**Universidad de las Ciencias Informáticas**

**Facultad 5**

**Centro de Consultoría y Desarrollo de Arquitecturas Empresariales**


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

# Mecanismo de captura de metadatos para el Replicador de datos REKO

**Autor:** Javier José Marín Morales

**Tutora:** Ing. Nayibi Martín Peña

“Año 58 de la Revolución”
La Habana, Cuba
Julio 2015

“La escalera del éxito no está aglomerada en lo más alto.”

Napoleón Hill

## Declaración de autoría

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

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

_____
Javier José Marín Morales
**Autor**

_____
Ing. Nayibi Martín Peña
**Tutor**

## Datos de contacto

**Autor:** Javier José Marín Morales.
**Institución:** Universidad de las Ciencias Informáticas (UCI), La Habana, Cuba
**E-mail:** jjmarin@estudiantes.uci.cu

**Tutor:** Nayibi Martín Peña.
**Institución:** Universidad de las Ciencias Informáticas (UCI), La Habana, Cuba
**Síntesis del Tutor:** Ingeniera en Ciencias Informáticas, UCI 2013. Especialista “B” en Ciencias Informáticas, UCI.
**E-mail:** nmartin@uci.cu

## Dedicatoria

Dedico esta tesis a toda mi familia por su apoyo en todo momento en especial a mis padres, a mi hermana, a mi hermano, a mi sobrina, a mi tío Ramiro y a mi tía KIKA.

## Agradecimientos

Agradecerles ante todo a mis queridos padres por creer siempre en mí, por todo el apoyo y amor que me han brindado en toda mi vida y por haber sabido guiarme por el camino correcto y ser mis ejemplos.

A mi hermana y hermano por formar parte de mí en todo momento y ayudarme en cualquier situación.

A mi tía KIKA y mi tío Ramiro por complacerme en todos mis caprichos.

A toda mi familia que son muchos por siempre estar presente en los momentos más difíciles de estos 5 años.

A mi profe guía Zaida por estar siempre presente.

A mi tutora por toda su ayuda en el desarrollo de esta tesis.

A mis antiguos compañeros de grupo Bravo, Alexis, Ever y Arnaldo por siempre tener un tema para pasarnos horas discutiendo.

A Yoanita por ser siempre mi amiga incondicional.

A mis amigos Orson, Erduin, Yandry, Mirnerys, Ernesto que más que mis amigos se convirtieron en mi segunda familia.

## Resumen

El Replicador de datos REKO es un software de réplica de datos entre bases de datos relacionales. Posee un mecanismo para la captura de metadatos de una base de datos, el cual permite capturar información de tablas y columnas para la réplica de estructuras, siendo este ineficiente pues no se logra capturar toda la información de la estructura de la base de datos. El presente trabajo de diploma propone un mecanismo para capturar más información de la estructura de una base de datos.

El funcionamiento del mecanismo radica en la captura de los metadatos de la base de datos a la que está conectado el Replicador de datos REKO, mediante la Interfaz de Programación de Aplicaciones (API por sus siglas en inglés) *DatabaseMetaData* y sentencias directas al gestor de base de datos, para todos los sistema de gestión de base de datos a los que el Replicador de datos REKO brinda soporte: PostgreSQL, MySQL, Microsoft SQL Server y Oracle. Para el desarrollo se utilizaron herramientas *Open Source* y librerías de clases de licencia gratuitas, así como la metodología de desarrollo Proceso Unificado Ágil (AUP por sus siglas en inglés), por lo que se obtuvo como resultado un mecanismo capaz de capturar toda la estructura de la base de datos, el cual permitirá en un futuro realizar la réplica de estructura de todos los componentes de una base de datos.

**Palabras Claves:** estructura de una base de datos, metadatos, réplica de estructura.

# Índice de contenidos

# Índice de figuras

# Índice de tablas

# Introducción

En la actualidad el almacenamiento de datos es fundamental para toda empresa u organización, que pretenda el uso estadístico de la información generada por sus procesos. El almacenamiento de esta información se realiza utilizando Bases de Datos (BD) almacenadas digitalmente, de manera tal que se pueda introducir, organizar y manipular los datos para poder extraer información significativa. Las BD almacenadas digitalmente han sido para las organizaciones una herramienta fundamental para garantizar la persistencia de datos, así como ofrecer una solución para los problemas de acumulación de datos físicos.

Con el desarrollo tecnológico, la necesidad de poder acumular grandes volúmenes de datos fue en aumento, trayendo consigo un incremento en el tiempo de búsqueda. Como una fuente viable para darle solución a este tipo de problema, se comienzan a utilizar Sistemas de Bases de Datos Distribuidas (SBDD), los cuales representan la estructura geográfica descentralizada de una organización, donde cada base de datos distribuida está compuesta por nodos de una red de computadoras.

Con el uso de SBDD, ha ido en aumento la necesidad de utilizar sistemas de réplicas. Estos sistemas permiten el proceso de propagación de los datos y la integridad de estos en el momento en punto de realizar actualizaciones entre los nodos distribuidos.

En la Universidad de las Ciencias Informáticas (UCI) y como fruto del desarrollo de aplicaciones dentro del Centro de Consultoría y Desarrollo de Arquitecturas Empresariales (CDAE) se encuentra Replicador de datos REKO, que es una herramienta de réplica de datos entre BD relacionales surgida a partir de la necesidad que se presentó en la distribución de datos sobre Oracle[1] 10g en el Sistema de Gestión Penitenciario Venezolano (SIGEP)[2].

El Replicador de datos REKO cuenta con un mecanismo que permite capturar los metadatos de la BD que se desea replicar para todos los Sistemas de Gestión de Base de Datos (SGBD) a los que brinda soporte: PostgreSQL, MySQL, Microsoft SQL Server y Oracle. Actualmente se logra capturar la información de las tablas y de las columnas. Sin embargo, este mecanismo se encuentra incompleto debido a que las definiciones de las tablas y columnas no almacenan toda

[1] Oracle Corporación es una de las mayores compañías de software del mundo. Sus productos van desde bases de datos hasta sistemas de gestión.

[2] Software desarrollado por la UCI para la República Bolivariana de Venezuela.

su información, además no captura información de otras estructuras como son vistas (*view*), disparadores (*trigger*), índices (*index*), reglas(*rule*), secuencias (*sequence*), restricciones (*constraints*), funciones (*function*).

Al no capturar la estructura completa de la BD durante la réplica de estructura, la información enviada al nodo destino, no contiene la información real de la BD del nodo origen, provocando errores de inconsistencias entre las estructuras de ambas BD. La réplica de datos se ve afectada por conflictos surgidos al intentar aplicar cambios de datos en estructuras incompletas. Además los cambios de estructura podrían enviarse parcialmente, lo que implicaría que la información enviada pierda veracidad, y por ende, existirían errores en el proceso de réplica de estructura.

Dada la **situación problemática** planteada, el **problema a resolver** radica en: ¿Cómo obtener la información necesaria de la estructura de una base de datos para el proceso de réplica de estructura empleando el Replicador de datos REKO?

Para la realización de la investigación se toma como **objeto de estudio** el proceso de captura de metadatos de una BD. Delimitado por el **campo de acción** el proceso de captura de metadatos de una BD en el Replicador de datos REKO.

Para darle solución al problema planteado se define como **objetivo general** desarrollar un mecanismo para la captura de metadatos de una base de datos que permita obtener la información necesaria para el proceso de réplica de estructura en el Replicador de datos REKO.

Teniendo en cuenta el objetivo general se plantea como **idea a defender** que el desarrollo de un mecanismo de captura de metadatos para el Replicador de datos REKO permitirá capturar la información necesaria de la estructura de una BD, por lo que constituirá una solución factible y robusta ante las necesidades de réplica de estructuras de los distintos sistemas que utilicen el Replicador de datos REKO.

Para darle cumplimiento a dichos objetivos se proponen las siguientes **tareas de investigación:**

1. Realización del marco conceptual para precisar los principales conceptos que se emplean en la investigación.
2. Elaboración del estado del arte de las herramientas que necesiten dentro de sus procesos la obtención de metadatos, para sintetizar los resultados alcanzados en la revisión bibliográfica e indagaciones realizadas referentes al tema.

3. Descripción de las herramientas, tecnologías y lenguajes a utilizar para definir el ambiente de desarrollo.
4. Definición de los requisitos funcionales y no funcionales para identificar las capacidades que tienen que ser alcanzadas por el sistema para cumplir los objetivos trazados.
5. Descripción de la arquitectura que soporta la implementación de las funcionalidades.
6. Realización del análisis y diseño del mecanismo para describir los artefactos.
7. Implementación de las funcionalidades que dan cumplimiento a los requisitos identificados.
8. Realización de pruebas al mecanismo en entornos de producción para valorar la calidad del producto implementado.
9. Valoración de los resultados obtenidos.

Para realizar la investigación se hará uso de los métodos científicos, entre ellos los **métodos teóricos**:

1. **Análisis Histórico-Lógico:** el cual se utilizó para la realización del estudio del estado arte del tema en cuestión, posibilitando el indicar datos sobre las descripciones de las herramientas de captura de metadatos.
2. **Analítico-Sintético:** el cual se utilizó para la clasificación y resumen de la información recopilada sobre las diferentes herramientas de captura de metadatos, permitiendo así determinar la solución más apropiada al problema planteado.
3. **Análisis documental:** en la consulta de la literatura en todo tipo de fuentes, relacionada con la captura de metadatos de una BD.

Dentro de los **métodos empíricos** empleados se encuentra:

- ✓ **Observación:** permitió conocer la realidad mediante la percepción directa de los objetos y fenómenos relacionados con la captura de metadatos.
- ✓ **Estadística Descriptiva:** para el análisis de los resultados obtenidos al aplicar las métricas para evaluar el diseño del software y los obtenidos en las pruebas unitarias.

El trabajo de diploma está estructurado por: introducción, tres capítulos, conclusiones, recomendaciones, bibliografía y anexos.

- ✓ **Capítulo 1.** Fundamentación Teórica: en este capítulo se define la base teórica de la presente investigación, incluye las tendencias actuales en la captura de metadatos, así como las aplicaciones que se usan para ello.

- ✓ **Capítulo 2.** Características y Diseño del sistema: en este capítulo se describe la solución propuesta para darle respuesta a la situación planteada, para lo cual se definen los requisitos funcionales y no funcionales y se realiza el diseño de la solución.

- ✓ **Capítulo 3.** Implementación y Pruebas: en este capítulo se muestra parte de la implementación de la solución, especificando el estándar de codificación, así como un análisis de los resultados obtenidos en el proceso de prueba.

# Capítulo #1. Fundamentación teórica

En el presente capítulo se abordarán los diferentes conceptos relacionados con los mecanismos, metadatos, y las tecnologías de captura de metadatos. Se hará un estudio del estado del arte de estas tecnologías centrándose en el proceso de captura de metadatos de cada una de ellas. Como metodología de desarrollo de software se utilizará Proceso Unificado Ágil (AUP por sus siglas en inglés). Y por último se hará una caracterización de las herramientas y lenguajes de los que se hará uso para la implementación de la solución propuesta.

## 1.1 Marco Conceptual

A continuación se muestran algunos conceptos tratados durante el transcurso de la investigación, a fin de introducir el tema y lograr una mejor comprensión del trabajo en cuestión.

### 1.1.1 Mecanismo

Un mecanismo es una combinación de operadores cuya función es producir, transformar o controlar un movimiento. Los mecanismos se construyen encadenando varios operadores mecánicos entre sí, de tal forma que la salida de uno se convierte en la entrada del siguiente.[1]

### 1.1.2 Metadatos

Un metadato no es más que un dato estructurado sobre la información, o sea, información sobre información, o de forma más simple, datos sobre datos. Los metadatos en el contexto de la Web, son datos que se pueden guardar, intercambiar y procesar por medio del ordenador y que están estructurados de tal forma que permiten ayudar a la identificación, descripción, clasificación y localización del contenido de un documento o recurso web y que, por tanto, también sirven para su recuperación. [2]

### 1.1.3 Application Program Interface (API)

Interfaz de Programación de Aplicaciones. Conjunto de convenciones internacionales que definen cómo debe invocarse una determinada función de un programa desde una aplicación. Cuando se intenta estandarizar una plataforma, se estipulan unos APIs comunes a los que deben ajustarse todos los desarrolladores de aplicaciones. Herramientas de programación para rutinas, protocolos y software.[3]

## 1.2 Tecnologías de captura de metadatos

### Estándar SQL

Este estándar habilita la portabilidad de aplicaciones SQL y es el resultado del esfuerzo de muchas organizaciones como Computer Associates, IBM, Informix, Oracle, Sybase, Microsoft, entre otros. En este estándar cada catálogo contiene un esquema de información denominado *INFORMATION _SCHEMA*, que incluye la descripción de los objetos contenidos en la base de datos. Entre los objetos que se proponen en este estándar se tienen las vistas: *COLUMNS, TABLES*, *TRIGGERS*, *VIEWS*, entre otras [4].

#### Open DataBase Connectivity

ODBC (Conectividad de Base de Datos Abierta) es un estándar desarrollado originalmente por Microsoft, que permite que las aplicaciones se conecten a diferentes orígenes de datos a través de una única interfaz. ODBC usa una interfaz escrita con el lenguaje de programación C, por lo tanto como C no es un lenguaje portable las aplicaciones JAVA perderían también automáticamente su portabilidad. ODBC se ha de instalar manualmente en cada máquina, en cambio los drivers de JDBC[3] como están escritos en JAVA son automáticamente instalables, portables y seguros.[4]
Este estándar posee algunas funciones que permiten obtener información acerca de los metadatos, como por ejemplo: SQLDescribeCols, que permite conocer el nombre, tipo de dato, precisión, escala y nulidad de las columnas de una tabla.

#### Microsoft ActiveX Data Objects (ADO)

Es la estrategia de Microsoft para proveer acceso a múltiples orígenes de datos relacionales y no relacionales, esto lo logra a través de OLE DB (Objeto Enlace Integración de Bases de Datos). OLE DB es una colección de interfaces que encapsulan los servicios de los administradores de orígenes de datos, y ha sido desarrollado siguiendo la misma filosofía de ODBC en el sentido de ser una especificación abierta. La diferencia radica en que mientras ODBC fue creado para dar soporte de acceso a bases de datos relacionales, OLE DB ha sido diseñado para orígenes de datos relacionales y no relacionales.[4]

#### Interfaz DatabaseMetaData

Esta interfaz es implementada por los proveedores de controladores para que los usuarios conozcan las capacidades y/o características de un Sistema de Gestión de Bases de Datos (SGBD) en combinación con el controlador basado en la tecnología JDBC™ ("*driver* JDBC") que se utiliza con ella. Diferentes

[3] Conector a Bases de Datos de JAVA (JDBC, por sus siglas en inglés).

SGBD a menudo admiten diferentes características, permiten implementar características de diferentes maneras, y el uso de diferentes tipos de datos. La información devuelta por los métodos de esta interfaz se aplica a las capacidades de un controlador particular y un SGBD particular que trabajen juntos [5] .

La interfaz *DatabaseMetaData* incluye alrededor de 150 métodos, que se pueden clasificar en categorías en función de los tipos de información que proporcionan. La interfaz *DatabaseMetaData* también contiene alrededor de 40 campos, que son constantes empleadas como valores de retorno en los diversos métodos de *DatabaseMetaData*.[6]

Diversos métodos de *DatabaseMetaData* proporcionan información sobre los objetos SQL. Estos pueden determinar los atributos de los objetos SQL de una base de datos. Estos métodos también devuelven objetos *ResultSet*, en los que cada fila describe un objeto concreto. [6]

Después de analizar las diferentes tecnologías de captura de metadatos se selecciona para dar solución al problema planteado el uso de la API *DatabaseMetaData*. Dado que es propia del lenguaje JAVA y es la que actualmente utiliza el Replicador de datos REKO. Permite analizar y obtener información sobre la estructura de una BD a la que se está conectado. Además se hará uso del dialecto SQL, pues *DatabaseMetaData* no brinda toda la información de las diferentes estructuras que conforman una BD. Las restantes herramientas hay que instalarlas en cada una de las estaciones de trabajo y la interfaz que usan está implementada en C, por lo que se debe utilizar un puente para la comunicación con JDBC.

