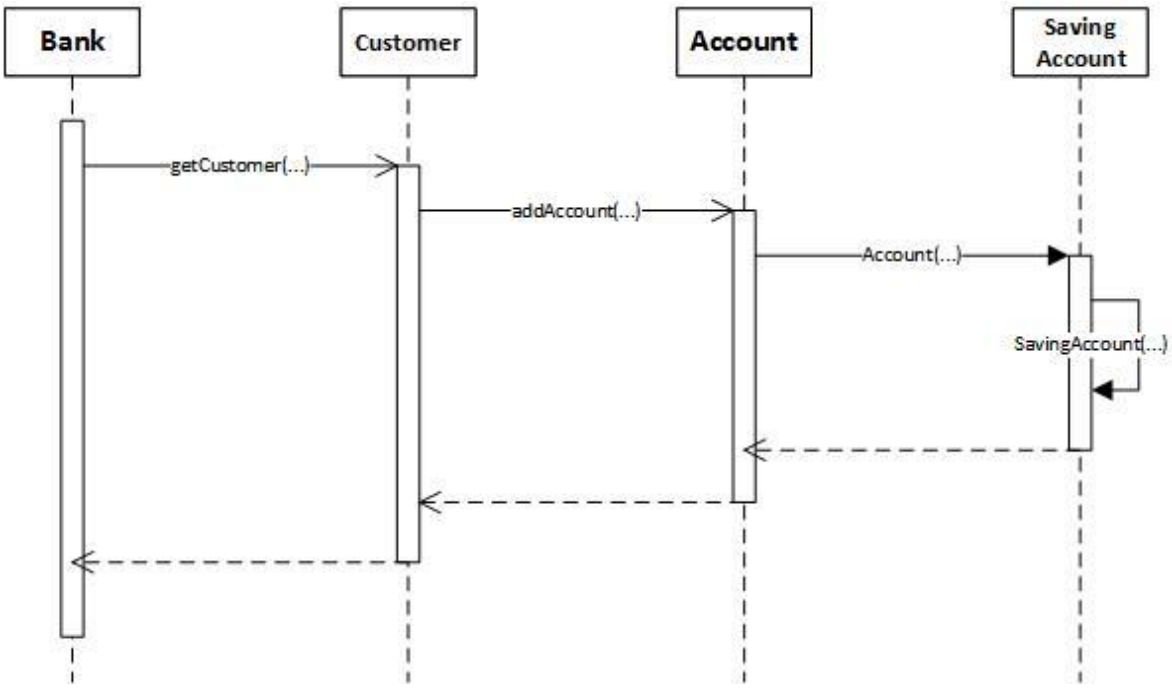

Ilustración 3 Crear cuenta de ahorros.

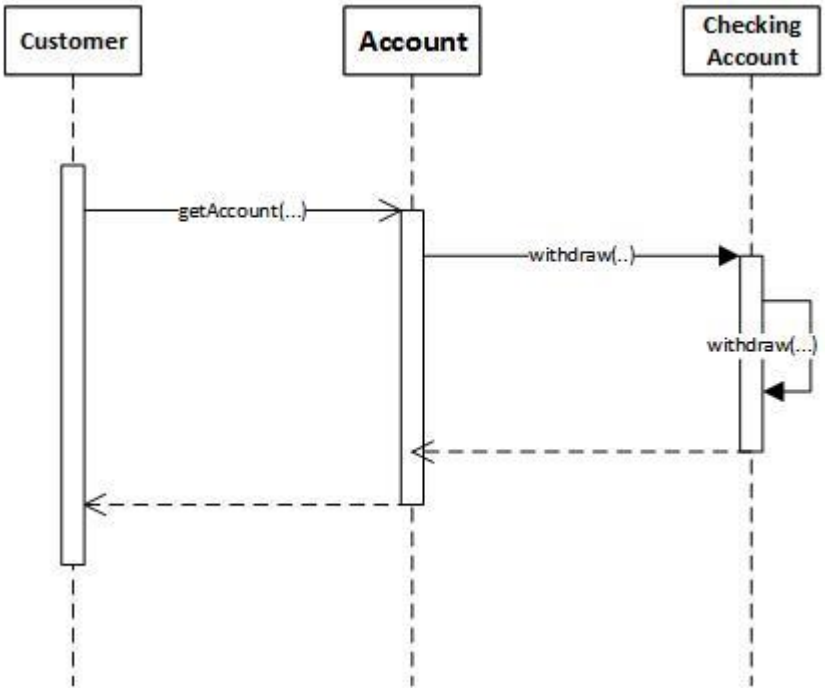

Ilustración 4 Disposición de efectivo de una cuenta de cheques.

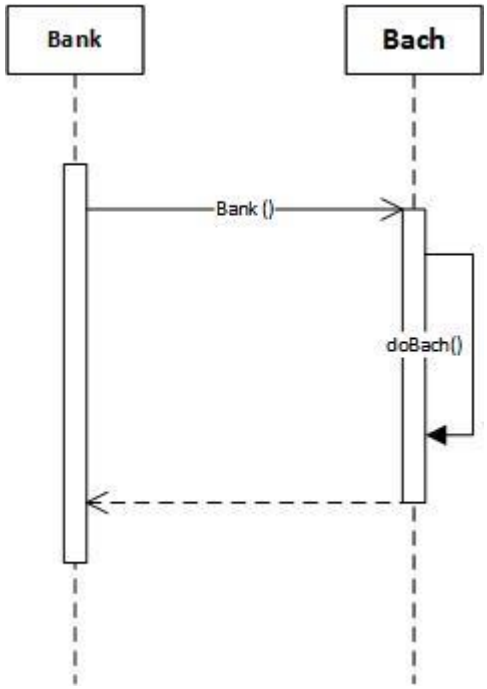

Ilustración 5 Respaldo de información.

9. Anexo B

Diagramas de secuencias de las operaciones del sistema Entorno.

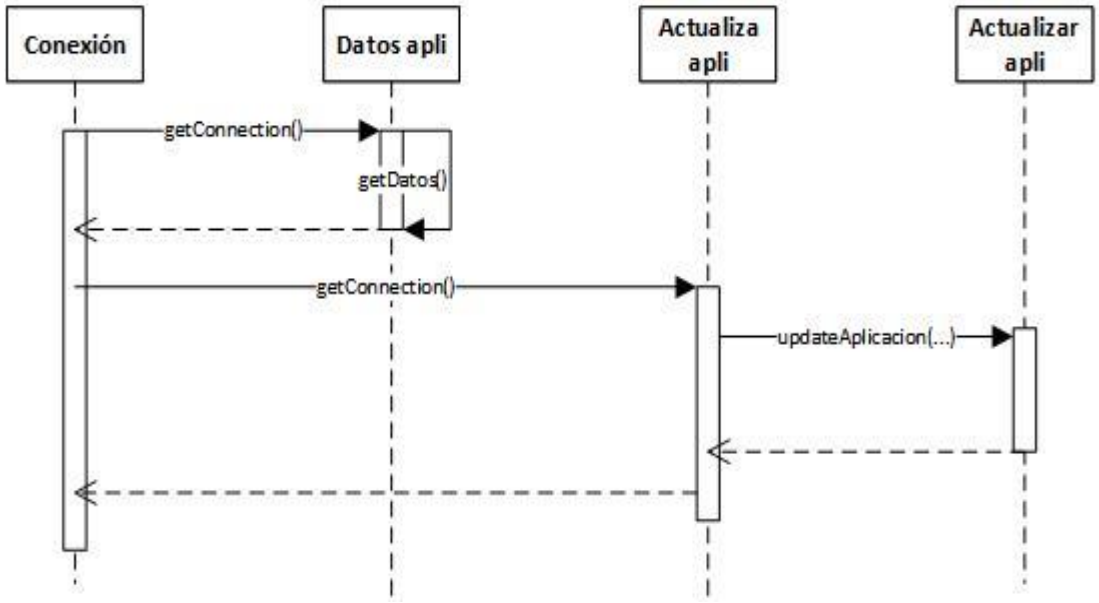

Ilustración 6 Actualizar aplicación.

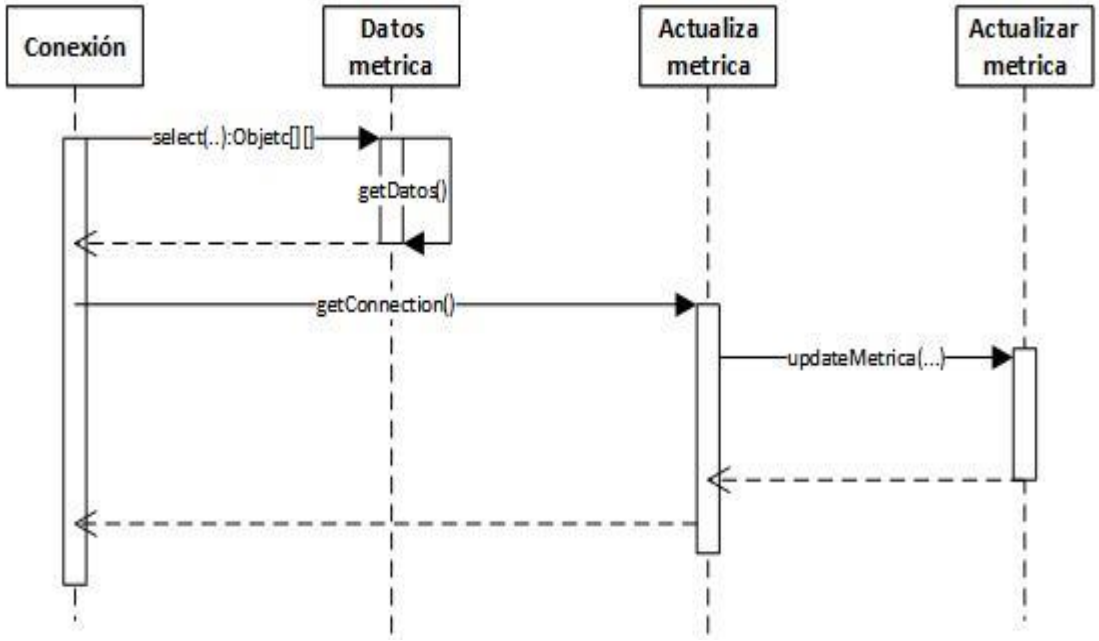

Ilustración 7 Actualizar métrica.

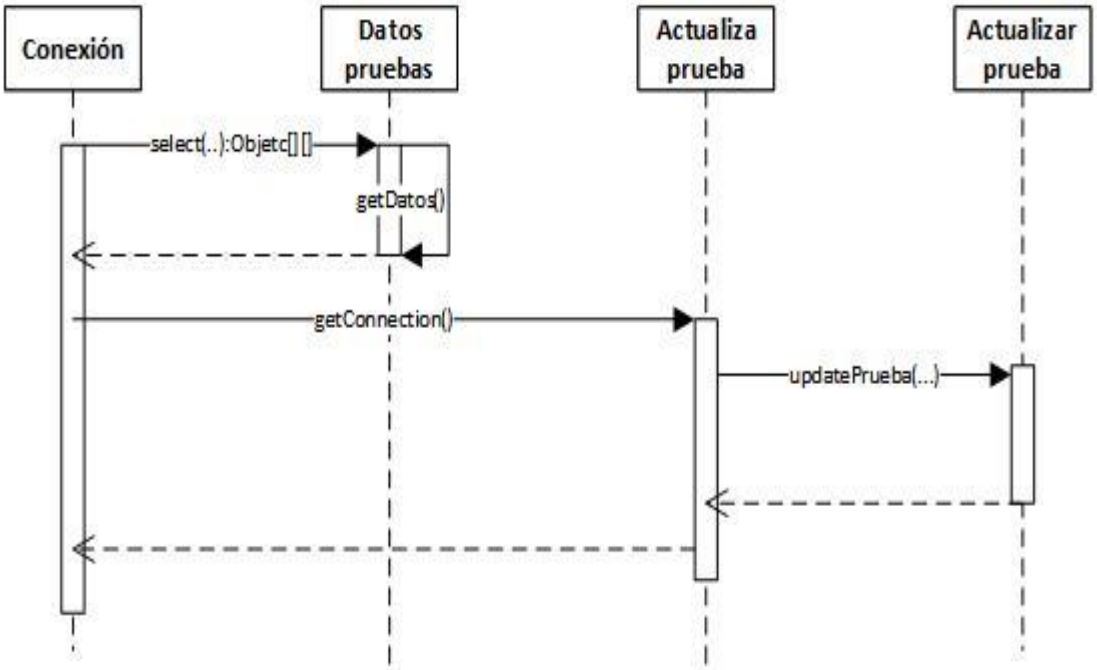

Ilustración 8 Actualizar prueba.

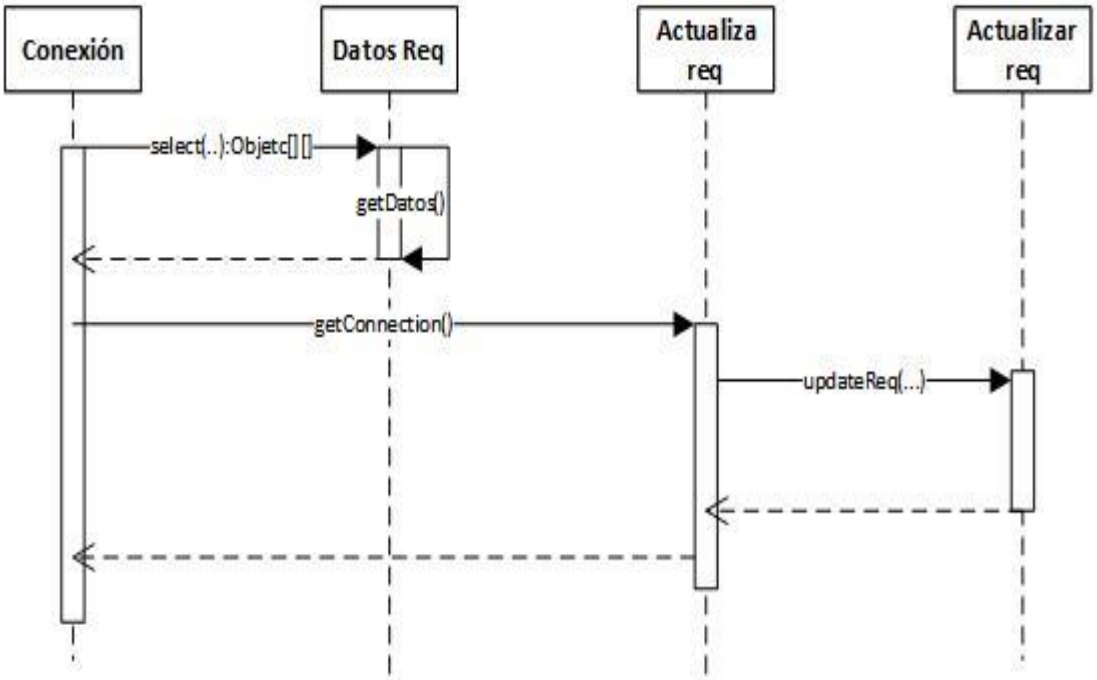

Ilustración 9 Actualizar requerimiento.

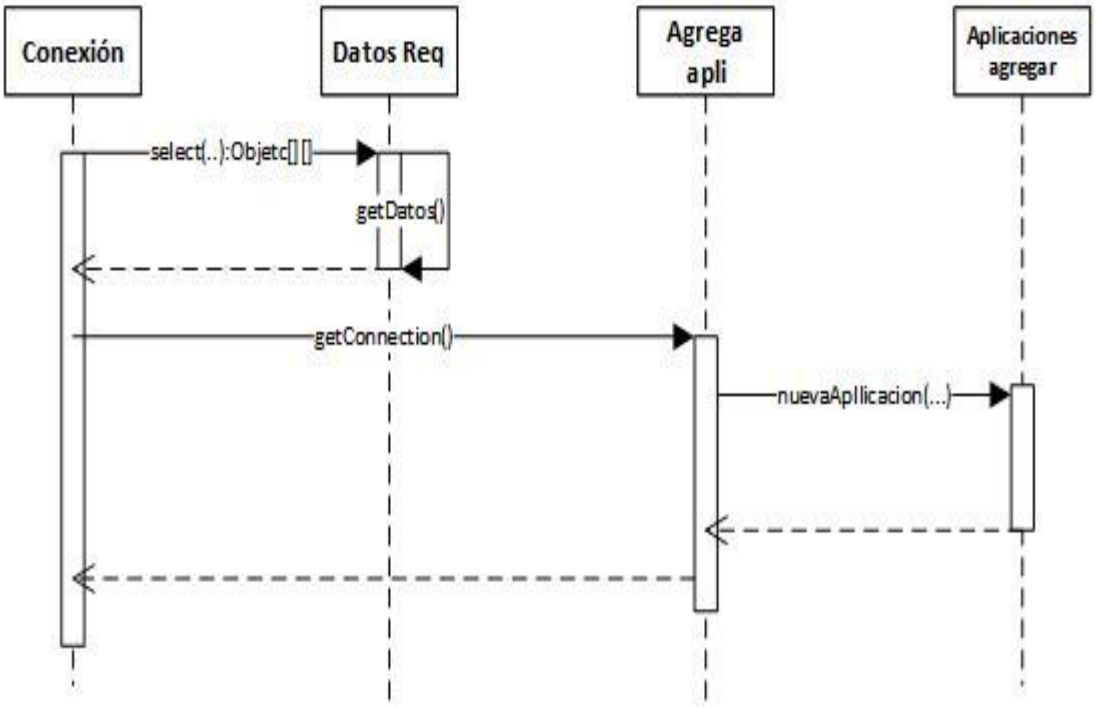

Ilustración 10 Añadir aplicación.