## 1.3 Características del Replicador de datos REKO

El Replicador de datos REKO surge ante la necesidad que presentó la distribución de datos sobre Oracle 10g en el SIGEP en el año 2007. El SIGEP necesitaba mantener actualizado el Centro de Datos con la información recopilada en todos los centros, así como todos los centros debían mantenerse actualizados en sentido contrario.[7]

El Replicador de datos REKO es un software que ha sido desarrollado en la UCI como producto para mantener actualizadas un conjunto de bases de datos y que pretende solventar las principales necesidades que tienen que ver con la distribución de datos entre los gestores más populares. Su diseño permite la réplica y sincronización de datos entre los gestores mayormente usados. Puede ser utilizado para la réplica y sincronización de datos con o sin conexión.

El Replicador de datos REKO constituye una solución informática de alto valor en la sustitución de importaciones dado que es una solución construida sobre una arquitectura en tecnologías modernas y libres.[8]

### 1.3.1 Principales Funcionalidades

Entre las principales funcionalidades que brinda el Replicador de datos REKO se encuentran las mencionadas a continuación:[7]

- ✓ soporte para réplica de datos en ambientes con conexión y sin conexión.
- ✓ soporte para réplica entre bases de datos con diferentes estructuras.
- ✓ soporte para réplica de ficheros externos a la base de datos.
- ✓ soporte para la sincronización de datos en ambientes con conexión y sin conexión.
- ✓ selección de los datos de réplica ajustada por filtros.
- ✓ soporte para réplica de datos entre gestores diferentes (PostgreSQL-Oracle-MySQL-Microsoft SQL Server).
- ✓ soporte para resolución de conflictos de forma automática y manual.
- ✓ monitoreo en tiempo real de los datos de réplica.
- ✓ réplica de datos de gran tamaño con capacidad de resumir el envío y el recibo de los mismos en caso de desconexión.
- ✓ soporte para la programación del momento de captura y envío de los datos de réplica.

### 1.3.2 Características y beneficios

Entre los principales beneficios y características que ofrece el Replicador de datos REKO, están que puede ser instalado independientemente de la plataforma que use el usuario, garantiza la seguridad en el envío de los datos con protocolos de comunicación segura, realiza la réplica de archivos de gran tamaño, además aplica el filtrado de la réplica a los datos capturado y replica datos en sistemas heterogéneos donde prevalecen diferentes gestores de bases de datos, así como detecta errores de conexión. Puede tomarse como una ventaja el hecho de que la administración y configuración de la réplica se realiza a través de una interfaz web, por lo que puede administrase de manera remota usando un navegador web, y cuenta además con una comunidad de desarrollo que respalda la integridad del software.[7]

## 1.4 Metodología de Proceso Unificado Ágil

Las metodologías para el desarrollo imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Tiene como principal objetivo aumentar la calidad del software que se produce en todas y cada una de sus fases de desarrollo, haciendo énfasis en la calidad y menor tiempo de construcción del software o lo que es lo mismo “producir lo esperado en el tiempo esperado y con el coste esperado”.[9]

Como política a seguir en el proyecto de desarrollo del Replicador de datos REKO, perteneciente al CDAE de la UCI, se decide la utilización de la metodología AUP para el desarrollo de la solución del problema anteriormente planteado.

### 1.4.1 Descripción de la metodología

AUP fue desarrollado por Scott Ambler en septiembre del 2005. Ambler previamente ha desarrollado otras metodologías ágiles. Se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y por ser iterativo e incremental. Describe un enfoque simple del desarrollo del software usando técnicas y conceptos ágiles. Algunas técnicas usadas por AUP incluyen el desarrollo orientado a pruebas, modelado y gestión de cambios ágiles, y refactorización de base de datos para mejorar la productividad.[10]

- ✓ **Características de AUP** [11]
  - Versión simplificada de la metodología Proceso Unificado de Racional (RUP).
  - Abarca siete flujos de trabajos, cuatro ingenieriles y tres de apoyo: Modelado, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de Proyectos y Ambiente.
  - El modelado agrupa los tres primeros flujos de RUP (Modelamiento del negocio, Requerimientos y Análisis y Diseño).

- ✓ **Ciclo de vida de AUP**[11]

El Proceso Unificado Ágil consta de cuatro fases que el proyecto atraviesa de forma secuencial. Dichas fases son, al igual que en el Proceso Unificado de Racional (RUP):

- Iniciación: El objetivo de esta fase es identificar el alcance inicial del proyecto, una arquitectura potencial para el sistema y obtener, si procede, financiación para el proyecto y la aceptación por parte de los promotores del sistema.
- Elaboración: Mediante esta fase se pretende identificar y validar la arquitectura del sistema.
- Construcción: El objetivo de esta fase consiste en construir software desde un punto de vista incremental basado en las prioridades de los participantes.
- Transición: En esta fase se valida y despliega el sistema en el entorno de producción.

De las 4 fases que propone AUP (Inicio, Elaboración, Construcción, Transición) se decide para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando el objetivo de la misma, se unifican las restantes 3 fases de AUP en una sola, a la que llamaremos Ejecución y se agrega una fase de Cierre [12]. Para una mayor comprensión se muestra la siguiente tabla:

Tabla 1**:** Fases de AUP [12].

| Fases AUP | Fases Variación AUP-UCI | Objetivos de las fases (Variación AUP-UCI) |
|---|---|---|
| **Inicio** | Inicio | Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto.<br><br>En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto. |
| **Elaboración**<br><br>**Construcción**<br><br>**Transición** | Ejecución | En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura.<br><br>Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.<br><br>Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente.<br><br>Además, en la transición se capacita a los usuarios finales sobre la utilización del software. |
| | Cierre | En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto. |

AUP propone 7 disciplinas (Modelo, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyecto y Entorno), se decide para el ciclo de vida de los proyectos de la UCI tener 8 disciplinas, pero a un nivel más atómico que el definido en AUP. Los flujos de trabajos: Modelado de negocio, Requisitos y Análisis y Diseño en AUP están unidos en la disciplina Modelo, en la variación para

la UCI se consideran a cada uno de ellos disciplinas [12]. Para una mayor comprensión se muestra la siguiente tabla:

Tabla 2: Disciplinas de AUP [12].

| Disciplinas AUP | Disciplinas Variación AUP-UCI | Objetivos Disciplinas (Variación AUP-UCI) |
|---|---|---|
| **Modelo** | Modelado de Negocio (opcional) | Disciplina destinada a comprender los procesos de negocio de una organización; cómo funciona el negocio que se desea informatizar para tener garantías de que el software desarrollado va a cumplir su propósito. |
| | Requisitos | Desarrollar un modelo del sistema que se va a construir. Comprende la administración de los requisitos funcionales y no funcionales del producto. |
| | Análisis y Diseño | Los requisitos pueden ser refinados y estructurados para conseguir una comprensión más precisa de estos, y una descripción que sea fácil de mantener y ayude a la estructuración del sistema<br><br>Se modela el sistema y su forma (incluida su arquitectura) para que soporte todos los requisitos, incluyendo los requisitos no funcionales. |
| **Implementación** | Implementación | En la implementación, a partir de los resultados del Análisis y Diseño se construye el sistema. |
| **Prueba** | Prueba Interna | Se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas. Se deben desarrollar artefactos de prueba como: diseños de casos de prueba, listas de chequeo y componentes de prueba ejecutables para automatizar las pruebas. |
| | Prueba de Liberación | Pruebas diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los |

| | | |
|---|---|---|
| | | entregables de los proyectos antes de ser entregados al cliente para su aceptación. |
| | Prueba de aceptación | Es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido. |
| **Despliegue** | Despliegue (opcional) | Constituye la instalación, configuración, adecuación, puesta en marcha de soluciones informáticas y entrenamiento al personal del cliente. |

Todas las disciplinas antes definidas (desde Modelado de negocio hasta Despliegue) se desarrollan en la Fase de Ejecución, de ahí que en la misma se realicen iteraciones y se obtengan resultados incrementales, ver Ilustración 1:

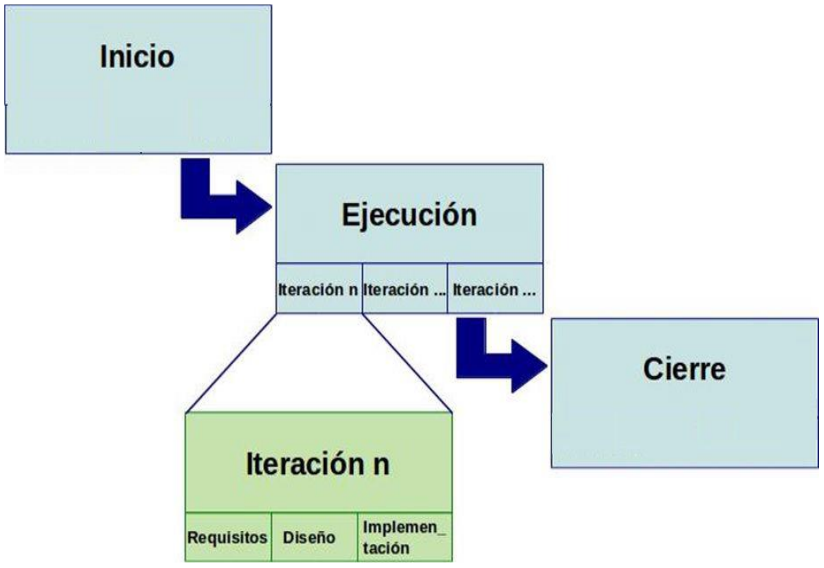



Ilustración 1: Fases e iteraciones de AUP [12]

## 1.5 Herramientas y Lenguajes para el Desarrollo del Sistema.

Desde sus inicios el Replicador de datos REKO ha sido diseñado e implementado en su totalidad usando herramientas de código abierto y librerías de clases con licencias gratuitas, lo que permite que el software pueda ser desplegado en cualquier sistema operativo. En el proceso de desarrollo del replicador se han

definido lenguajes y herramientas para alcanzar una organización ingenieril dentro del equipo de desarrollo. A continuación se describen las herramientas que se utilizarán.

Tabla 3: Herramientas y lenguajes para el desarrollo del sistema.

| **Herramienta CASE[4]** | | |
|---|---|---|
| **Nombre** | **Versión** | **Descripción** |
| **Visual Paradigm** | 8.0 | Es una herramienta UML[5] que ha sido creada para resistir el ciclo de vida completo del proceso de desarrollo del software. El mismo brinda un conjunto de ayudas, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Fue diseñado para una extensa escala de usuarios interesados en la utilización del enfoque orientado a objetos. Permite incrementar la calidad del software, la reutilización, portabilidad y estandarización de la documentación.[13] |
| **Plataforma de Desarrollo** | | |
| **Nombre** | **Versión** | **Descripción** |
| **JEE[6]** | 1.7 | Plataforma de desarrollo de aplicaciones en JAVA, reduce el coste y complejidad del desarrollo. Aplicaciones desarrolladas con JEE pueden ser desplegadas rápidamente y fácilmente mejoradas.[14] |
| **Lenguaje de Modelado** | | |
| **Nombre** | **Versión** | **Descripción** |
| **UML** | 2.0 | Lenguaje de modelado que ayuda a especificar, visualizar y documentar esquemas de sistemas de software, incluyendo su estructura y diseño. No sirve para determinar qué hacer en primer |

[4] Ingeniería de Software Asistida por Computación

[5] Por sus siglas en inglés, Unified Modeling Language, en español Lenguaje de Modelado Unificado

[6] Por sus siglas en Inglés, Java Enterprise Edition, traducidas formalmente al español como Java Empresarial.

| | | |
|---|---|---|
| | | lugar o cómo diseñar el sistema, sino que simplemente le ayuda a visualizar el diseño y a hacerlo más accesible para otros.[15] |
| **Lenguaje de Programación** | | |
| **Nombre** | **Versión** | **Descripción** |
| **JAVA** | 7.0 | Es un lenguaje orientado a objeto, de una plataforma independiente. Fue desarrollado con la idea original de usarlo para la creación de páginas WEB, tiene muchas similitudes con el lenguaje C[7] y C++[8], permite la modularidad por lo que se pueden hacer rutinas individuales que sean usadas por más de una aplicación [16]. |
| **Ambiente de Desarrollo** | | |
| **Nombre** | **Versión** | **Descripción** |
| **Eclipse** | *Kepler Service Release* 2 | Eclipse es una comunidad para individuos que quieran colaborar con software de código abierto. Este proyecto está enfocado en la construcción de una plataforma de desarrollo abierta compuesta de frameworks extensibles, herramientas y ejecutables para, construir, desplegar y gestionar software a través del ciclo de vida de este.[17] |
| **Framework** | | |
| **Nombre** | **Versión** | **Descripción** |
| **JUnit** | 4.4 | Conjunto de bibliotecas utilizadas para hacer pruebas unitarias de aplicaciones JAVA. Permite realizar la ejecución de clases JAVA de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.[18] |
| **Gestor de BD** | | |

[7] Lenguaje de programación orientado a la implementación de sistemas operativos.

[8] Lenguaje de programación creado a partir del lenguaje de programación C y con mecanismos que permiten la manipulación de objetos.

| Nombre | Versión | Descripción |
|---|---|---|
| **Microsoft SQL Server** | Microsoft SQL Server 2008 | Potente gestor de bases de datos que provee una plataforma productiva, inteligente y confiable que permite controlar las solicitudes críticas de aplicaciones, minimiza el tiempo y costos de desarrollo. Provee un motor de base de datos escalable ideal para tareas críticas.[19] |
| **Oracle** | 11g | Es un SGBD relacional desarrollado por la Corporación Oracle. Se considera uno de los sistemas de base de datos más completos, destacando su soporte a transacciones.[20] |
| **PostgreSQL** | 9.4 | Es un SGBD relacional orientada a objetos de software libre. Su desarrollo es dirigido por una comunidad de desarrolladores y organizaciones comerciales. Se destaca por su alta concurrencia.[20] |
| **MySQL** | 4.5.0.9 | Es un SGBD relacional multihilo y multiusuario. Se destaca pues funciona sobre múltiples plataformas y múltiple motores de búsqueda.[20] |

## 1.6 Conclusiones Parciales

- ✓ Se utilizará para la captura de los metadatos la interfaz *DatabaseMetaData* y consultas directas al gestor de BD pues la interfaz *DatabaseMetaData* no permite obtener toda la estructura de las bases de datos.
- ✓ Se utilizarán las mismas herramientas, lenguajes y metodología que actualmente tiene definido el Replicador de datos REKO, recreando un ambiente propicio para el desarrollo de la solución.

# Capítulo #2. Características y diseño del sistema

En este capítulo se describirá la propuesta de solución del mecanismo para la captura de metadatos mediante el análisis y diseño del sistema, tomando como referencia los conceptos y términos definidos en el primer capítulo, y regido por la metodología AUP.

## 2.1 Descripción del Proceso a Automatizar

El Replicador de Datos REKO funciona en un entorno multi-maestro con una instancia en cada nodo, donde cada nodo puede realizar modificaciones en la estructura de la BD a la que está conectada. Actualmente el software posee un mecanismo que permite la captura de información de tablas y columnas de la BD para todos los SGBD a los que el Replicador de datos REKO brinda soporte: PostgreSQL, MySQL, Microsoft SQL Server y Oracle.

Al iniciar la aplicación, el mecanismo actual no es capaz de capturar toda la información necesaria para el proceso de réplica de estructura, provocando incoherencias entre las bases de datos y conflictos tanto en la réplica de datos como en la réplica de estructura. Para resolver los problemas surgidos se propone el desarrollo de un mecanismo de captura de metadatos, que proporcione la información necesaria de la BD para lograr realizar el proceso de réplica de estructura.