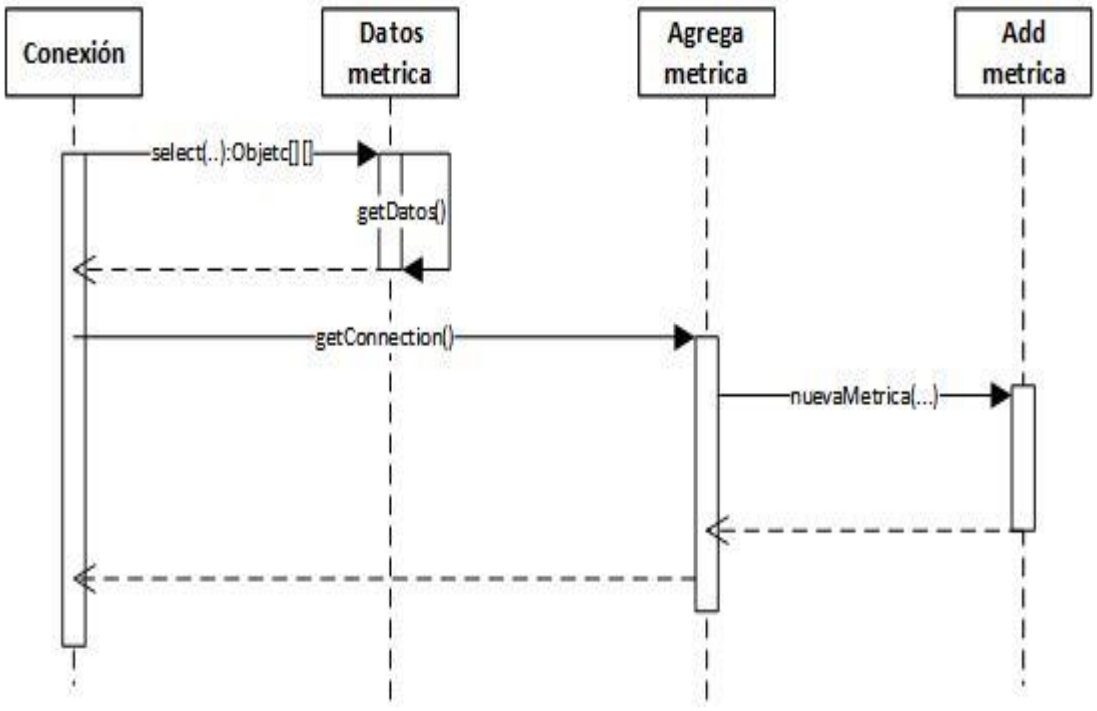

Ilustración 11 Añadir métrica.

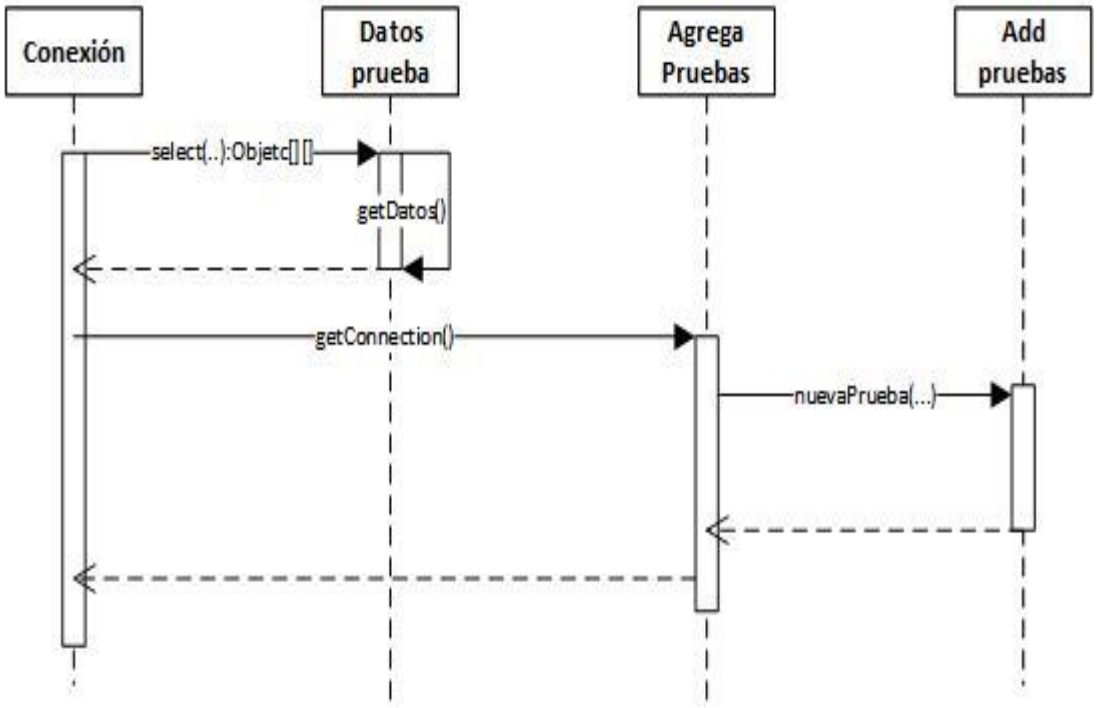

Ilustración 12 Añadir prueba.

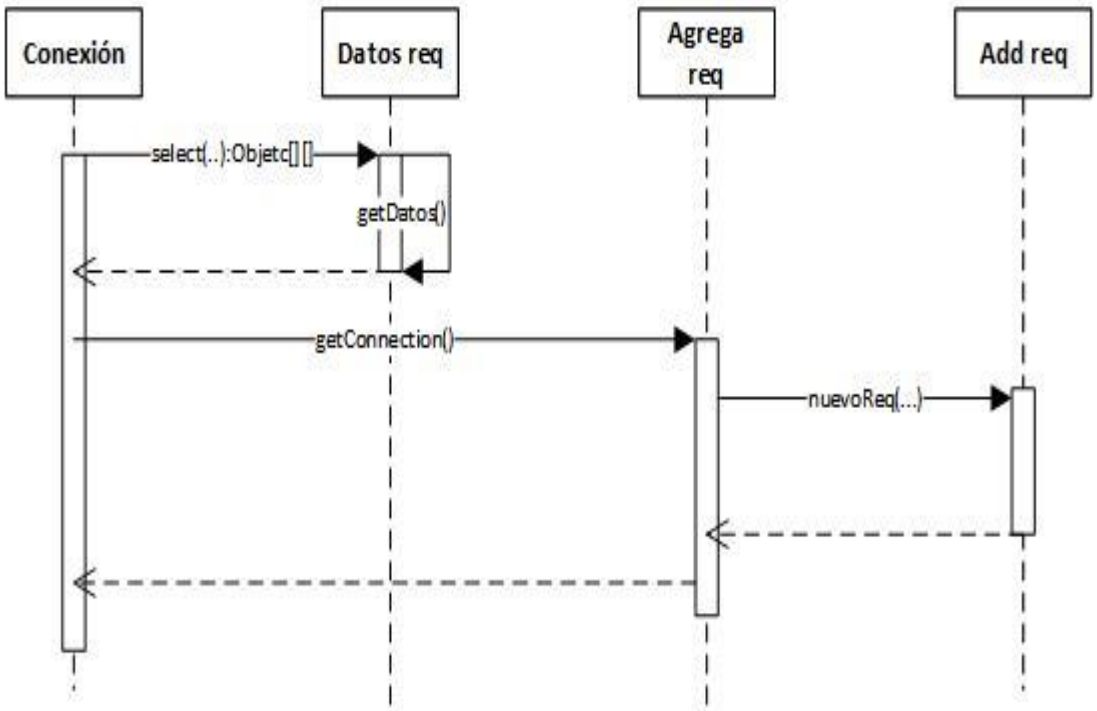

Ilustración 13 Añadir requerimiento.

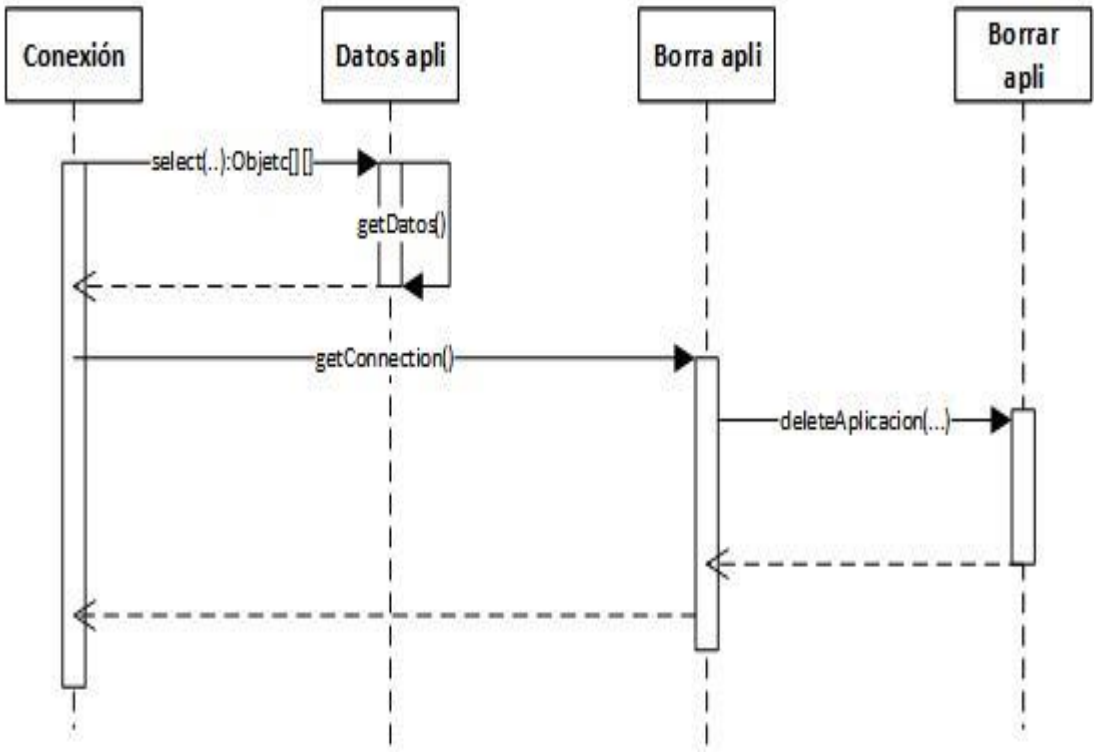

Ilustración 14 Borrar aplicación.

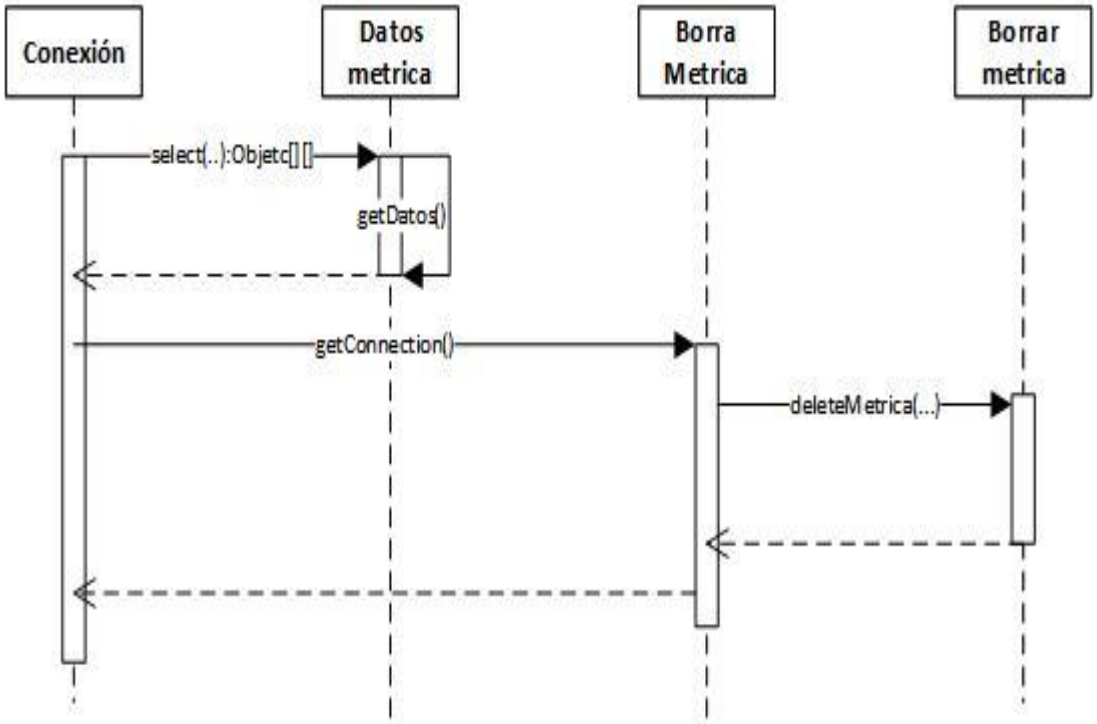

Ilustración 15 Borrar métrica

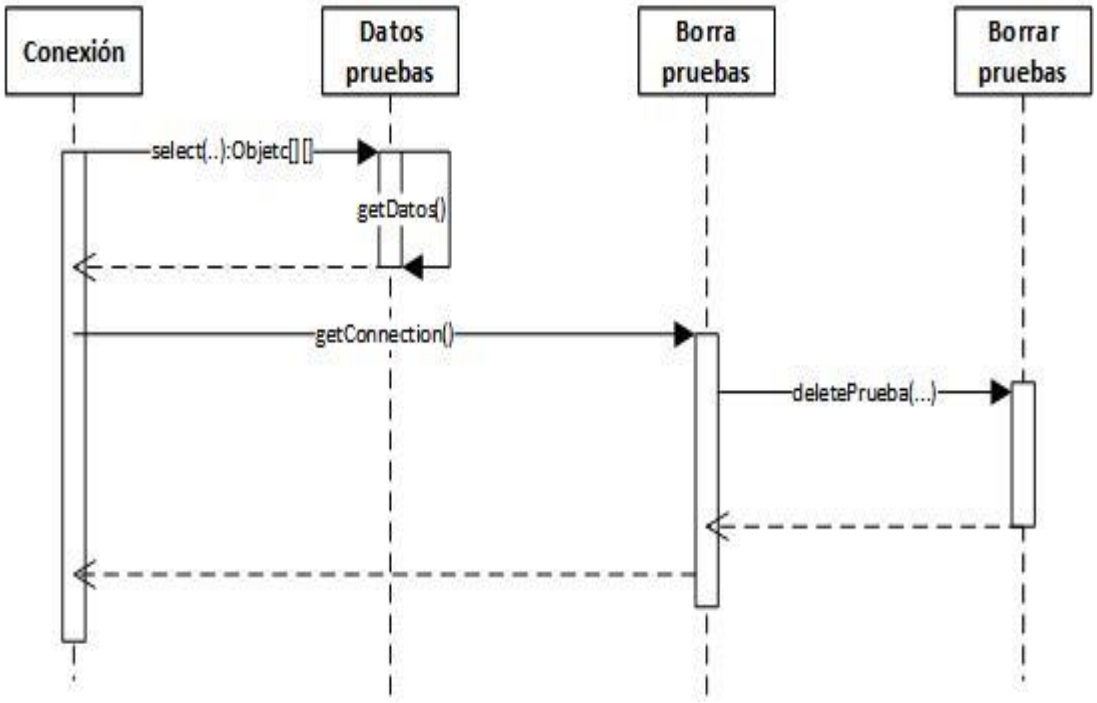

Ilustración 16 Borra prueba.

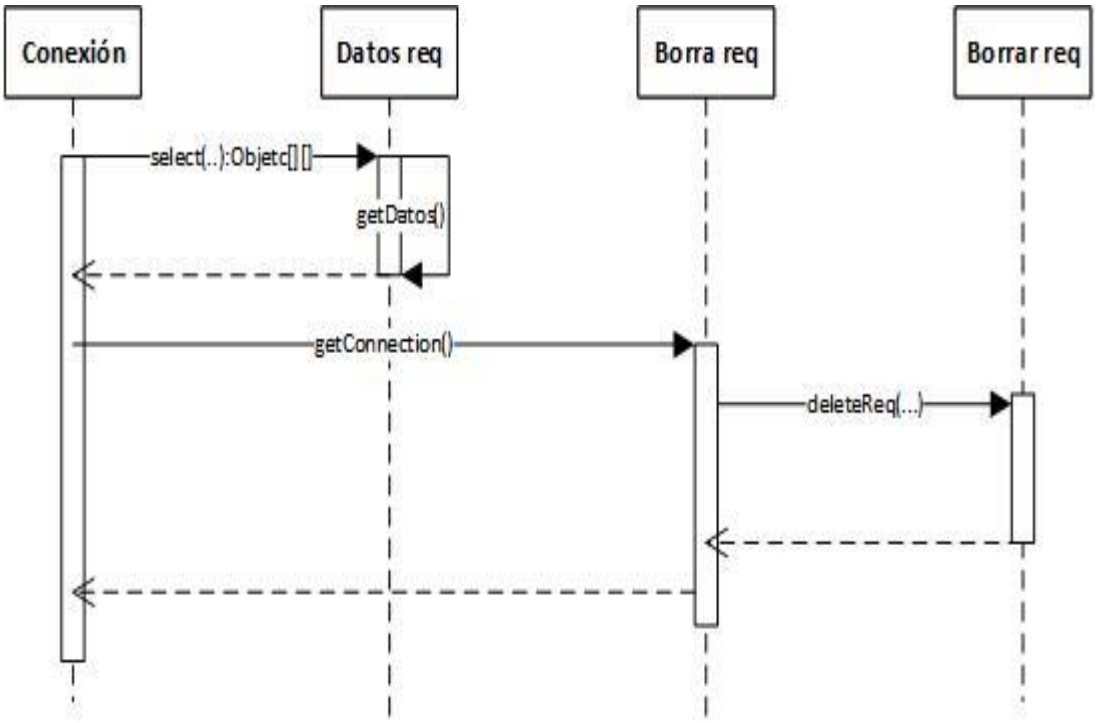

Ilustración 17 Borra requerimiento.