Para analizar y obtener la información de la estructura de la BD deberá ser utilizada la API *DatabaseMetaData* y los dialectos definidos en el Replicador de datos REKO. Se hará uso de la API para adquirir la información que esta aprovisiona, a través de las funcionalidades obtenidas. Mientras que, para adquirir el resto información se hará uso de los dialectos, en los cuales se construirán las sintaxis de las consultas SQL que luego serán ejecutadas en la BD.

Primeramente se deberá conocer a qué SGBD se conectará el software, para determinar que consultas serán ejecutadas en la DB, en el caso de que deban utilizarse los dialectos. Luego se creará una estructura que estará compuesta inicialmente por los esquemas extraídos de la DB. Por cada Esquema se obtendrán las Funciones, Vistas, Secuencias y Tablas, para luego adicionar estos elementos a la estructura Esquema. Además por cada Tabla se capturarán Columnas, Índices, Reglas, Disparadores y Restricciones, posteriormente se añadirán a la estructura Tabla. Cada estructura deberá contener además sus atributos específicos, definidos a partir de las características del SGBD al cual el software está conectado.

## 2.2 Modelo Conceptual

Con la construcción del modelo conceptual se logra una mejor comprensión del dominio del problema; el modelo conceptual no es más que una representación visual de los conceptos u objetos del mundo real que son significativos para el problema o el área que se analiza; representando las clases conceptuales, no los componentes de software. Puede verse como un modelo que comunica a los interesados, cuáles son los términos importantes y cómo se relacionan entre sí.[21]

En la ilustración 2 se muestran los principales conceptos relacionados con la problemática planteada. El modelo conceptual se debe analizar desde arriba hacia abajo, comenzando por el concepto Capturador_Metadatos.

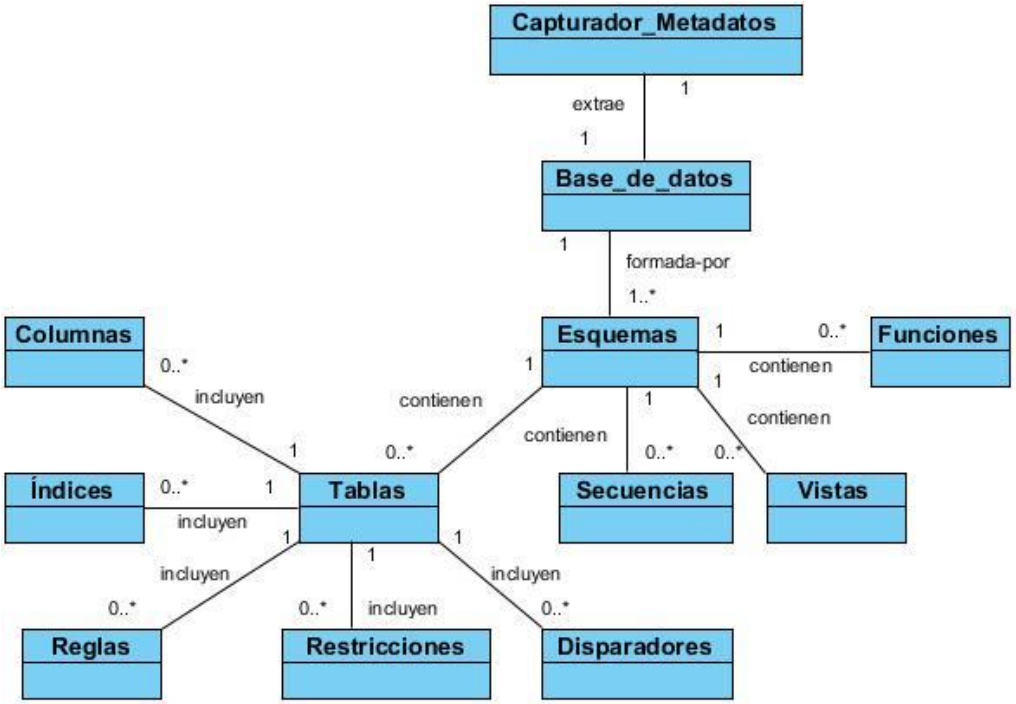



Ilustración 2: Modelo conceptual

- ✓ **Capturador_Metadatos:** concepto relacionado con la forma de capturar los metadatos de la BD.
- ✓ **Base_de_ datos:** concepto relacionado con la BD a la cual se le capturará los metadatos que está formada por esquemas.
- ✓ **Esquemas:** concepto relacionado con la estructura de una BD, contiene las funciones, vistas, secuencias y tablas de la BD.
- ✓ **Funciones:** concepto relacionado con la estructura de una función de la BD.

- ✓ **Secuencias:** concepto relacionado con la estructura de una secuencia de la BD.
- ✓ **Vistas:** concepto relacionado con la estructura de una vista de la BD.
- ✓ **Tablas:** concepto relacionado con la estructura de una tabla de la BD que incluye índices, columnas, disparadores, restricciones y reglas.
- ✓ **Índices:** concepto relacionado con la estructura de un índice de la tabla.
- ✓ **Columnas:** concepto relacionado con la estructura de una columna de la tabla.
- ✓ **Restricciones:** concepto relacionado con la estructura de una restricción de la tabla.
- ✓ **Disparadores:** concepto relacionado con la estructura de un disparador de la tabla.
- ✓ **Reglas:** concepto relacionado con la estructura de una regla de la tabla.

## 2.3 Modelo del Sistema

Un modelo del sistema describe lo que supuestamente hará un sistema, pero no cómo implementar dicho sistema. Idealmente, una representación de un sistema, debería mantener toda la información sobre la entidad que se está representando.[22]

### 2.3.1 Requisitos Funcionales

Los requisitos funcionales de un sistema describen lo que el sistema debe hacer. Estos requerimientos dependen del tipo de software que se desarrolle, de los posibles usuarios del software y del enfoque general tomado por la organización al redactar requerimientos.[23]

- ✓ **RF1:** Capturar metadatos de función.
- ✓ **RF2:** Capturar metadatos de vista.
- ✓ **RF3:** Capturar metadatos de secuencia.
- ✓ **RF4:** Capturar metadatos de tabla.
  - **RF4.1:** Capturar metadatos de columna.
  - **RF4.2:** Capturar metadatos de índice.
  - **RF4.3:** Capturar metadatos de disparadores.
  - **RF4.4:** Capturar metadatos de regla.
  - **RF4.5:** Capturar metadatos de restricciones.

### 2.3.2 Especificación de requisitos.

Tabla 4: Especificación de requisitos.

| No | Nombre | Descripción | Prioridad | Complejidad |
|---|---|---|---|---|
| **RF1** | Capturar metadatos de función. | El sistema cargará automáticamente la información respecto a las funciones de la BD para todos los SGBD que soporta el Replicador de datos REKO. | Alta | Alta |
| **RF2** | Capturar metadatos de vista. | El sistema cargará automáticamente la información respecto a las vistas de la BD para todos los SGBD que soporta el Replicador de datos REKO. | Alta | Alta |
| **RF3** | Capturar metadatos de secuencia. | El sistema cargará automáticamente la información respecto a las secuencias de la BD para todos los SGBD que soporta el Replicador de datos REKO. | Alta | Alta |
| **RF4** | Capturar metadatos de tabla. | El sistema cargará automáticamente la información respecto a las tablas de la BD para todos los SGBD que soporta el Replicador de datos REKO. | Alta | Alta |
| **RF4.1** | Capturar metadatos de columna. | El sistema cargará automáticamente la información respecto a las columnas de la BD para todos los SGBD que soporta el Replicador de datos REKO. | Alta | Alta |

| | | | | |
|---|---|---|---|---|
| **RF4.2** | Capturar metadatos de índice. | El sistema cargará automáticamente la información respecto a los índices de la BD para todos los SGBD que soporta el Replicador de datos REKO. | Alta | Alta |
| **RF4.3** | Capturar metadatos de disparadores. | El sistema cargará automáticamente la información respecto a los disparadores de la BD para todos los SGBD que soporta el Replicador de datos REKO. | Alta | Alta |
| **RF4.4** | Capturar metadatos de regla. | El sistema cargará automáticamente la información respecto a las reglas de la BD para todos los SGBD que soporta el Replicador de datos REKO. | Alta | Alta |
| **RF4.5** | Capturar metadatos de restricciones. | El sistema cargará automáticamente la información respecto a las restricciones, que pueden ser del tipo *check, primary key, exclusion, foreing key* y *unique,* de la BD para todos los SGBD que soporta el Replicador de datos REKO. | Alta | Alta |

### 2.3.3 Requisitos no Funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe cumplir, entiéndase estas propiedades como las características que hacen al producto atractivo, usable, rápido y confiable. [22]

Tabla 5: Requisitos no funcionales [24].

| Requisito no funcional | Descripción |
|---|---|
| ✓ **Fiabilidad** | Grado en que un sistema, producto o |

| | |
|---|---|
| | componente realiza funciones especificadas en las condiciones especificadas por un período de tiempo determinado.<br>**Sub-Atributos de calidad**<br>- Tolerancia a fallos: Grado en que un sistema, producto o componente esté funcionando como es a pesar de la presencia de fallos de hardware o software.<br>- Recuperabilidad: Grado en el cual, en caso de una interrupción o un fracaso, un producto o sistema puede recuperar los datos directamente afectados y restablecer el estado deseado del sistema.<br>- Madurez: Grado en que un sistema, producto o componente deberá cumplir las necesidades de fiabilidad en el funcionamiento normal.<br>- Disponibilidad: Grado en que un sistema, producto o componentes está operativo y accesible cuando se requiere para su uso. |
| ✓ **Mantenibilidad** | Grado de eficacia y eficiencia con la que un producto o sistema pueden ser modificados por los mantenedores destinados.<br>**Sub-Atributos de calidad**<br>- Analizabilidad: Grado de eficacia y eficiencia con la que es posible evaluar el impacto sobre un producto o sistema de un cambio previsto a uno o más de sus partes, o para diagnosticar un producto para deficiencias o causas de los fallos, o para identificar las partes que se desea modificar. |

| | |
|---|---|
| | - Modificabilidad: Grado en que un producto o sistema pueden modificarse de manera efectiva y eficiente sin introducir defectos o degradar la calidad del producto existente.<br>- Capacidad para ser probado: Grado de eficacia y eficiencia con la que los criterios de prueba se pueden establecer para un sistema, equipo o componente y las pruebas se pueden realizar para determinar si se han cumplido los criterios. |
| ✓ **Eficiencia en el rendimiento** | Desempeño en relación con la cantidad de recursos utilizados bajo las condiciones establecidas.<br>**Sub-Atributos de calidad**<br>- Comportamiento en el tiempo: Grado en que los tiempos de respuesta y procesamiento y las tasas de rendimiento de un producto o sistema, al realizar sus funciones, cumplir con los requisitos.<br>- Utilización de recursos: Grado en el que las cantidades y tipos de recursos utilizados por un producto o sistema, al realizar sus funciones, cumplir con los requisitos. |

Para ver el escenario donde se ve implícito cada requisito no funcional en la solución consultar Anexo 1.

### 2.3.4 Descripción de Historias de Usuario.

Entre los artefactos que define la metodología seleccionada se encuentran las historias de usuario. Las historias de usuario son utilizadas en las metodologías de desarrollo ágiles para la especificación de requisitos, son una forma rápida de administrar los requisitos de los usuarios

sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos, también permiten responder rápidamente a los requisitos cambiantes.[25]

Tabla 6: Historia de usuario 1.

| **HU:1** | **Nombre del requisito: Capturar metadatos de función.** |
|---|---|
| **Programador:** Javier J. Marín Morales | **Iteración Asignada:** 1 |
| **Prioridad:** Alta | **Tiempo Estimado:** 15 días |
| **Riesgo en Desarrollo:**<br>✓ Rotura de alguna estación de trabajo perteneciente al proyecto.<br>✓ Problemas eléctricos. | **Tiempo Real:** 15 días |
| **Descripción:** El sistema cargará automáticamente la información que describe la estructura de las funciones de la BD.<br>El mecanismo extraerá los metadatos correspondiente a las funciones de la BD, mediante consultas al esquema *INFORMATION _SCHEMA* de cada gestor de BD y el método *getProcedureColumns* de la interfaz de programación de aplicaciones *DatabaseMetadata.* La información capturada se almacenará en la clase *Function* del paquete *dbmetadata*. | |

Tabla 7: Historia de usuario 2.

| **HU:2** | **Nombre del requisito: Capturar metadatos de vista.** |
|---|---|
| **Programador:** Javier J. Marín Morales | **Iteración Asignada:** 1 |
| **Prioridad:** Alta | **Tiempo Estimado:** 7 días |
| **Riesgo en Desarrollo:**<br>✓ Rotura de alguna estación de trabajo perteneciente al proyecto.<br>✓ Problemas eléctricos. | **Tiempo Real:** 7 días |
| **Descripción:** El sistema cargará automáticamente la información que describe la estructura de las vistas de la BD. | |

| |
|---|
| El mecanismo extraerá los metadatos correspondiente a las vistas de la BD, mediante consultas al esquema *INFORMATION _SCHEMA* de cada gestor de BD y el método *getTables* de la interfaz de programación de aplicaciones *DatabaseMetadata.* La información capturada se almacenará en la clase *view* del paquete *dbmetadata*. |

Tabla 8: Historia de usuario 3.

| **HU:3** | **Nombre del requisito: Capturar metadatos de secuencia.** |
|---|---|
| **Programador:** Javier J. Marín Morales | **Iteración Asignada:** 1 |
| **Prioridad:** Alta | **Tiempo Estimado:** 7 días |
| **Riesgo en Desarrollo:**<br>✓ Rotura de alguna estación de trabajo perteneciente al proyecto.<br>✓ Problemas eléctricos. | **Tiempo Real:** 7 días |
| **Descripción:** El sistema cargará automáticamente la información que describe la estructura de las secuencias de la BD.<br>El mecanismo extraerá los metadatos correspondiente a las secuencias de la BD, mediante consultas al esquema *INFORMATION _SCHEMA* de cada gestor de BD. La información capturada se almacenará en la clase *sequence* del paquete *dbmetadata*. | |

Tabla 9: Historia de usuario 4.

| **HU:4** | **Nombre del requisito: Capturar metadatos de tabla.** |
|---|---|
| **Programador:** Javier J. Marín Morales | **Iteración Asignada:** 1 |
| **Prioridad:** Alta | **Tiempo Estimado:** 20 días |
| **Riesgo en Desarrollo:**<br>✓ Rotura de alguna estación de trabajo perteneciente al proyecto.<br>✓ Problemas eléctricos. | **Tiempo Real:** 20 días |
| **Descripción:** El sistema cargará automáticamente la información que describe la estructura de las tablas | |

de la BD.

El mecanismo extraerá los metadatos correspondiente a las tablas, índices, restricciones, reglas, disparadores y columnas de la BD, mediante consultas al esquema *INFORMATION _SCHEMA* de cada gestor de BD y los métodos de la interfaz de programación de aplicaciones *DatabaseMetadata.* La información capturada se almacenará en las clases *table, column, trigger, index, rule y constraint* del paquete *dbmetadata*.

## 2.4 Descripción de la Arquitectura del Replicador de datos REKO

Una arquitectura de software define la estructura del sistema. Esta estructura se constituye de componentes -módulos o piezas de código- que nacen de la noción de abstracción, cumpliendo funciones específicas, e interactuando entre sí con un comportamiento definido [26].

### 2.4.1 Estilo Arquitectónico

Los estilos arquitectónicos son una abstracción de tipos de elementos y aspectos formales a partir de diversas arquitecturas específicas. Un estilo arquitectónico encapsula decisiones esenciales sobre los elementos arquitectónicos y enfatiza restricciones importantes de los elementos y sus relaciones posibles.