10. Anexo C

Imágenes del proceso para obtener los diagramas a través de la aplicación UMBRELLO.

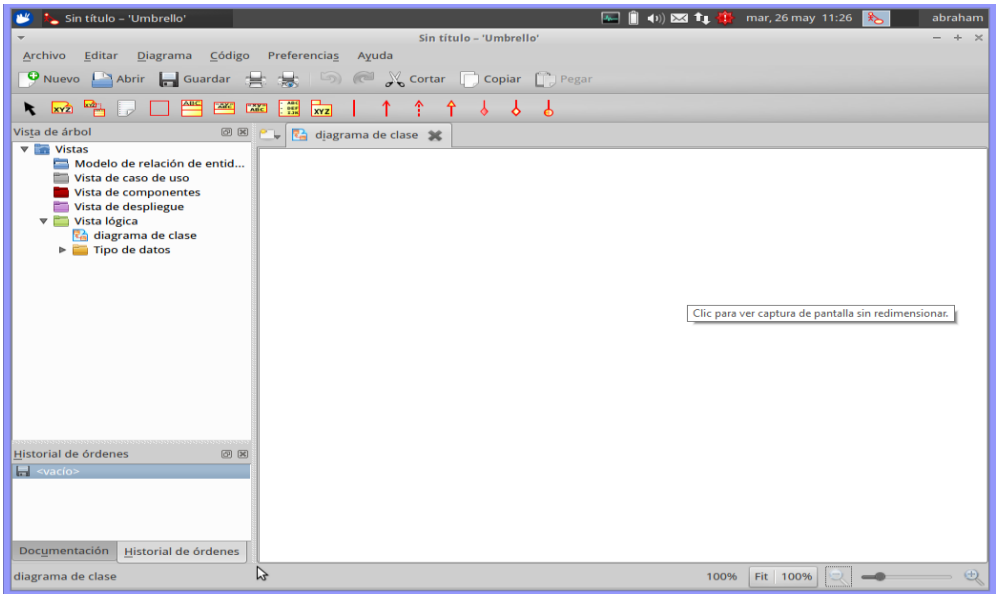

Ilustración 18 Pantalla de inicio de la aplicación Umbrello

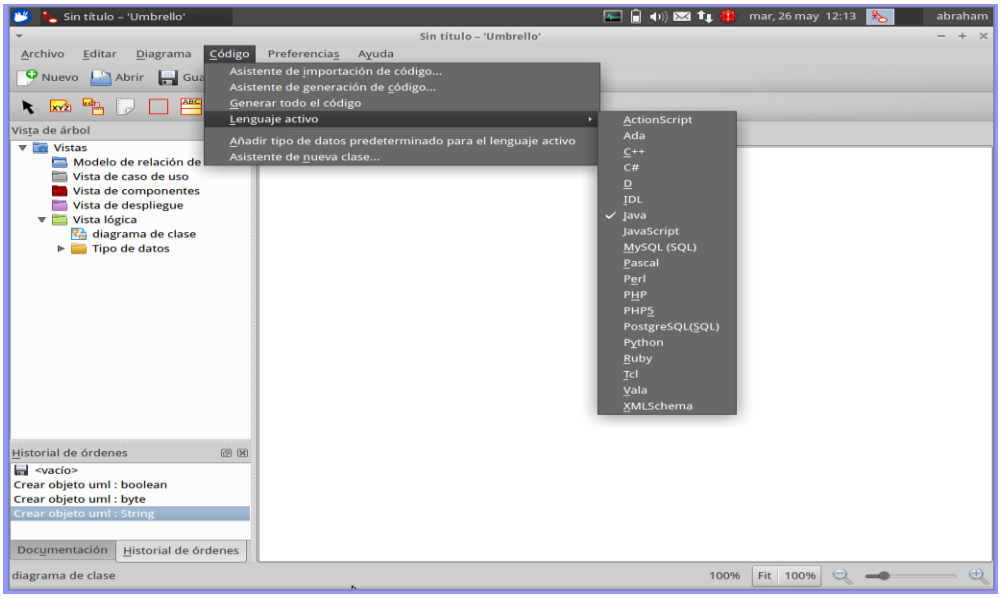

Ilustración 19 Selección del lenguaje de programación

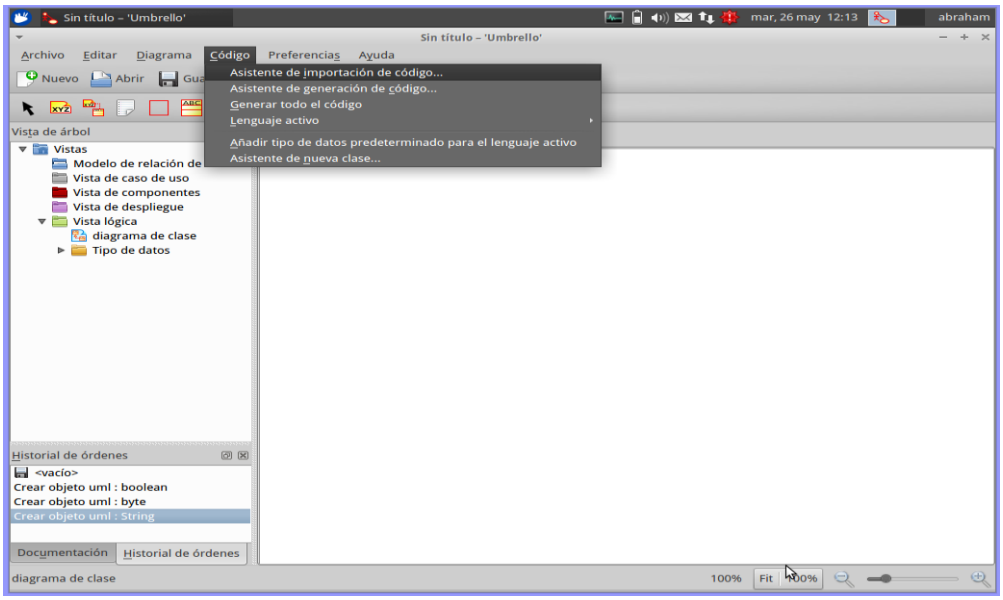

Ilustración 20 Importación de código etapa 1

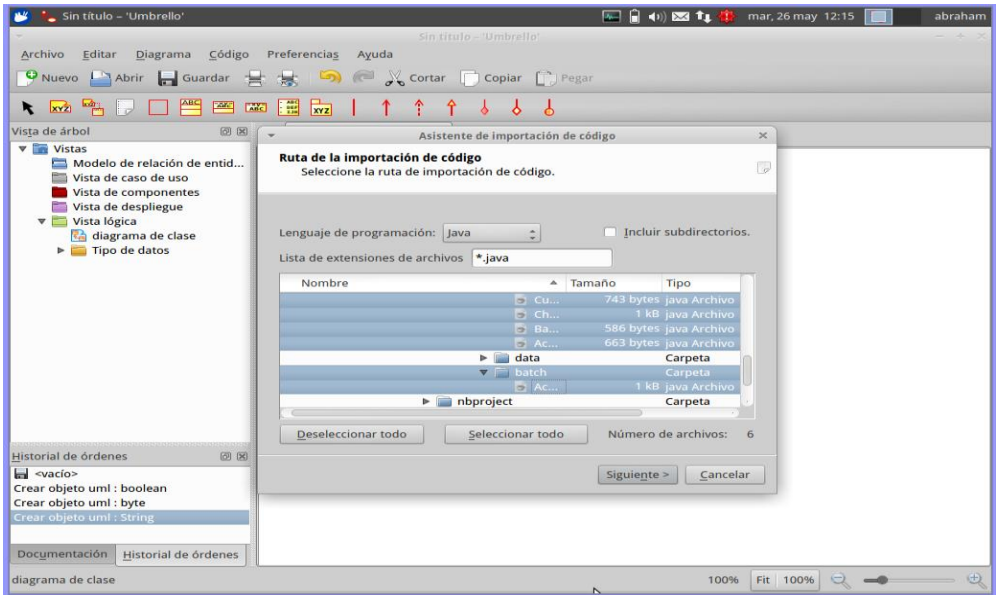

Ilustración 21 Importación de código etapa 2

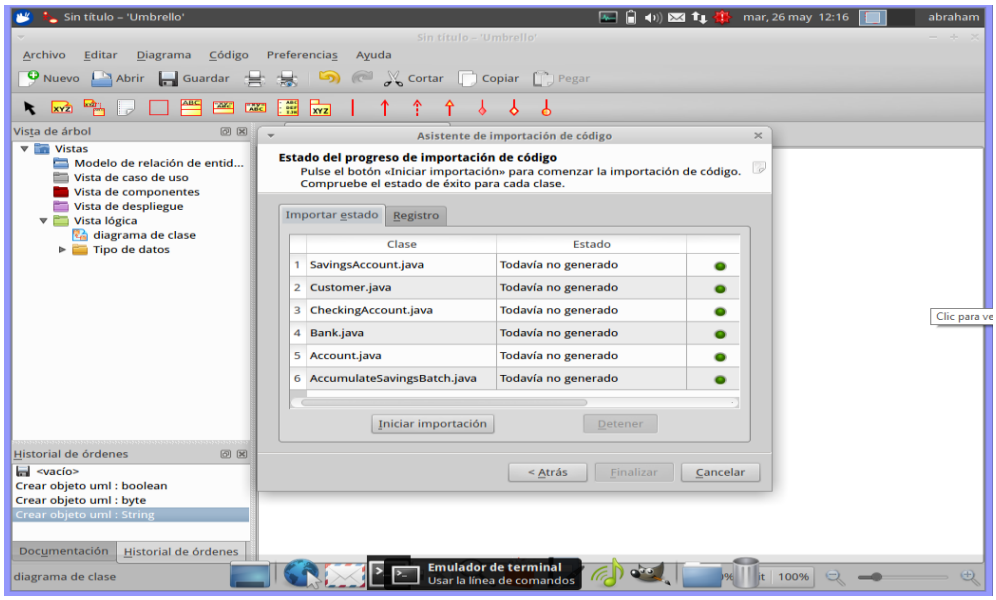

Ilustración 22 Importación de código etapa 3

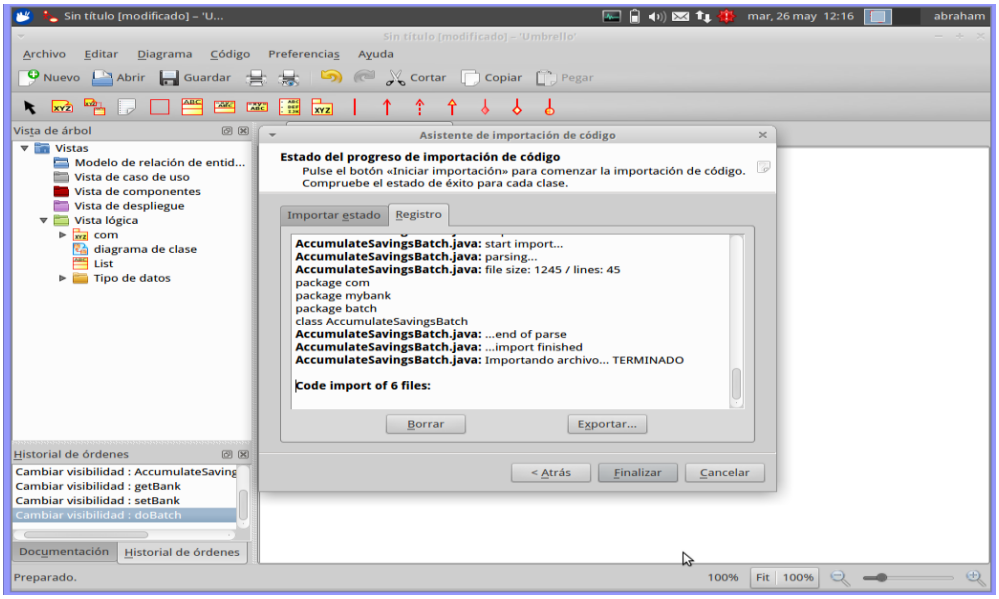

Ilustración 23 Importación de código etapa 4

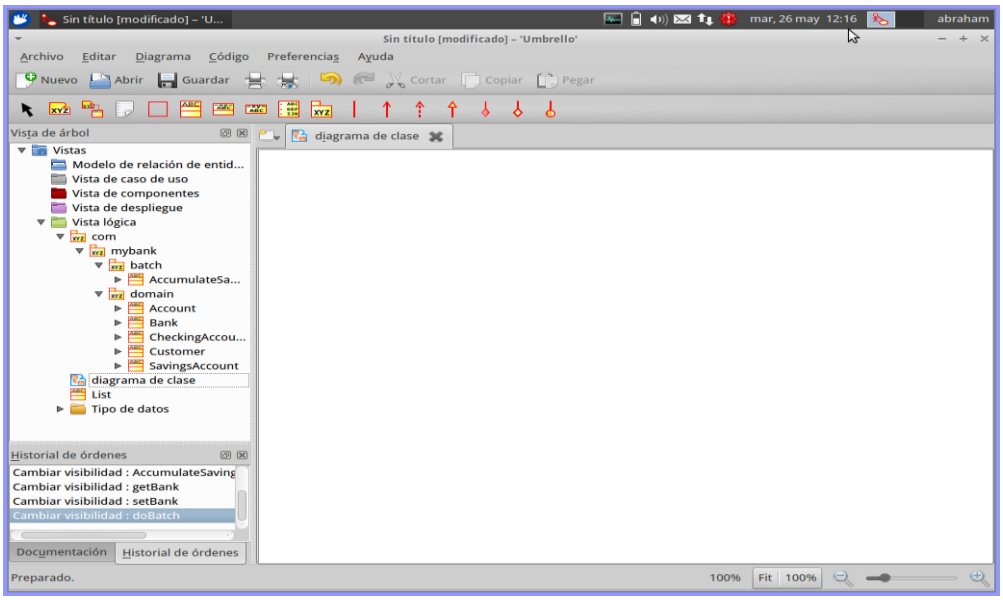

Ilustración 24 Importación de código terminada

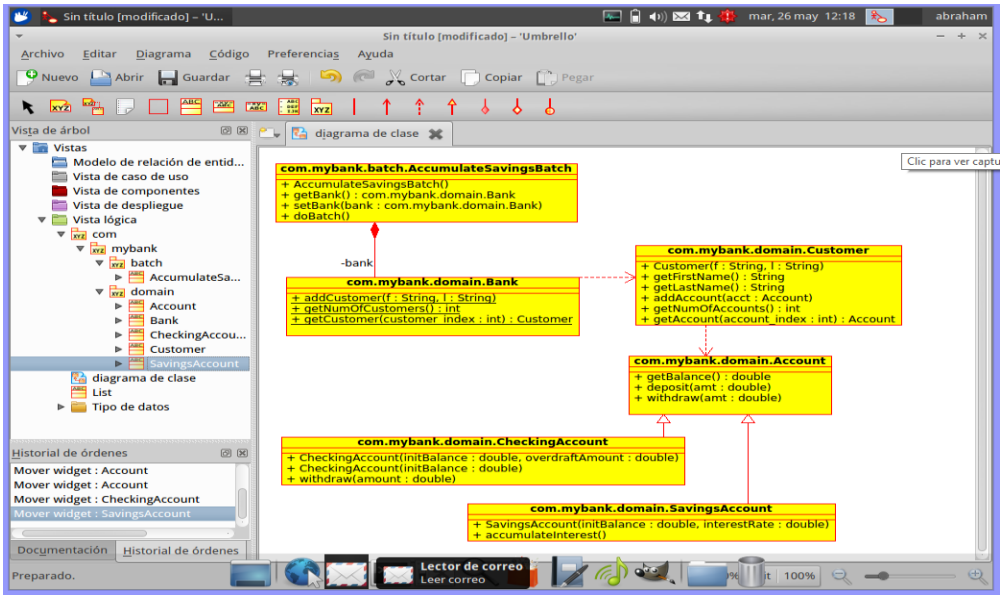

Ilustración 25 Generación de diagramas de clase

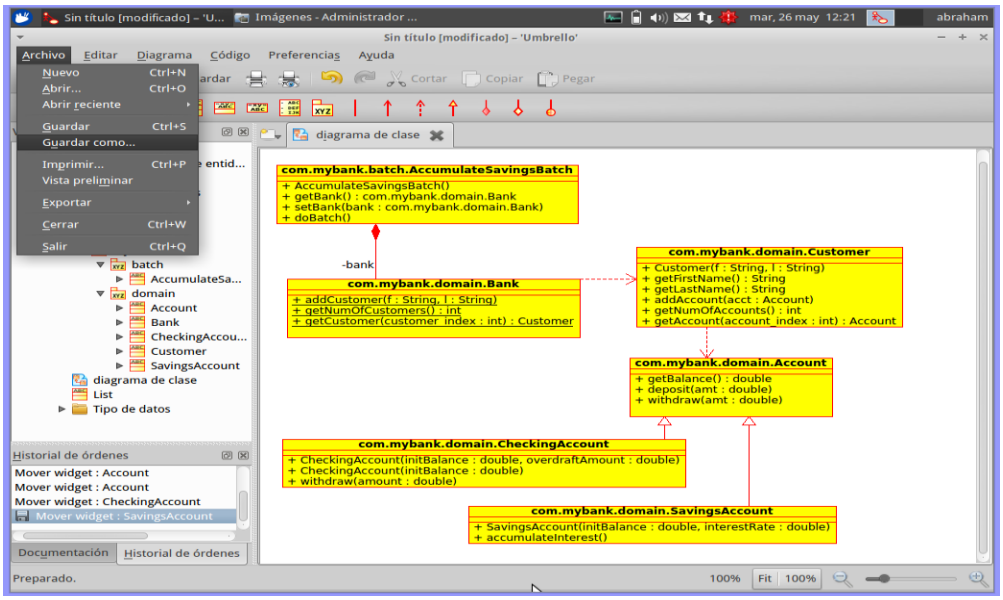

Ilustración 26 Guardar proyecto fase 1

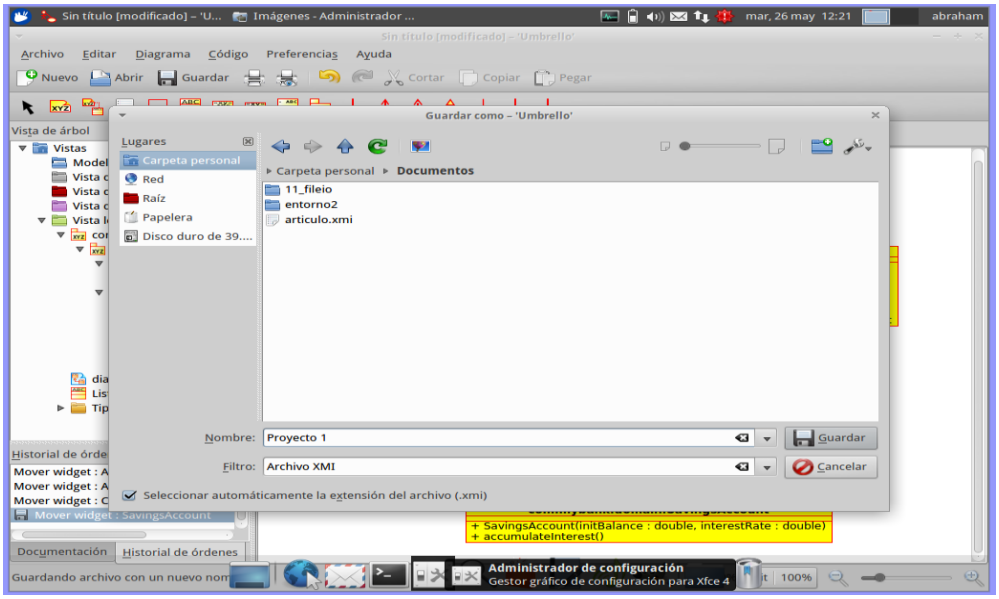

Ilustración 27 Guardar proyecto fase 2

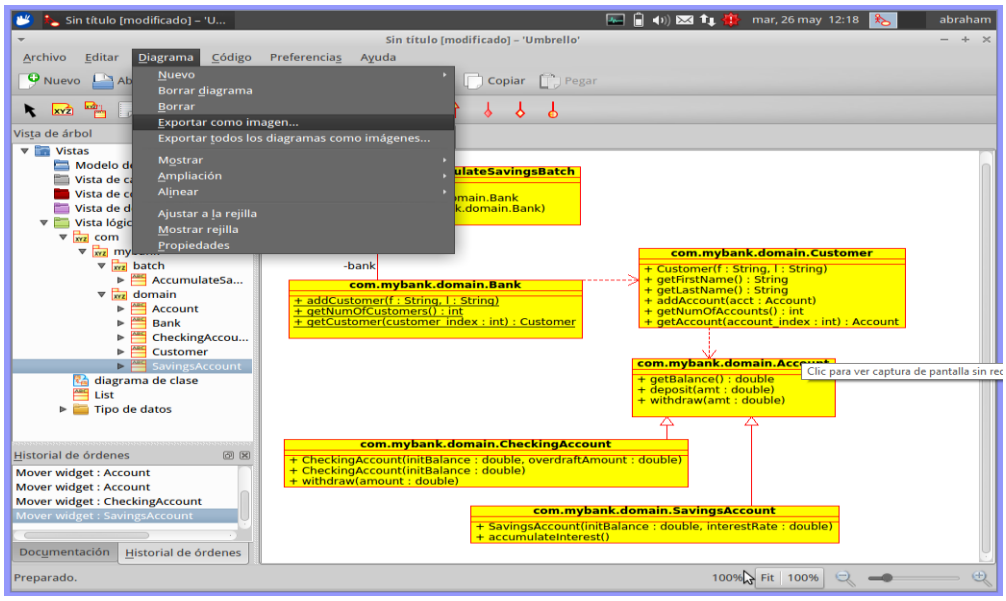

Ilustración 28 Exportar imagen fase 1

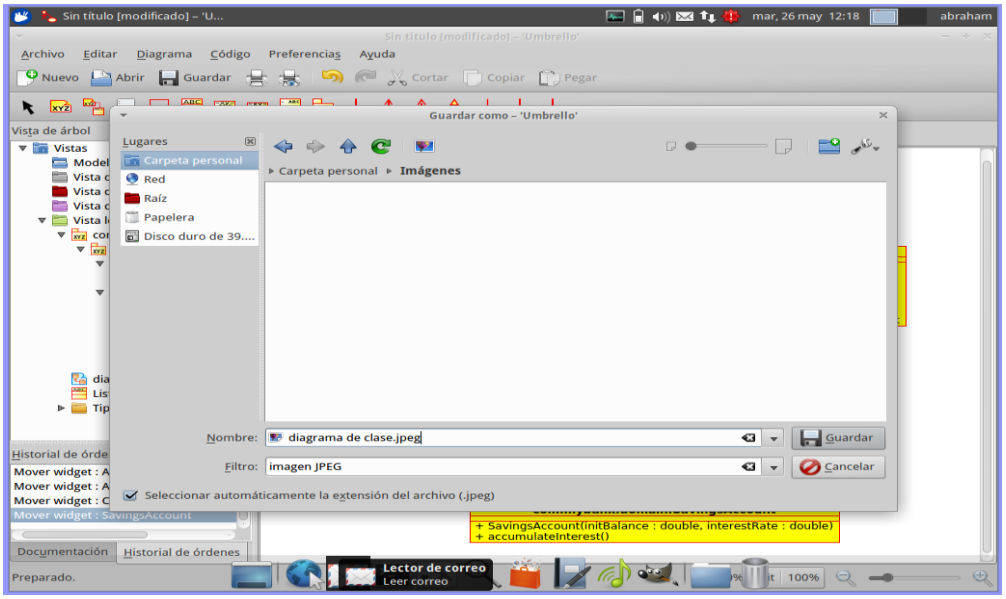

Ilustración 29 Exportar imagen fase 2

11. Referencias

A. Yadav, R. A. (2010). Does Coupling Really Affect Complexity? *International Conference on Computer and Communication Technology (ICCCT)*, (págs. 583-588). Allahabad, Uttar Pradesh.

Abreu. (1996). Evaluating the impact of object oriented design on software quality. *Proceedings of 3rd international software metrics symposium*, 90-99.

Ahmet Okutan, O. T. (2014). Software defect prediction using Bayesian networks. *Empirical Software Engineering, 19*(1), 154-181.

Arboleda Adrian, Z. G. (2012). Software for fault diagnosis using knowledge models in petri nets. *Dyna, 79*(173), 96-103.

Arthur Benjamin, G. C. (2015). *The fascinating world of graph theory.* New Jersey: Princenton.