Una arquitectura basada en componentes describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. Esta arquitectura se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas. Esto provee un nivel de abstracción mayor que los principios de orientación a objetos y no se enfoca en asuntos específicos de los objetos como los protocolos de comunicación y la forma como se comparte el estado. El estilo de arquitectura basado en componentes se caracteriza por ser un estilo de diseño para aplicaciones compuestas por componentes individuales. Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas y porque define una aproximación de diseño que usa componentes, los que se comunican a través de interfaces que contienen métodos, eventos y propiedades.[27]

La Arquitectura Basada en Componentes posibilita alcanzar un mayor nivel de reutilización de software. Permite que las pruebas sean ejecutadas de manera individual a cada uno de los componentes antes de probar el sistema integrado. Cuando existe un bajo acoplamiento entre componentes, el desarrollador es libre de actualizar o agregar componentes según sea necesario, sin afectar otras partes del sistema. Dado que un componente puede ser construido y luego

mejorado continuamente, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.[27]

El Replicador de datos REKO presenta una arquitectura que puede definirse como basada en componentes, debido a que todas sus partes encapsulan un conjunto de comportamientos que pueden ser reemplazados por otros. Los principales componentes presentes en el software son:

- ✓ ***Capturador_cambios*:** encargado de capturar los cambios que se realizan en la base de datos y entregarlos al distribuidor.
- ✓ ***Distribuidor:*** determina el destino de cada cambio realizado en la BD, los envía y se responsabiliza de su llegada.
- ✓ ***Administrador:*** permite realizar las configuraciones principales del software como el registro de nodos, configuración de las tablas a replicar y el monitoreo del funcionamiento.
- ✓ ***Aplicador:*** ejecuta en la BD los cambios que son enviados hacia él desde otro nodo de réplica.

Para realizar la captura de los metadatos de una BD es necesario hacer modificaciones en el paquete *dbmetadata* y en el paquete *trigger* correspondientes al componente Capturador_cambios, debido a que los elementos que no son capturados con la API *DatabaseMetaData* deben ser pedidos directamente al gestor donde se encuentre la BD, donde al iniciar el sistema este debe cargar todos los metadatos correspondientes a los elementos que conforman una BD independientemente del gestor de BD que contenga la BD dígase tablas, columnas, disparadores, índices, restricciones, reglas, vistas, secuencias y funciones.

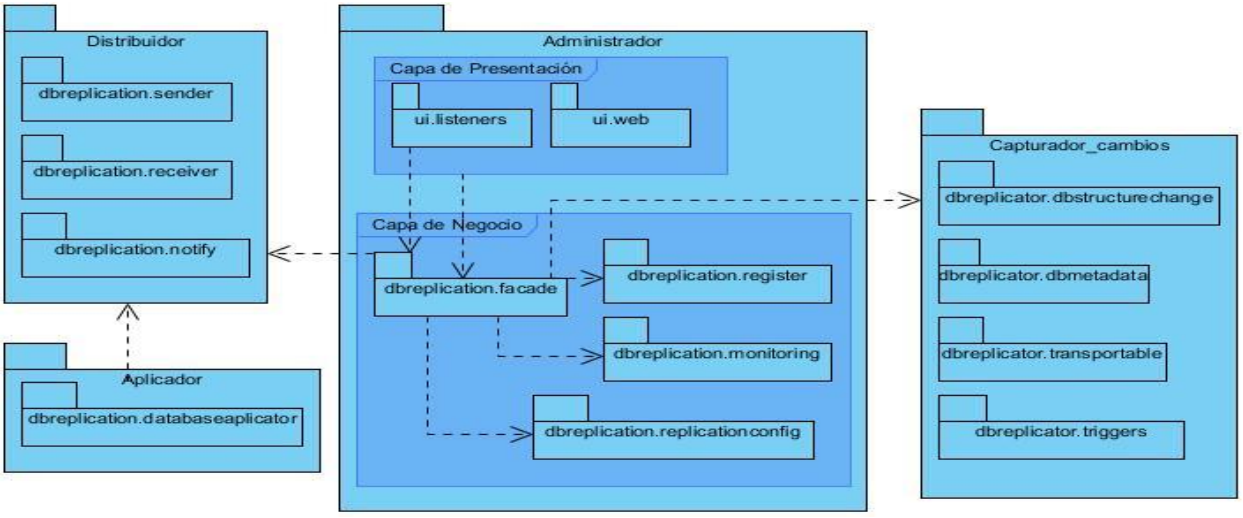



Ilustración 3: Vista de los principales componentes del Replicador de datos REKO.[28]

### 2.4.2 Patrones de diseño

Dentro del marco de la ingeniería de software se puede definir un patrón de diseño como una solución reutilizable a un problema que ocurre frecuentemente dentro de un contexto en el diseño de software. Un patrón de diseño no es un diseño final que se puede incluir directamente en el código sino que se trata de un modelo o descripción para la forma de resolver un problema que puede ser utilizado en otras situaciones distintas. [29]

**Patrones Grasp:** Los Patrones Generales de Software para Asignar Responsabilidades (GRASP, por sus siglas en inglés) describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable. [30]

- ✓ **Experto:** asigna la responsabilidad de crear objetos e implementar métodos a la clase que contiene la información necesaria para realizar estas acciones. En este caso la clase experto sería **MetadataManager** debido a que la misma contiene la información necesaria para realizar acciones con las demás clases.
- ✓ **Creador:** permite identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. Este patrón fue utilizado para el diseño de la clase **MetadataManager** debido a que la misma es la encargada de crear la estructura de los diferentes componentes de la BD.
- ✓ **Alta cohesión**: asigna responsabilidades de manera que la información que almacena una clase sea coherente y esté relacionada con la clase [31]. Cada clase diseñada para el desarrollo del mecanismo contiene sólo la información relacionada con las acciones que requiera ejecutar.
- ✓ **Bajo acoplamiento:** permite diseñar con el objetivo de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases [31]. Cada clase diseñada para el mecanismo contiene las funcionalidades necesarias para evitar, en caso de modificar una de ellas, tener que modificar la menor cantidad posible.

## 2.5 Modelo del diseño

La fase de diseño expande y detalla los modelos de análisis tomando en cuenta todas las implicaciones y restricciones técnicas. El propósito del diseño es especificar una solución que trabaje y pueda ser fácilmente convertida en código fuente y construir una arquitectura simple y fácilmente extensible [32].

### 2.5.1 Diagrama de Paquetes

Un paquete es un mecanismo utilizado para agrupar elementos de UML. Permite organizar los elementos modelados con UML, facilitando de ésta forma el manejo de los modelos de un sistema complejo. [33]

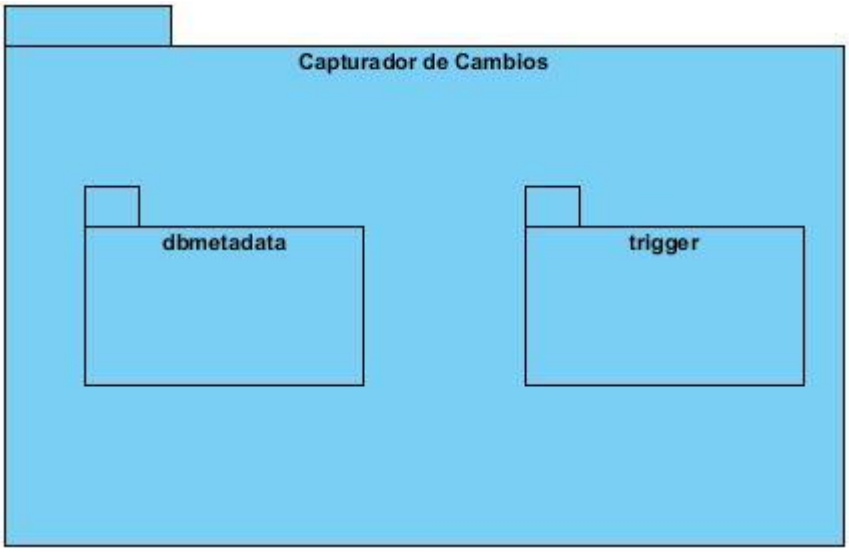



Ilustración 4: Diagrama de paquete general.

### 2.5.2 Diagrama de clases

Describen el objeto y las estructuras de información que se utilizan en la aplicación, tanto de forma interna como en la comunicación con los usuarios. Esta información se describe sin hacer referencia a ninguna implementación concreta [34].

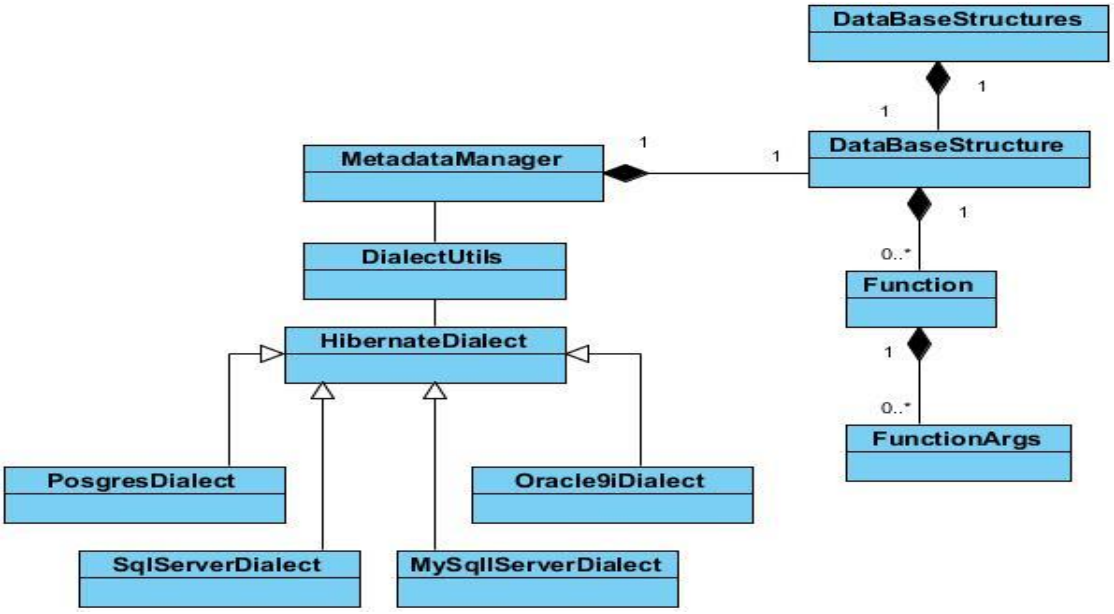



Ilustración 5: Diagrama de clases de la HU1.

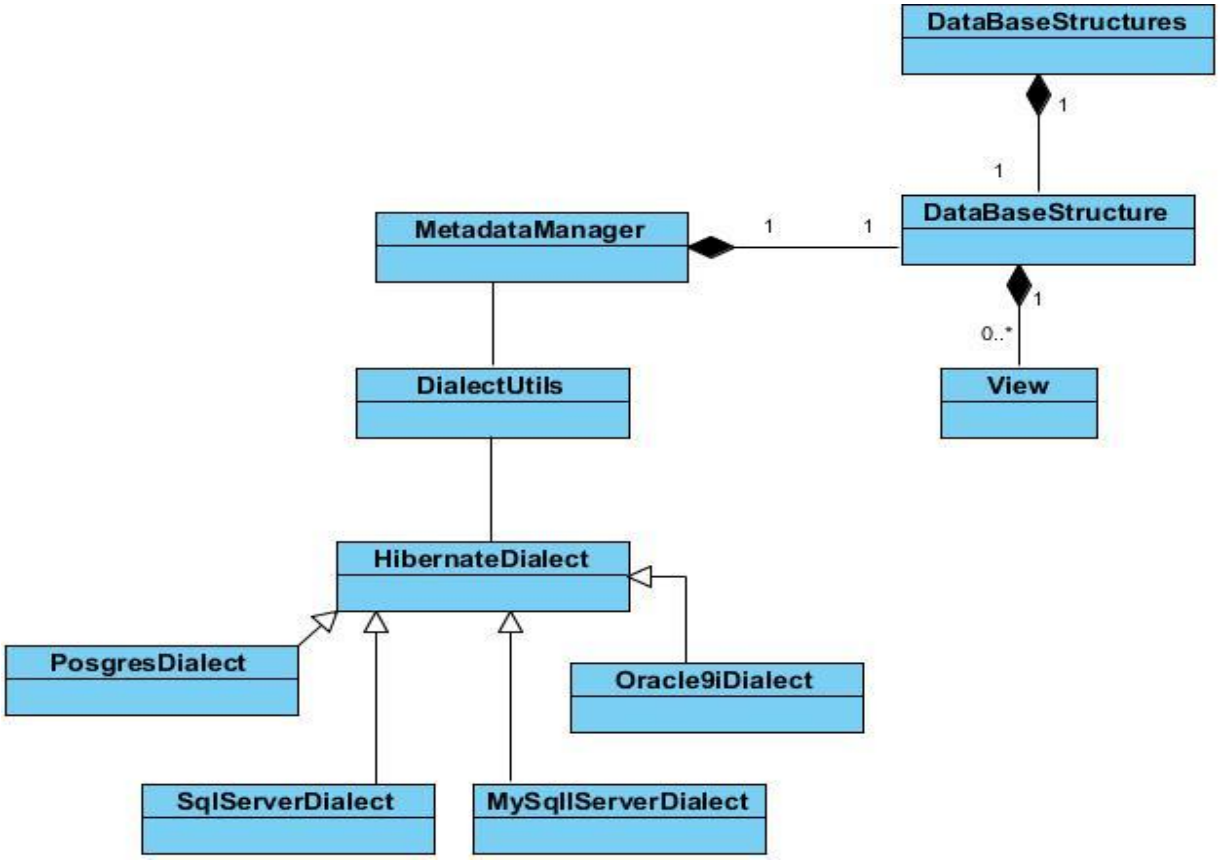



Ilustración 6: Diagrama de clase HU2

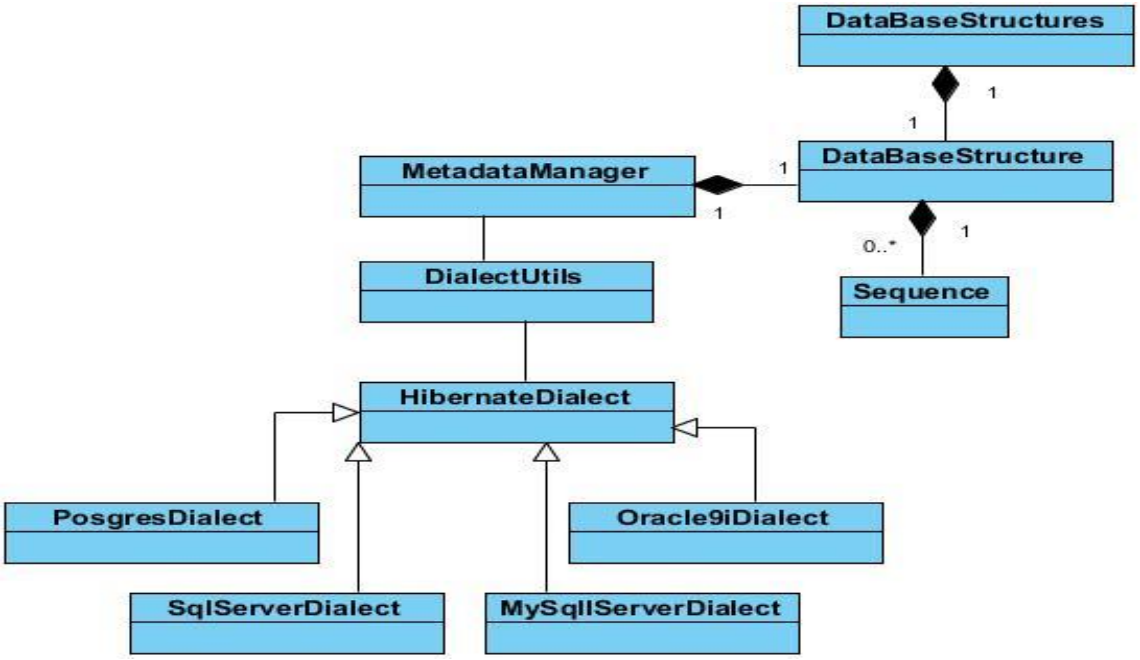



Ilustración 7: Diagrama de clase HU3

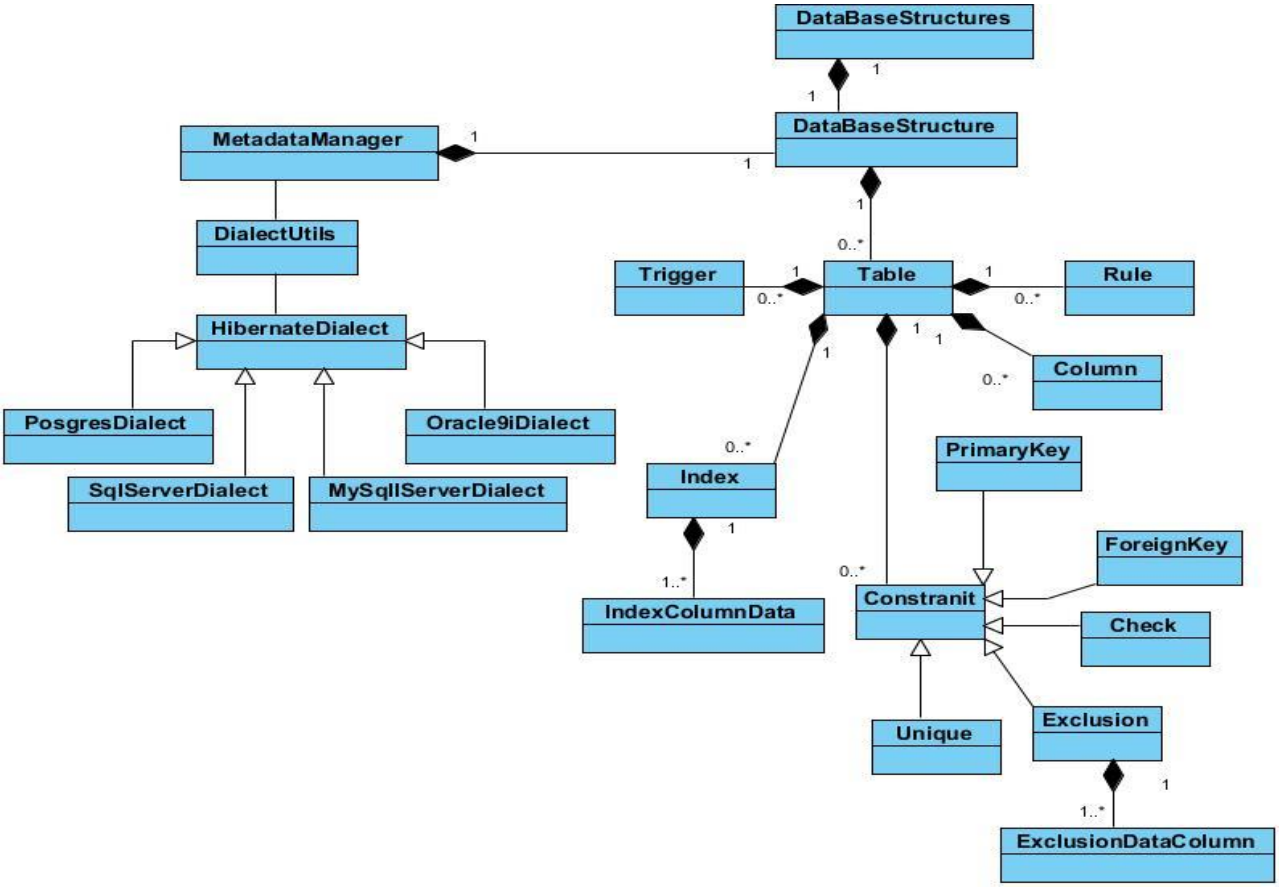



Ilustración 8: Diagrama de clase HU4

**Descripción de las clases.**

Tabla 10: Descripción de la clase MetadataManager.

| **Descripción de la clase MetadataManager** | |
|---|---|
| **Nombre:** MetadataManager | |
| **Tipo de clase:** Controladora | |
| **Atributo** | **Tipo** |
| metadataCache | private |
| dataSource | private |
| currentConnection | private |
| structures | private |
| dialect | private |
| **Responsabilidades** | |
| **Nombre:** setDataSource(dataSource: DataSource) | |
| **Descripción:** establece el datasource a partir del cual se obtiene la conexión a la BD. | |
| **Nombre:** getStructure(squema: String): DataBaseStructure | |
| **Descripción:** devuelve la estructura del esquema dado. | |
| **Nombre:** getStructure():DataBaseStructure | |
| **Descripción:** devuelve la estructura del esquema public. | |
| **Nombre:** getDataBaseStructures():DataBaseStructure | |
| **Descripción:** devuelve todas las estructuras obtenidas hasta el momento. | |
| **Nombre:** getMetadata():DatabaseMetaData | |
| **Descripción:** devuelve un objeto de tipo DatabaseMetadata. | |
| **Nombre:** getTableNames(schema: String): LinkedList<String> | |
| **Descripción:** devuelve una lista con los nombres de las tablas de la BD dado un esquema. | |
| **Nombre:** getViewNames(schema: String): LinkedList<String> | |
| **Descripción:** devuelve una lista con los nombres de las vistas de la BD dado un esquema | |
| **Nombre:** getSequenceNames(schema: String): LinkedList<String>. | |
| **Descripción:** devuelve una lista con los nombres de las secuencias de la BD dado un esquema. | |
| **Nombre:** getFunctionNames(schema: String): LinkedList<String>. | |

| |
|---|
| **Descripción:** devuelve una lista con los nombres de las funciones de la BD dado un esquema. |
| **Nombre:** hasTableOid(schema: String, tableName: String): Boolean. |
| **Descripción:** devuelve si una tabla tiene el atributo oid dado un esquema y el nombre de la tabla. |
| **Nombre:** getPrimaryKey(schema: String, tableName: String): PrimaryKey. |
| **Descripción:** devuelve una llave primaria dado un esquema y el nombre de una tabla. |
| **Nombre:** getForeignKeys(schema: String, tableName: String): LinkedList<ForeignKey>. |
| **Descripción:** devuelve una lista de llave foránea dado un esquema y el nombre de una tabla. |
| **Nombre:** getCheckConstraints( schema: String, tableName: String): LinkedList<Check>. |
| **Descripción:** devuelve una lista de restricciones (constraint) del tipo check dado un esquema y el nombre de una tabla. |
| **Nombre:** getUniqueConstraints(schema: String, tableName: String): LinkedList<Unique>. |
| **Descripción:** devuelve una lista de restricciones (constraint) del tipo unique dado un esquema y el nombre de una tabla. |
| **Nombre:** getExclusionConstraints(schema: String, tableName: String): LinkedList<Exclusion>. |
| **Descripción:** devuelve una lista de restricciones (constraint) del tipo exclusión dado un esquema y el nombre de una tabla. |
| **Nombre:** getIndex(schema: String, tableName: String, indexName: String): Index. |
| **Descripción:** devuelve un index dado un esquema, el nombre de una tabla y el nombre del index. |
| **Nombre:** getTriggers(schema: String, tableName: String): LinkedList<Trigger>. |
| **Descripción:** devuelve una lista de disparadores (trigger) dado un esquema y el nombre de una tabla. |
| **Nombre:** getRules(schema: String, tableName: String): LinkedList<Rule>. |
| **Descripción:** devuelve una lista de reglas (rule) dado un esquema y el nombre de una tabla. |
| **Nombre:** getTable(schema: String, tableName: String): Table. |
| **Descripción:** devuelve una tabla dado un esquema y el nombre de la tabla. |
| **Nombre:** getView(schema: String, viewName: String): View. |
| **Descripción:** devuelve una vista (view) dado un esquema y el nombre de la vista. |

| |
|---|
| **Nombre:** getSequence(schema: String, sequenceName: String): Sequence. |
| **Descripción:** devuelve una secuencia (sequence) dado un esquema y el nombre de la secuencia. |
| **Nombre:** getFunction(schema: String, functionName: String): Function. |
| **Descripción:** devuelve una función (function) dado un esquema y el nombre de la función. |
| **Nombre:** getSchemas():LinkedList<String>. |
| **Descripción:** devuelve una lista con los nombres de los esquemas de la BD. |

Tabla 11: Descripción de la clase DataBaseStructure.

| **Descripción de la clase DataBaseStructure** | |
|---|---|
| **Nombre:** DataBaseStructure | |
| **Tipo de clase:** Auxiliar | |
| **Atributo** | **Tipo** |
| dbName | Private |
| tableMap | Private |
| viewMap | Private |
| sequenceMap | Private |
| functionMap | Private |
| **Responsabilidades** | |
| **Nombre:** addTable(table:Table). | |
| **Descripción:** añade una tabla a la estructura. | |
| **Nombre:** addView(view:View). | |
| **Descripción:** añade una vista (view) a la estructura. | |
| **Nombre:** addFunction(function:Function). | |
| **Descripción:** añade una función (function) a la estructura. | |
| **Nombre:** addSequence(sequence:Sequence). | |
| **Descripción:** añade una secuencia (sequence) a la estructura. | |

### 2.5.3 Diagrama de Despliegue

El Diagrama de despliegue es un diagrama estructurado que muestra la arquitectura del sistema desde el punto de vista del despliegue (distribución) de los artefactos del software en los destinos de despliegue.

Los artefactos representan elementos concretos en el mundo físico que son el resultado de un proceso de desarrollo. Ejemplos de artefactos son archivos ejecutables, bibliotecas, archivos, esquemas de bases de datos, archivos de configuración. [35]

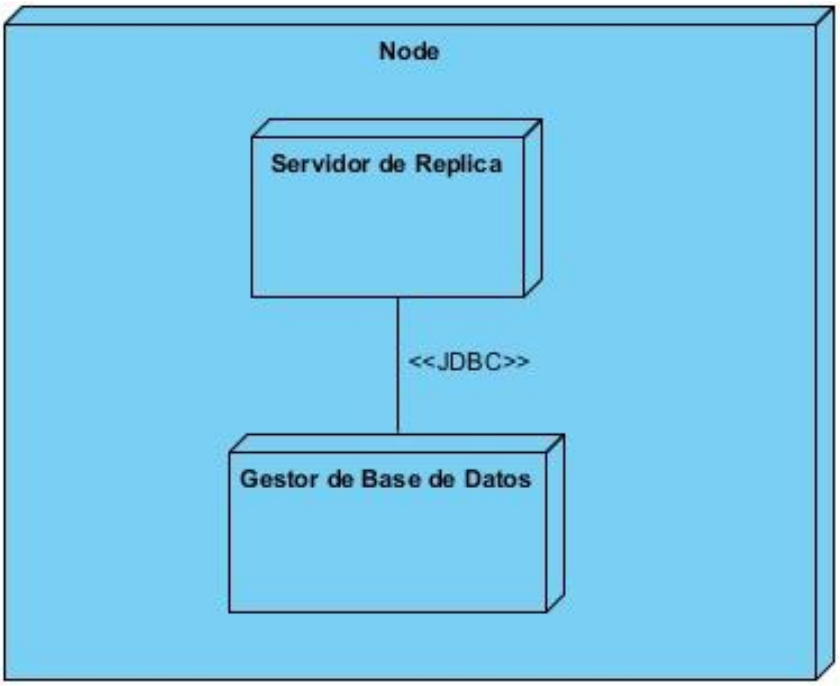



Ilustración 9: Diagrama de despliegue

## 2.6 Conclusiones Parciales

- ✓ Se describió el proceso a informatizar y se elaboró un diagrama conceptual donde se expusieron los principales conceptos que intervienen en el sistema.
- ✓ Se identificaron los requisitos tanto funcionales como no funcionales, dando paso a las historias de usuario donde se describieron cada uno de los requisitos funcionales.
- ✓ Se realizó la descripción de la arquitectura utilizada en el Replicador de datos REKO y se fundamentaron y ejemplificaron los patrones de diseño utilizados.

# Capítulo #3. Implementación y pruebas

En el presente capítulo se aborda sobre una de las fases más importante en el ciclo de vida de un software, la fase de prueba, el diseño de los casos de prueba, ejecución y análisis de los resultados obtenidos. Así como la elaboración del modelo de implementación donde se obtendrá el diagrama de componentes, además de evaluar el diseño aplicando métricas de software.

## 3.1 Modelo de implementación

El modelo de implementación es comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes se pueden encontrar datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos.[36]

### 3.1.1 Diagramas de componentes

Los diagramas de componentes ilustran las piezas del software, que conformarán un sistema. Un diagrama de componentes tiene un nivel más alto de abstracción que un diagrama de clase, usualmente un componente se implementa por una o más clases en tiempo de ejecución. Estos son bloques de construcción, como eventualmente un componente puede comprender una gran porción de un sistema [37].

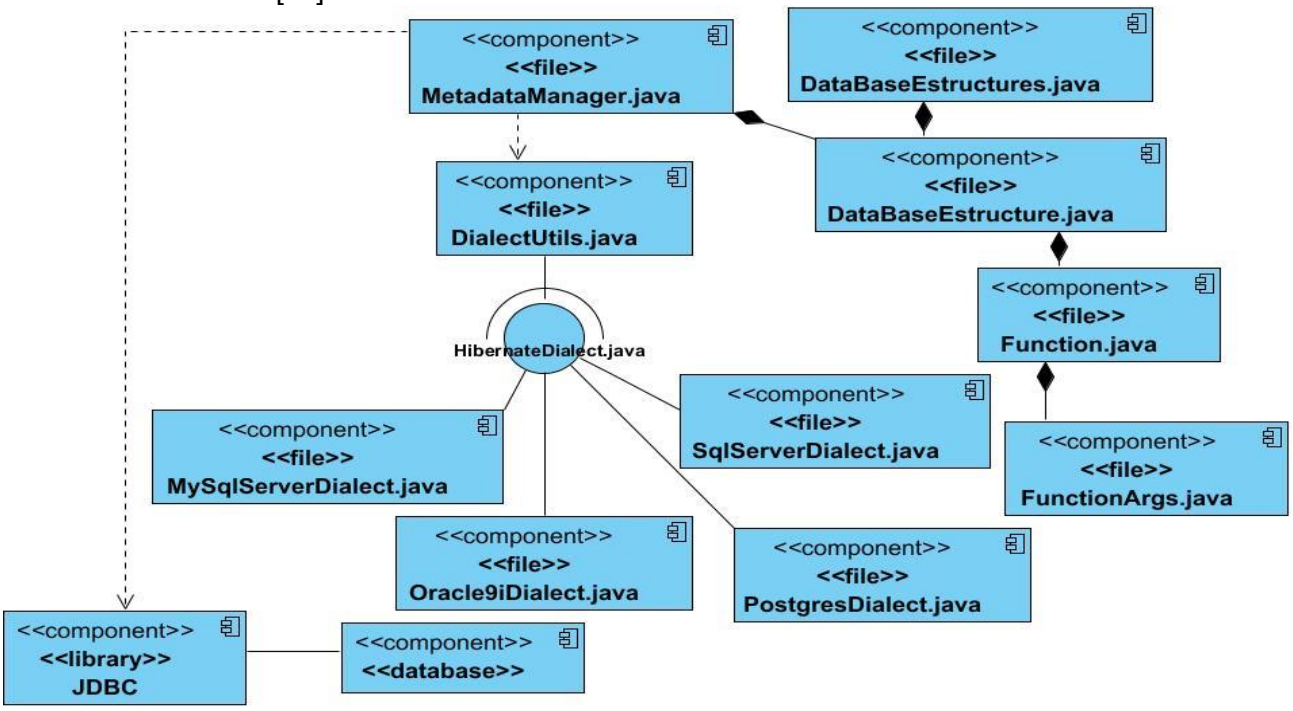



Ilustración 10: Diagrama de componente del diagrama de clase de la HU1.