Atchara Mahaweerawat, P. S. (2004). Fault prediction in obect-oriented software using neural network techniques. 27-34.

Banda Madrid Abraham, D. N. (2015). Predicción de fallos en un sistema de software orientado a objetos mediante su arquitectura. *CIINDET* .

Banda Madrid Abraham, L. D. (2017). Análisis y evaluación de la calidad de diseño de sistemas de software orientados a objetos. *CIM*.

Béla Újházi, R. F. (2010). New Conceptual Coupling and Cohesion Metrics for Object-Oriented Systems. *10th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM).* Timisoara .

Cass, S. (2014). Top 10 Programming Languages. *IEEE Spectrum*. Recuperado el 11 de Agosto de 2014

Chidamber, S. R. (1994). A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on 20.6*, 476-493.

Chitra S. Atole, K. V. (2006). Assessment of Package Cohesion and Coupling Principles for Predicting the Quality of Object Oriented Design. *1st International Conference on Digital Information Management* , (págs. 1-5). Bangalore.

Davila, L. (Junio de 2008). Evaluación y análisis de la calidad en los sistemas en Internet. *Tesis Doctoral.* Centro de investigación y de estudios avanzados del IPN.

Da-wei, E. (2007). The Software Complexity Model and Metrics for Object-Oriented. *IEEE International Workshop on Anti-counterfeiting, Security, Identification.* Xiamen, Fujian .

Gomaa, H. (2013). *Software Modeling & Design.* New York: Cambridge University Press.

Google. (27 de Marzo de 2012). *Google Inc.* (Google Developers) Recuperado el 5 de Septiembre de 2014, de https://developers.google.com/java-dev-tools/codepro/doc/

Group, O. M. (22 de Mayo de 2015). *Unified Modeling Language™ (UML®) Resource Page*. Obtenido de http://www.uml.org/

Halim, A. (2013). Predict Fault-Prone Classes using the Complexity of UML Class Diagram. *International Conference on Computer, Control, Informatics and Its Applications (IC3INA).* Jakarta .

I.Sommerville. (2011). *Ingeniería de Software.* México: Pearson Education.

IEEE. (2009). *IEEE Standard for a Software Quality.* New York.

Irrazába Emmanuel, G. J. (2010). Ánalisis de métricas y herramientas de código libre para medir la mantenibilidad. *Revista Española de Innovacion, Calidad e Ingeniería de Software, 6*(3), 56-65.

J. Rumbaugh, I. J. (2000). *El lenguaje unidicado de modelado. Manual de referencia.* Madrid: Pearson Education .

Jim Arlow, I. N. (2005). *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design.* Westford, Massachusetts: Pearson Education .

Kan, S. H. (2003). *Metrics and models in software quality engineering.* Pearson Education.

KDE. (5 de octubre de 2014). *Umbrello - The UML Modeller*. Obtenido de Umbrello Project: https://umbrello.kde.org/

Ladan Tahvildari, A. S. (2000). Categorization of Object-Oriented Software Metrics. *Canadian Conference on Electrical and Computer Engineering*, (págs. 235-239). Halifax, NS.

Lalji Prasad, A. N. (2009). EXPERIMENTAL ANALYSIS OF DIFFERENT METRICS (OBJECT-OREINTED AND STRUCTURAL) OF SOFTWARE. *First International Conference on Computational Intelligence, Communication Systems and Networks (CICSYN)*, (págs. 235-240). Indore.

Lionel C. Briand, J. D. (1998). Predicting Fault-Prone Classes with Design Measures in Object-Oriented Systems. *The Ninth International Symposium on Software Reliability Engineering.* Paderborn .

Lorenz Mark, K. J. (1994). *Object-Oriented Software Metrics.* Prentice Hall.

Lorenz, M. a. (1994). *Object-oriented software metrics: a practical guide.* Prentice-Hall, Inc.

M. Iyapparaja, D. S. (2012). Coupling and Cohesion Metrics in Java for Adaptive Reusability Risk Reduction. *IET Chennai 3rd International on Sustainable Energy and Intelligent Systems (SEISCON 2012), .* Tiruchengode .

McCabe, T. J. (1976). A complexity Measure. *IEEE Transactions on Software Engineering*, 308-320.

Melo, F. B. (1996). Evaluating the Impact of Object-Oriented Design on Software Quality. *3rd International Software Metrics Symposium (METRICS'96), IEEE*.

Meng-Lai Yin, J. P. (2004). Software Complexity Factor in Software Reliability Assessment. *Reliability and Maintainability, 2004 Annual Symposium - RAMS .*

Mukesh Bansal, P. C. (2014). Critical Analysis of Object Oriented Metrics in Software Development. *Fourth International Conference on Advanced Computing & Communication Technologies (ACCT)*, (págs. 197-201). Rohtak .

Musa, J. D. (2004). *Software Reliability Engineering.* Authorhouse.

Nancy Leveson, C. T. (1993). The investigation of the Therac-25 Accidents. *Computer*, 18-41. Obtenido de http://www.cs.jhu.edu/~cis/cista/445/Lectures/Therac.pdf

Norman Fenton, J. B. (2015). *Software Metrics.* United States: CRC Press.

Norman Fenton, M. N. (2007). Predicting software defects in varying development lifecycles. *ScienceDirect, 49*, 32-43.

Park, R. E., Goethert, W. B., & Florac, W. A. (1996). *Goal-Driven Software Measurement —A Guidebook.* Pittsburgh: Software Engineering Institute Carnegie Mellon University.

Perez, R. D. (7 de 11 de 2019). *Slideserve.* Obtenido de Slideserve: https://www.slideserve.com/sandro/an-lisis-de-algoritmos

Pradeep Singh, S. V. (2012). Empirical Investigation of Fault Prediction Capability of Object Oriented Metrics of Open Source Software. *International Joint*

Conference on Computer Science and Software Engineering (JCSSE). Bangkok .

Puneet Kumar Goyal, G. J. (2014). QMOOD metric sets to assess quality of java program. *International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT).* Ghaziabad .

Robert E. Park, W. B. (1996). *Goal-Driven Software Measurement A Guidebook.* Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

Sahar R. Ragab, H. H. (2010). Object Oriented Design Metrics and Tools A survey. *The 7th International Conference on Informatics and Systems (INFOS).* Cairo.

Sánchez, A. B.-C. (2013). Priorización de casos de prueba. Avances y retos. *Novática: Revista de la Asociación de Técnicos de Informática*, 27-32.

Santosh Singh Rathore, A. G. (2012). Investigating object-oriented design metrics to predict fault-proneness of software modules. *Sixth International Conference on Software Engineering (CONSEG)*, (págs. 1-10). Indore.

Shatnawi, R. (2014). Empirical study of fault prediction for open-source systems using the Chidamber and Kemerer metrics. *IET Software, 8*, 113-119.

Shyam R. Chidamber, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 476-493.

Sukainah Husein, A. O. (2009). A Coupling and Cohesion Metrics Suite for Object-Oriented Software. *International Conference on Computer Technology and Development*, (págs. 421-425). Kota Kinabalu.

Toolworks, S. (18 de Octubre de 2014). *Understand*. Obtenido de Scientific Toolworks, Inc: https://scitools.com/

Tu Honglei, S. W. (2009). The Research on Software Metrics and Software Complexity Metrics. *International Forum on Computer Science-Technology and Applications.* Chongqing .

Valenzuela, J. (2 de otubre de 1999). *El Pais.* Obtenido de El Pais Web site: http://elpais.com/diario/1999/10/02/sociedad/938815207_850215.html

Vibhash Yadav, R. S. (2013). Prediction design quality of object-oriented software using UML diagrams. *3rd International Advance Computing Conference (IACC)*, (págs. 1462-1467). Ghaziabad .

Villalobos, A. R. (20 de Abril de 2012). *Grafos Software para la construcción, edición y análisis de grafos*. Obtenido de http://arodrigu.webs.upv.es/grafos/doku.php?id=inicio

Wagner, S. (2010). A Bayesian network approach to assess and predict software quality using activity-based quality models. *Journal Information and Software Technology, 52*(11), 1230-1241.

Weisfeld, M. (2013). *The Object-Oriented Thougth Process.* Pearson Education.

Weyuker. (1998). Evaluating Software Complexity Metrics. *IEEE Transactions on Software Engineering*, 1357-1365.

Winter Victor, R. C. (2013). Sextant: A Tool to Specify and Visualize Software Metrics for Java Source-Code. *4th International Workshop on Emerging Trends in Software Metrics (WeTSOM 2013)*, (págs. 49-55). San Francisco.

XIA, F. (1996). Module Coupling: a design metric. *Asia-Pacific Software Engineering Conference.* Seoul .

Yamada, S. (2014). *Software Reliability Modeling Fundamentals and Applications.* Tottori, Japan: Springer.

Yuming Zhou, H. L. (2006). Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 771-789.