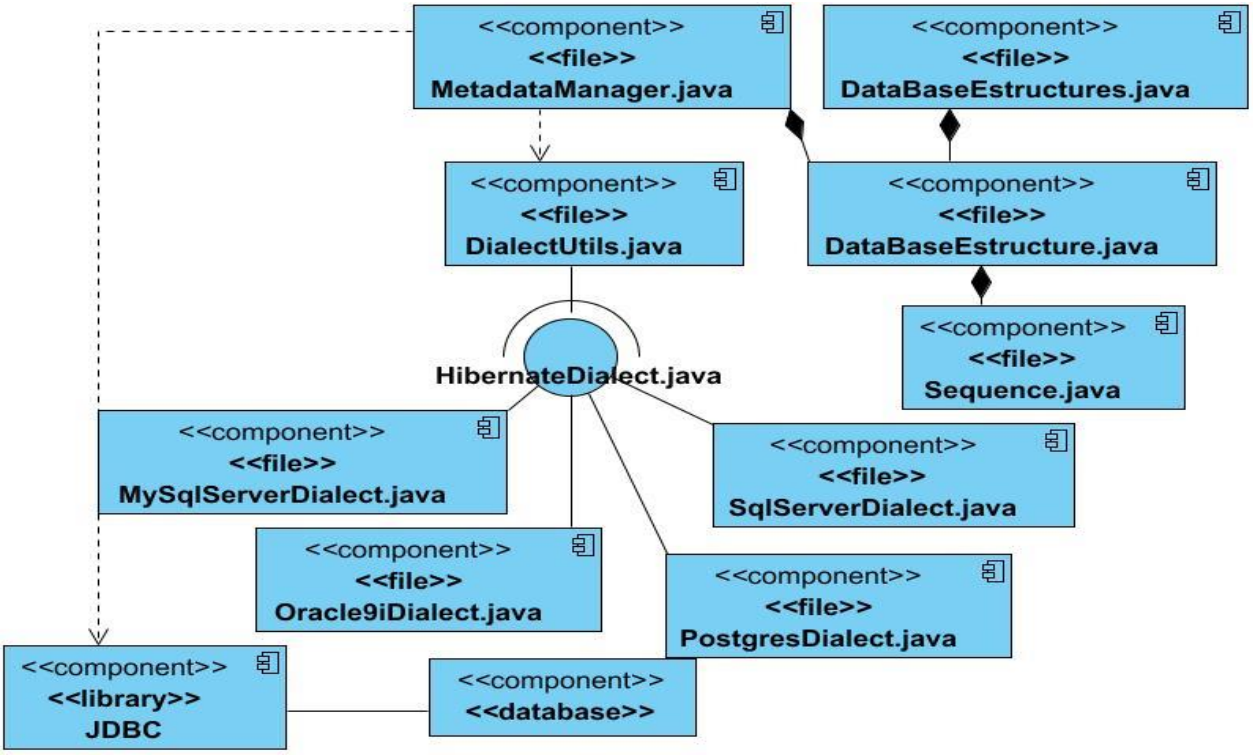



Ilustración 11: Diagrama de componente del diagrama de clase de la HU2

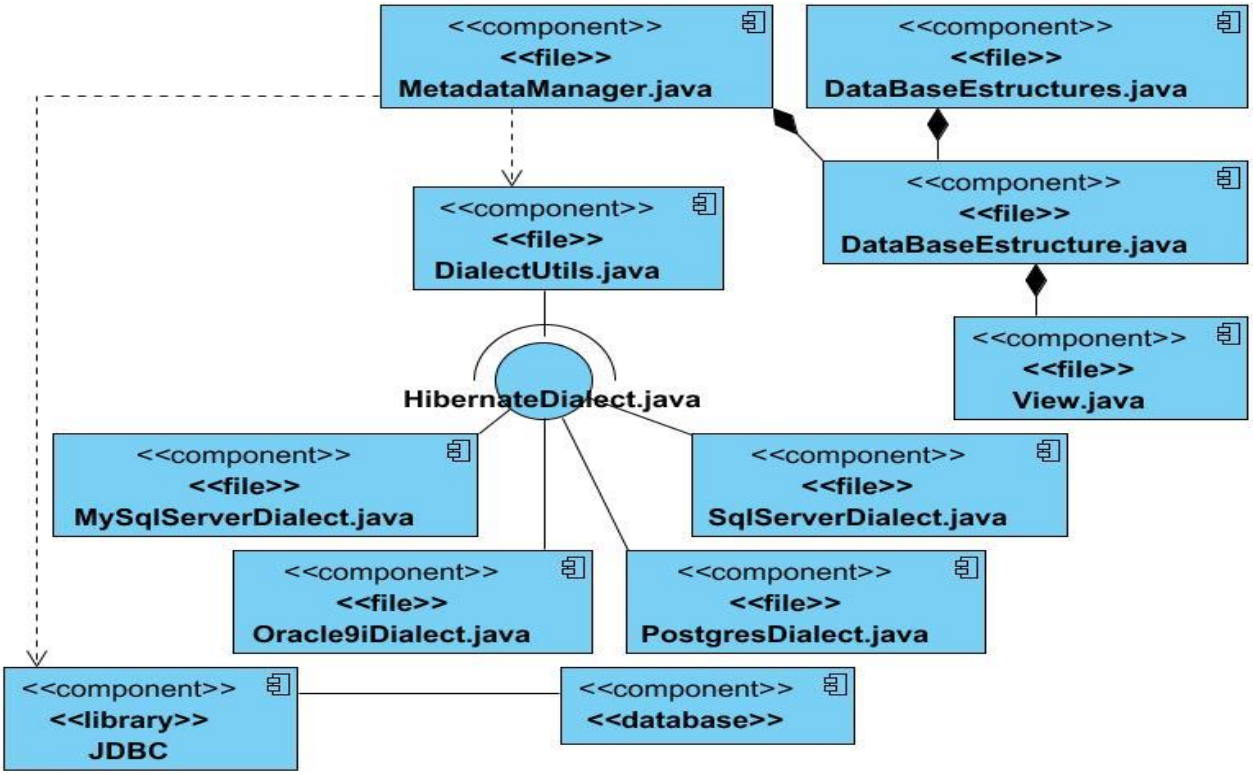



Ilustración 12: Diagrama de componente del diagrama de clase de la HU3

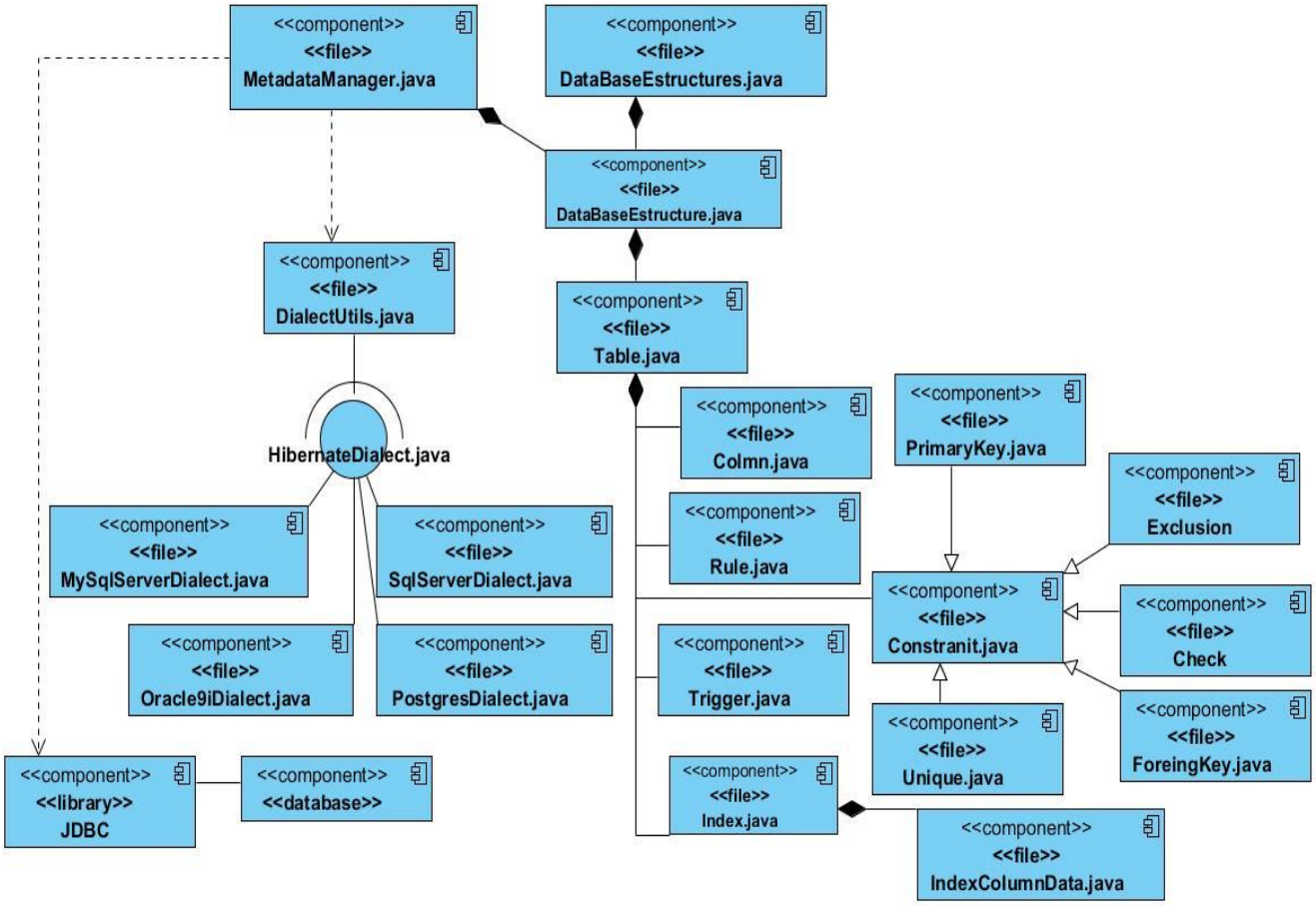



Ilustración 13: Diagrama de componente del diagrama de clase de la HU4.

### 3.1.2 Código fuente

Se define como código fuente al texto desarrollado en un lenguaje de programación y que debe ser compilado o interpretado para poder ejecutarse en un ordenador, también llamado computadora [38].

Con el objetivo de alcanzar un mayor entendimiento del código fuente se definió como estándar de codificación el utilizado en el desarrollo del proyecto Replicador de datos REKO. Entre las reglas definidas se encuentran:

- Utilizar la variante *lowerCamelCase* para los identificadores del tipo variables y métodos. Las palabras comenzarán con minúsculas y si los identificadores están compuestos por varias palabras, las siguientes empezarán con mayúscula. [39]

- Utilizar la variante *UpperCamelCase* para los identificadores del tipo clase, enumeradores e interfaces. Todas las palabras que componen a dichos identificadores empezarán con

mayúscula.[39]

A continuación se muestran fragmentos de código fuente pertenecientes a implementaciones importantes de la clase *MetadataManager*:

**Este método es el encargado de crear una primary key**

```
private PrimaryKey getPrimaryKey(String schema, String tableName)
                    throws SQLException {
              PrimaryKey primaryKey = new PrimaryKey();
              ResultSet primaryKeys = getMetadata().getPrimaryKeys(null, schema,
                          tableName); // obtiene atributos de PK
              while (primaryKeys.next()) {
              primaryKey.setName(primaryKeys.getString("PK_NAME"));// obtiene nombre de PK
              String column = primaryKeys.getString("COLUMN_NAME");// obtiene nombre de columna PK
                    primaryKey.addColumn(column);
              }

              if(dialect instanceof Oracle9iDialect){ //compruebo si dialecto es oracle
                    ResultSet primaryKeyDef = getPrimaryKeyDef(schema,
                                primaryKey.getName());// obtiene atributos de PK
                    if (primaryKeyDef.next()) {
                          String deferrable = new String("NOT DEFERRABLE");

      primaryKey.setDeferrable(primaryKeyDef.getString("DEFERRABLE").equals(deferrable));
                          String deferred=new String("IMMEDIATE");

      primaryKey.setDeferred(primaryKeyDef.getString("DEFERRED").equals(deferred));
              }}
              if (dialect instanceof PostgresDialect) {//compruebo si dialecto es oracle
                    ResultSet primaryKeyDef = getPrimaryKeyDef(schema,
                                primaryKey.getName());
                    if (primaryKeyDef.next()) {
                          primaryKey

      .setDeferrable(primaryKeyDef.getBoolean("DEFERRABLE"));
                          primaryKey.setDeferred(primaryKeyDef.getBoolean("DEFERRED"));
                          String relOptionsClass = primaryKeyDef.getString("OPTIONS");
                          if (relOptionsClass != null) {
                                relOptionsClass = relOptionsClass.replace("{fillfactor=",
                                            "");
                                relOptionsClass = relOptionsClass.replace("}", "");

      primaryKey.setFillFactor(Integer.parseInt(relOptionsClass));
                          }
                    }
              }
              return primaryKey;}
```

**Este método se encarga de construir la estructura de la BD.**

```
public DatabaseStructure getStructure(String schema) throws SQLException {
            DatabaseStructure dbStructure = new DatabaseStructure();

            String dbName = getCurrentDBName();
            dbStructure.setDbName(dbName + "." + schema);

            ===== tables =====
      LinkedList<String> tableNames = getTableNames(schema);//obtiene los nombre de las tablas
            for (String tableName : tableNames) {
                  if (dialect instanceof SqlServerDialect) {
                        switch (tableName) {
                        case "sysdiagrams":
                              continue;
                        }
                  }
                  Table table = getTable(schema, tableName);//devuelve una tabla
                  dbStructure.addTable(table);// annade la table a la estructura
            }

       //views
      LinkedList<String> viewNames = getViewNames(schema); //obtiene los nombre de las vistas
            for (String viewName : viewNames) {
                  View view = getView(schema, viewName); //devuelve una vista
                  dbStructure.addView(view); // annade la vista a la estructura
            }

            //sequences
            if (dialect instanceof PostgresDialect) {
LinkedList<String> sequenceNames = getSequenceNames(schema); //obtiene los nombre de las secuencias
                  for (String sequenceName : sequenceNames) {
            Sequence sequence = getSequence(schema, sequenceName); //devuelve una secuencia
                        dbStructure.addSequence(sequence); // annade la secuencia a la estructura
                  }
            }

            //functions
LinkedList<String> functionNames = getFunctionNames(schema); //obtiene los nombre de las funciones
                  for (String functionName : functionNames) {
            Function function = getFunction(schema, functionName); //devuelve una secuencia
                        dbStructure.addFunction(function); // annade la secuencia a la estructura
                  }

            return dbStructure;// devuelve la estructura
      }
```

## 3.2 Evaluación del diseño aplicando métricas de software

La evaluación del software a partir de métricas es una medida cuantitativa que proporciona una visión profunda de la eficacia del proceso del software. Se reúnen datos básicos y son analizados, comparados con promedios, y evaluados para determinar las mejoras en la calidad y

productividad. Las métricas son también utilizadas para señalar áreas con problemas de manera que se puedan desarrollar los remedios y mejorar el proceso del software [40].

Las métricas empleadas están diseñadas para evaluar los siguientes atributos de calidad [40]:

- **Responsabilidad:** Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- **Complejidad de implementación**: consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- **Reutilización:** consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento:** consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.
- **Complejidad del mantenimiento:** consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirectamente, pero fuertemente en los costes y la planificación del proyecto.
- **Cantidad de pruebas:** consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (unidad) del producto (componente, módulo, clase, conjunto de clases, entre otras) diseñado.

Las métricas seleccionadas para evaluar la calidad del diseño del Mecanismo de captura de metadatos para el Replicador de datos REKO fueron Tamaño Operacional de Clases (TOC) y Relaciones entre Clases (RC).

**Tamaño Operacional de Clases (TOC):** Está dado por el número de métodos u operaciones (de instancia privada y heredada) que están encapsulados dentro o por una clase. Evalúa los siguientes atributos de calidad [40]:

Tabla 12 Atributos de calidad de evalúa TOC [40].

| Atributo de Calidad | Modo en que lo afecta |
|---|---|
| Responsabilidad | Aumento del TOC provoca aumento de la responsabilidad asignada a la clase. |
| Complejidad de implementación | Aumento del TOC provoca aumento de la complejidad de implementación de la clase. |
| Reutilización | Aumento del TOC provoca disminución del grado de reutilización de la clase. |

Para la evaluación de estos atributos de calidad se definieron criterios y categorías de evaluación. Para ver estos criterios consultar Anexo 2 Criterios de evaluación de la métrica TOC.

**Relaciones entre Clases (RC):** Está dado por el número de relaciones de uso de una clase con otra. Evalúa los siguientes atributos de calidad [40]:

Tabla 13 Atributos de calidad que evalúa RC [40].

| Atributo de Calidad | Modo en que lo afecta |
|---|---|
| Acoplamiento | Aumento del RC provoca aumento del Acoplamiento de la clase. |
| Complejidad de mantenimiento | Aumento del RC provoca aumento de la complejidad del mantenimiento de la clase. |
| Reutilización | Aumento del RC provoca disminución en el grado de reutilización de la clase. |
| Cantidad de pruebas | Aumento del RC provoca aumento de la Cantidad de pruebas de unidad necesarias para probar una clase. |

Para la evaluación de dichos atributos de calidad, se definieron los criterios y categorías de evaluación. Para ver estos criterios consultar Anexo 3 Criterios de evaluación de la métrica RC.

### 3.2.1 Resultados obtenidos en la aplicación de la métrica TOC

Ver Instrumento de evaluación de la métrica TOC en el Anexo 4 Instrumento de evaluación de la métrica TOC.

La Ilustración 14 muestra la representación de los resultados obtenidos agrupados en los intervalos definidos. El gráfico refleja que la mayoría de las clases tienen de 1 a 20 procedimientos. Esto demuestra que el funcionamiento general del componente está distribuido equitativamente entre las diferentes clases.

Los resultados obtenidos luego de aplicar las métricas TOC arrojaron que el diseño propuesto tiene una calidad aceptable, teniendo en cuenta que el 54 % de las clases poseen una cantidad de procedimientos menor o igual al promedio general de 16,5, esto conlleva a que las evaluaciones sean positivas en los atributos de calidad involucrados.

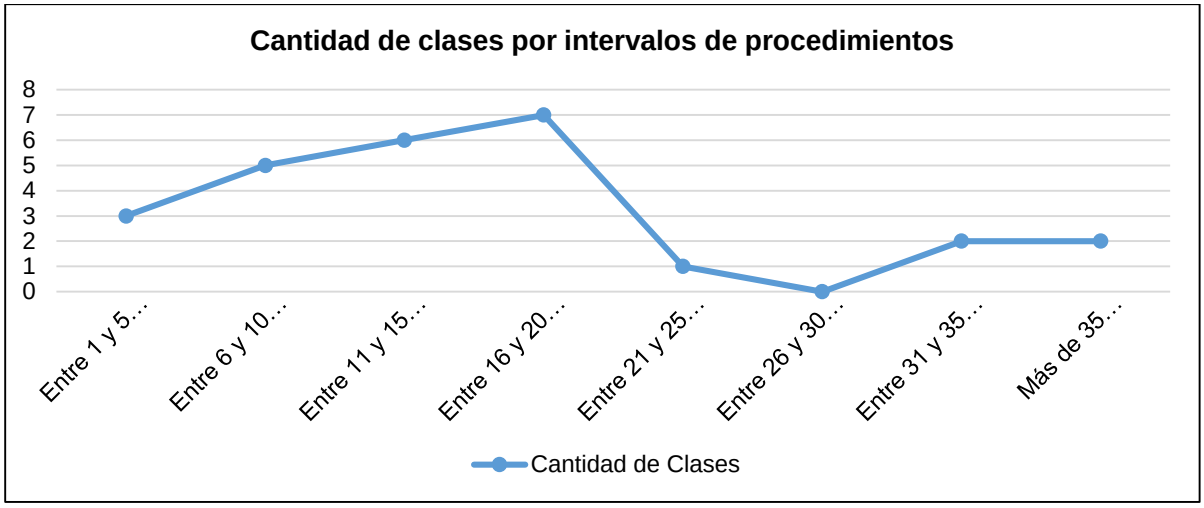



Ilustración 14 Representación de las clases según la cantidad de operaciones

La Ilustración 15 muestra la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad. Quedó demostrado que el 54% de las clases tienen una baja responsabilidad ya que este tributo se distribuyó equitativamente entre todas las clases del sistema. Esta característica permite que en caso de fallos como la responsabilidad está distribuida de forma equilibrada ningún componente sea demasiado crítico como para dejar fuera de servicio el sistema.

La Ilustración 16 muestra la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de implementación. El gráfico muestra que el 54% de las clases tienen una baja complejidad, este atributo está distribuido equitativamente. Esta característica permite mejorar el mantenimiento y soporte de las clases.

La Ilustración 17 muestra la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización. Queda demostrado que el diseño de la solución es eficiente ya que el 54 % de las clases tienen un alto grado de reutilización.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC, se puede concluir que el diseño del Mecanismo de captura de metadatos para el Replicador de datos REKO tiene una buena calidad.

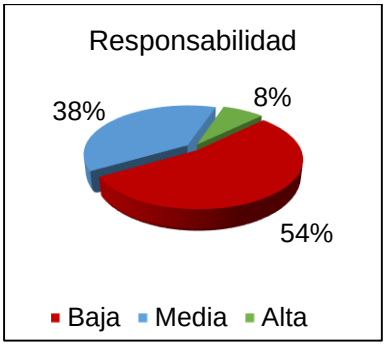



Ilustración 15: Evaluación para el atributo Responsabilidad.

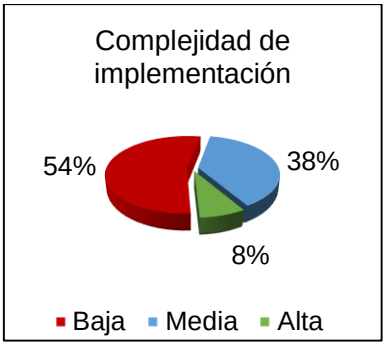



Ilustración 16: Evaluación para el atributo Complejidad de Implementación.

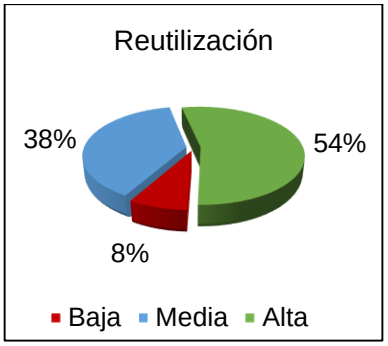



Ilustración 17: Evaluación para el atributo Reutilización.

### 3.2.2 Resultados obtenidos en la aplicación de la métrica RC

Ver Instrumento de evaluación de la métrica RC en el Anexo 5 Instrumento de evaluación de la métrica RC.

La Ilustración 18 muestra la representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos. La Ilustración 19 muestra la representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

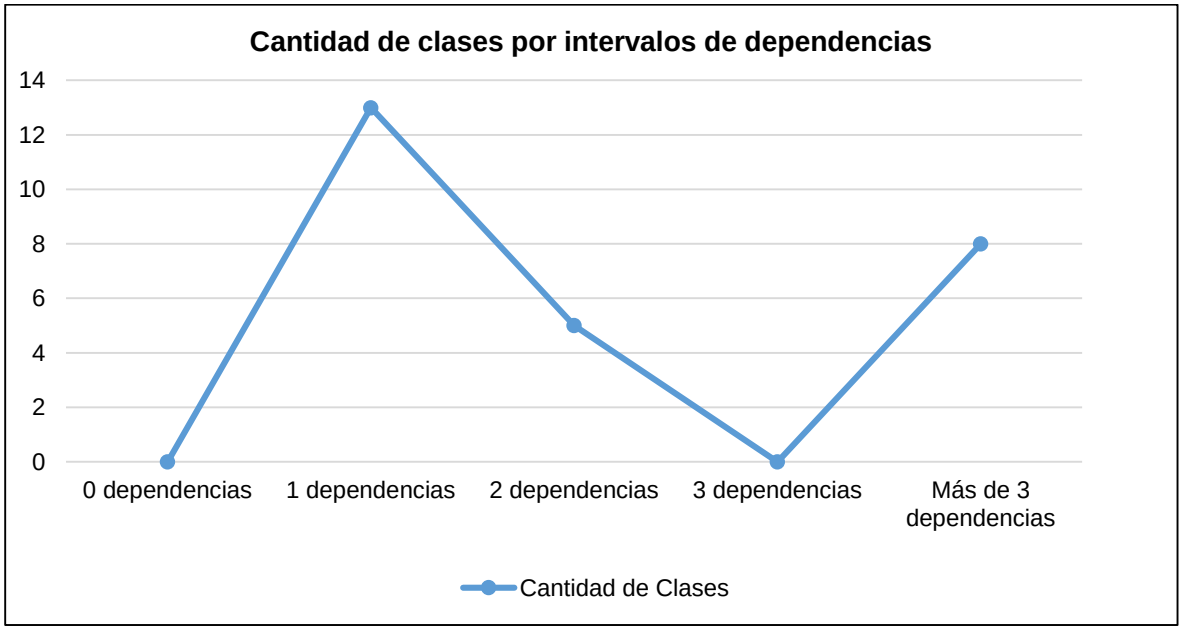



Ilustración 18 Representación de las clases según la cantidad de relaciones de uso

Los resultados obtenidos luego de aplicar las métricas RC arrojan que el diseño propuesto tiene una calidad aceptable, teniendo en cuenta que el 69 % de las clases poseen una cantidad de relaciones de uso menor o igual al promedio general de 2,4, esto conlleva a que las evaluaciones sean positivas en los atributos de calidad involucrados.

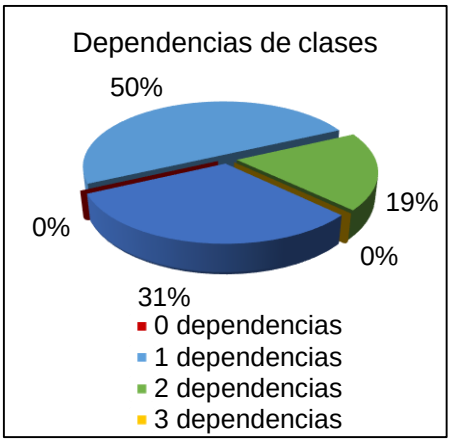



Ilustración 19 Representación en % de los resultados agrupados en los intervalos.

La Ilustración 20 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento. Se evidencia un diseño eficiente al quedar reflejado que un 50 % de las clases cuentan con un bajo acoplamiento.

La Ilustración 21 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de mantenimiento. Queda demostrada la eficiencia de la arquitectura del sistema al evidenciarse que un 70 % de las clases posee una baja complejidad de mantenimiento.

La Ilustración 22 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización. Esto evidencia que el 70% de las clases poseen una alta reutilización lo que es un factor fundamental que debe ser tenido en cuenta en el desarrollo de software.

La Ilustración 23 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de pruebas. Esto evidencia que un 70% de las clases posee baja cantidad de pruebas, lo que representa un valor favorable para el diseño realizado.

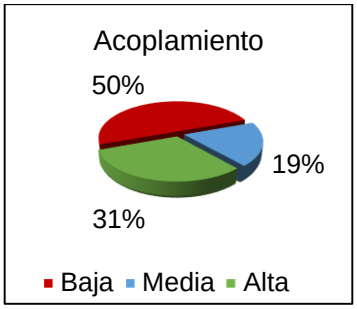



Ilustración 20: Evaluación para el atributo Acoplamiento.

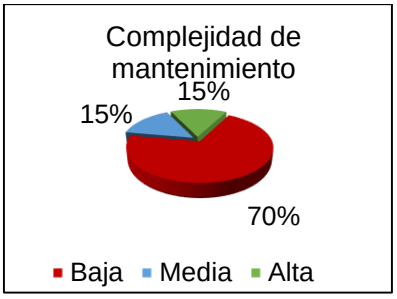



Ilustración 21: Evaluación para el atributo Complejidad de mantenimiento.

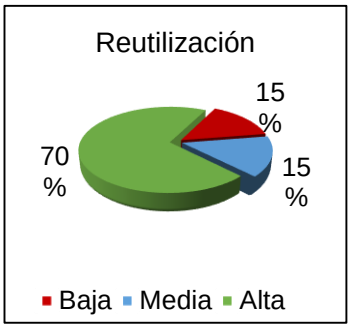



Ilustración 22: Evaluación para el atributo Reutilización.

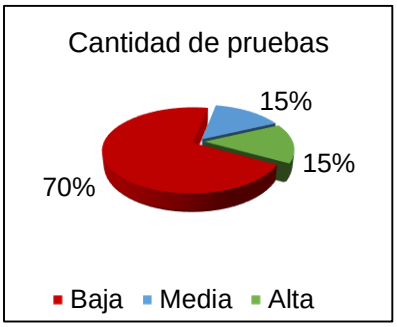



Ilustración 23: Evaluación para el atribut Cantidad de pruebas.

### 3.2.3 Matriz de inferencia de indicadores de calidad

La matriz inferencia de indicadores de calidad, también llamada matriz de cubrimiento, es una representación estructurada de los atributos de calidad y métricas utilizadas para evaluar la calidad del diseño propuesto. Dicha matriz permite conocer si los resultados obtenidos de las relaciones atributo/métrica son positivos o no, llevando estos resultados a una escalabilidad numérica [40].

Si los resultados son positivos se le asigna el valor de 1, si son negativos toma valor 0 y si no existe relación es considerada como nula y es representada con un guión simple (-). Luego se puede obtener un resultado general para cada atributo promediando todas sus relaciones no nulas. A continuación se muestran los resultados obtenidos [40].

Tabla 14 Rango de valores para la evaluación de la relación atributo/métrica.

| Atributo/Métrica | TOC | RC | Promedio |
|---|---|---|---|
| Responsabilidad | 1 | (-) | 1 |
| Complejidad de | 1 | (-) | 1 |

| | | | |
|---|---|---|---|
| implementación | | | |
| Reutilización | 1 | 1 | 1 |
| Acoplamiento | (-) | 1 | 1 |
| Complejidad de mantenimiento | (-) | 1 | 1 |
| Cantidad de pruebas | (-) | 1 | 1 |

Tabla 15 Rango de valores para la evaluación de la relación atributo/métrica [40].

| Categoría | Rango de Valores |
|---|---|
| Malo | <= 0.4 |
| Regular | >0.4 y <=0.7 |
| Bueno | >0.7 |

La Ilustración 24 muestra la gráfica de los resultados obtenidos de los atributos de calidad evaluados en las métricas aplicadas anteriormente, donde todos los atributos de calidad mantienen un buen comportamiento.

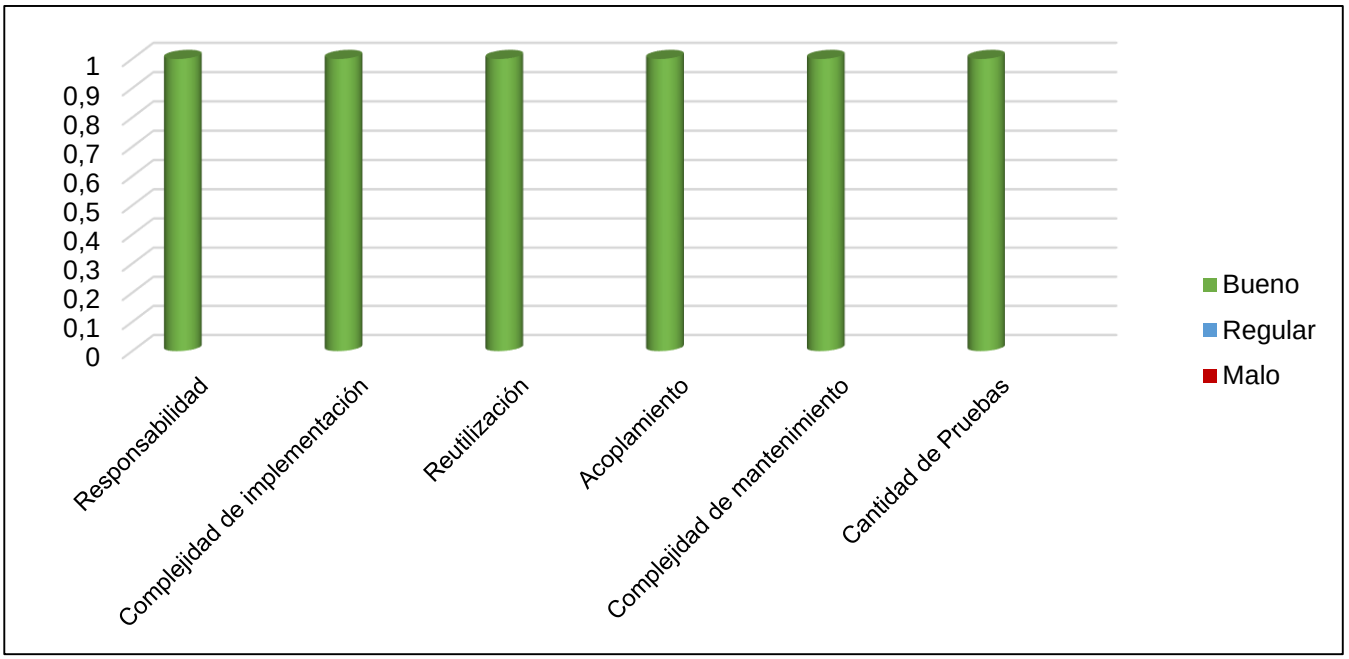



Ilustración 24 Resultados obtenidos de la evaluación de los atributos de calidad.

## 3.3 Pruebas de Software

El único instrumento adecuado para determinar la calidad de un producto de software es el proceso de pruebas. En este proceso se ejecutan pruebas dirigidas a componentes del software

o al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos. [41]

### 3.3.1 Prueba de caja blanca

El método de caja blanca permite desarrollar pruebas de tal forma que asegure que la operación interna se ajusta a las especificaciones, y que todos los componentes internos se han probado de forma adecuada. Realiza un examen minucioso de los detalles procedimentales, comprobando los caminos lógicos del programa, comprobando los bucles y condiciones, y examinado el estado del programa en varios puntos [42].

Las pruebas de caja blanca intentan garantizar que:

- ✓ se ejecutan al menos una vez todos los caminos independientes de cada módulo.
- ✓ se utilizan las decisiones en su parte verdadera y en su parte falsa.
- ✓ se ejecuten todos los bucles en sus límites.
- ✓ se utilizan todas las estructuras de datos internas.

Al mecanismo de captura de metadatos de una base de datos se le realizaron pruebas de caja blanca utilizando la técnica de camino básico. Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico de caminos de ejecución (caminos independientes), a partir de los cuales se elaboran los casos de prueba, que garantizan la ejecución de cada sentencia del programa al menos una vez, durante las pruebas [42].

### 3.3.2 Aplicación de pruebas de Caja Blanca

Para la ejecución de la técnica de camino básico se aplicaron cuatro pasos fundamentales:

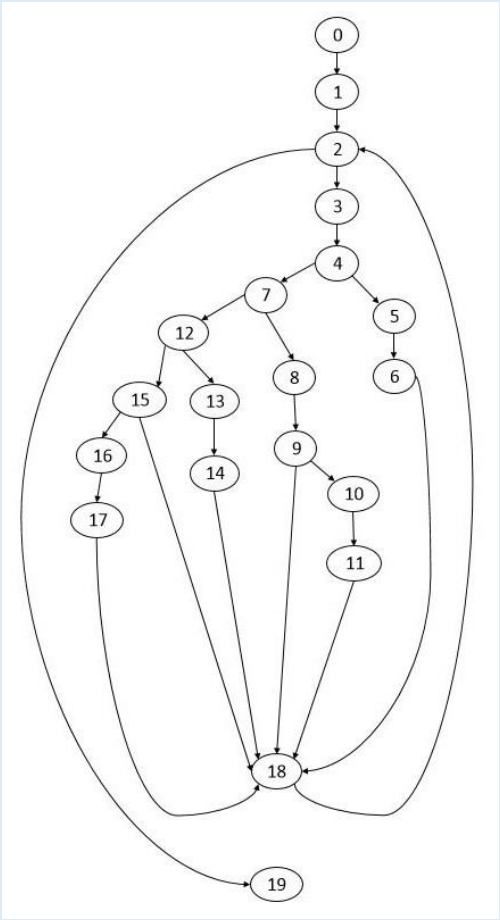
- ✓ A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
- ✓ Se calcula la complejidad ciclomática del grafo.
- ✓ Se determina un conjunto básico de caminos independientes.
- ✓ Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

| MetadataManager. getUniqueConstraints | |
|---|---|
| 0 | private LinkedList<Unique> getUniqueConstraints(String schema,<br>String tableName) throws SQLException { |
| 1 | LinkedList<Unique> uniques = new LinkedList<>(); |

```
1               ResultSet uniqueDef = getUniqueDef(schema, tableName);
2               while (uniqueDef.next()) {
3                       Unique unique = new Unique(uniqueDef.getString("UNIQUE_NAME"));
4                       if(dialect instanceof Oracle9iDialect){
5                               String deferrable = new String("NOT DEFERRABLE");
5
unique.setDeferrable(uniqueDef.getString("DEFERRABLE").equals(deferrable));
5                               String deferred=new String("IMMEDIATE");
5                               unique.setDeferred(uniqueDef.getString("DEFERRED").equals(deferred));
5                               unique.setStatus(uniqueDef.getString("STATUS"));
5                               unique.addColumn(uniqueDef.getString("COLUMN_NAME"));
6                       }
7                       else if (dialect instanceof PostgresDialect) {
8                               unique.setDeferrable(uniqueDef.getBoolean("DEFERRABLE"));
8                               unique.setDeferred(uniqueDef.getBoolean("DEFERRED"));
8                               Integer[] conkeys = (Integer[]) uniqueDef.getArray(
                                                "COLUMN_NUMS").getArray();
8                               unique.setConKeys(conkeys);
8                               String relOptionsClass = uniqueDef.getString("OPTIONS");
9                               if (relOptionsClass != null) {
10                                      relOptionsClass = relOptionsClass.replace("{fillfactor=","");
10                                      relOptionsClass = relOptionsClass.replace("}", "");
10                                      unique.setFillFactor(Integer.parseInt(relOptionsClass));
11                              }
12                      } else if (dialect instanceof MySqlServerDialect) {
13                              unique.addColumn(uniqueDef.getString("COLUMN_NAME"));
14                      }
15                      else if (dialect instanceof SqlServerDialect) {
16                              unique.addColumn(uniqueDef.getString("Column_Name"));
```

```
17                          }
18                          uniques.add(unique);}
19                  return uniques;  }
```

**Construcción del grafo correspondiente al código fuente**




Complejidad ciclomática:

A= 26 (aristas)

N= 20 (nodos)

V(G) = A – N + 2

V(G) = 25 – 20 + 2 = 7

P = 7(nodos predicados)

V(G) = P + 1

V (G) = 6 + 1.

V(G) = 7

1. 0,1,2,19
2. 0,1,2,3,4,5,6,18,2,19
3. 0,1,2,3,4,7,8,9,18,19
4. 0,1,2,3,4,7,8,9,10,11,18,2,19
5. 0,1,2,3,4,7,12,13,14,18,2,19
6. 0,1,2,3,4,7,12,15,18,2,19
7. 0,1,2,3,4,7,12,15,16,17,18,2,19

Tabla 16: Casos de pruebas que fuerzan la ejecución de cada camino.

| No | Caso de Prueba | Resultado esperado |
|---|---|---|
| **1** | El resultset uniqueDef está vacío. | Retorna una lista de unique vacía. |
| **2** | El dialecto identificado es Oracle. | Retorna una lista de unique. |
| **3** | El dialecto identificado es PostgreSQL y el atributo relOptionsClass es null. | Retorna una lista de unique sin atributo fillfactor. |

| | | |
|---|---|---|
| **4** | El dialecto identificado es PostgreSQL y el atributo relOptionsClass es distinto de null. | Retorna una lista de unique con atributo fillfactor. |
| **5** | El dialecto identificado es MySQL. | Retorna una lista de unique. |
| **6** | No se ha identificado ningún dialecto. | Retorna una lista de unique. |
| **7** | El dialecto identificado es Micorsoft SQL Server. | Retorna una lista de unique. |

Mediante la técnica de camino básico se pudo conocer que debería devolver cada método según la información que se capturaba de la BD.

### 3.3.3 Pruebas Unitarias

Es la prueba enfocada a los elementos más pequeño del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera. La prueba de unidad siempre está orientada a caja blanca [42].

Para comprobar cada caso de prueba, se desarrollaron pruebas unitarias a los métodos a los cuales se les aplicó la técnica del camino básico. Las pruebas unitarias permitieron verificar el comportamiento de la clase *MetadataManager* y sus métodos. Mediante estas pruebas fue posible comparar la respuesta que debería obtenerse de un método con lo que realmente se adquiría de él.

Para la ejecución de estas pruebas fue necesario diseñar una clase denominada *MetadataManagerTest* la cual contenía métodos de la clase *MetadataManager* a los que se realizarían las pruebas. A continuación se muestran algunas de las pruebas realizadas y el resultado que arrojaron.

```
public void testGetPrimaryKey() throws SQLException {
            String nuevo="nuevo";
            String prueba="prueba";
            PrimaryKey esperado= new PrimaryKey();
            String name="autor_pkey";
            String columns[] ={"id_autor"};
            int  fillFactor=0;
            boolean deferrable=false;
            boolean deferred=false;

            for(int i=0; i<1; i++){
                  esperado.addColumn(columns[i]);
            }
            esperado.setDeferrable(deferrable);
            esperado.setDeferred(deferred);
            esperado.setFillFactor(fillFactor);
            esperado.setName(name);
            PrimaryKey actual= metadatamanager.getPrimaryKey(nuevo, prueba);
```

```
            assertEquals(esperado, actual); }
public void testGetView() throws SQLException{
      View esperado=new View();
      String schema="nuevo";
      String name="viewprueba";
      String query=" SELECT autor.id_autor,    autor.nombre,    autor.apellidos
FROM autor;";
      boolean securityBarrier=false;
      String security="null";
      String definer="null";
      String checkOption="null";
      esperado.setName(name);
      esperado.setCheckOption(checkOption);
      esperado.setQuery(query);
      esperado.setDefiner(definer);
      esperado.setSecurity(security);
      esperado.setSecurityBarrier(securityBarrier);
      View actual= metadatamanager.getView(schema,name);
      assertEquals(esperado, actual);
}
```

En total se desarrollaron 13 casos de pruebas, en una primera iteración se obtuvo 4 no conformidades debido a que no se capturaban todos los elementos que conformaban las diferentes estructuras de la BD. En una segunda iteración se le dieron solución a los problemas detectados en la primera iteración, por lo que el resultado de las pruebas fue satisfactorio. A continuación se ilustran los resultados obtenidos en la vista de JUnit, los métodos que tienen una cruz azul fueron los que resultaron fallidos:

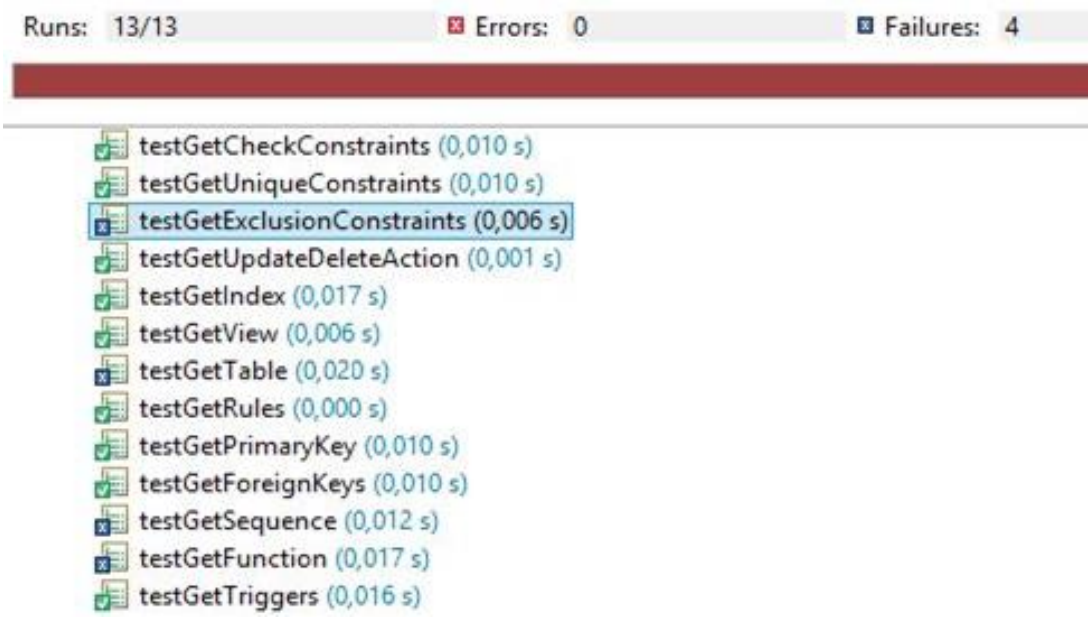


Ilustración 25: Resultados de la primera etapa de pruebas

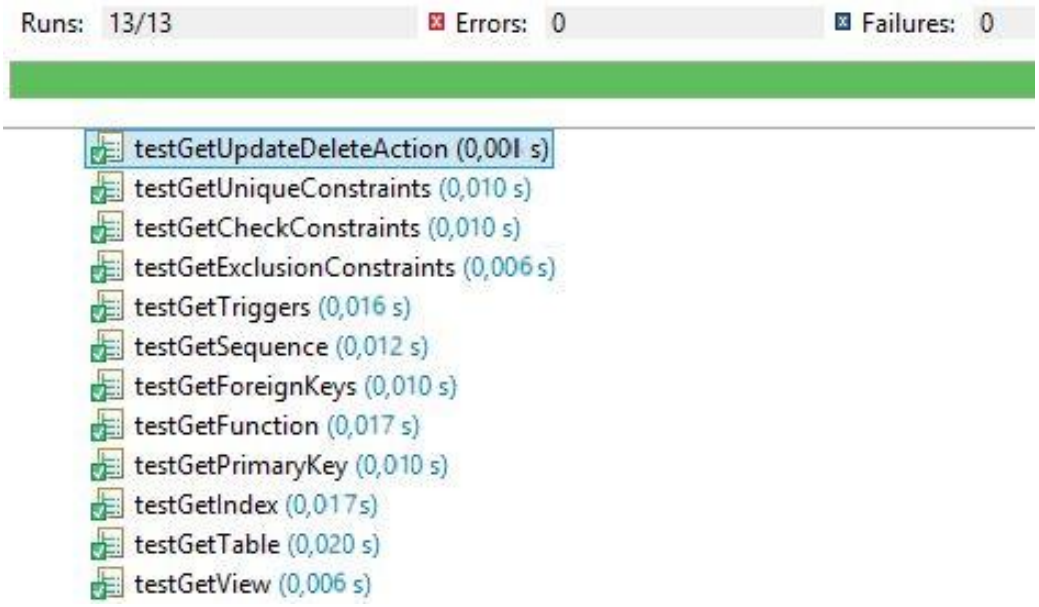



Ilustración 26: Resultados de la segunda etapa de pruebas

## 3.4 Resultados obtenidos

- ✓ La aplicación demostró ser completamente funcional, dando solución a todos los requisitos planteados.
- ✓ Los errores detectados durante el proceso de prueba fueron solucionados.
- ✓ Todas las pruebas dieron resultados satisfactorios, de esta manera se demuestra que el sistema cumple con cada uno de los casos de uso que lo conforman.

A continuación se muestra una gráfica donde se compara el tiempo de ejecución del mecanismo anterior 0,025 segundos, con el del nuevo mecanismo desarrollado 0,135 segundos, obteniéndose como resultado una diferencia de 0,115 segundos entre ambos mecanismo, dicho valor no afecta el tiempo de ejecución del Replicador de datos REKO.

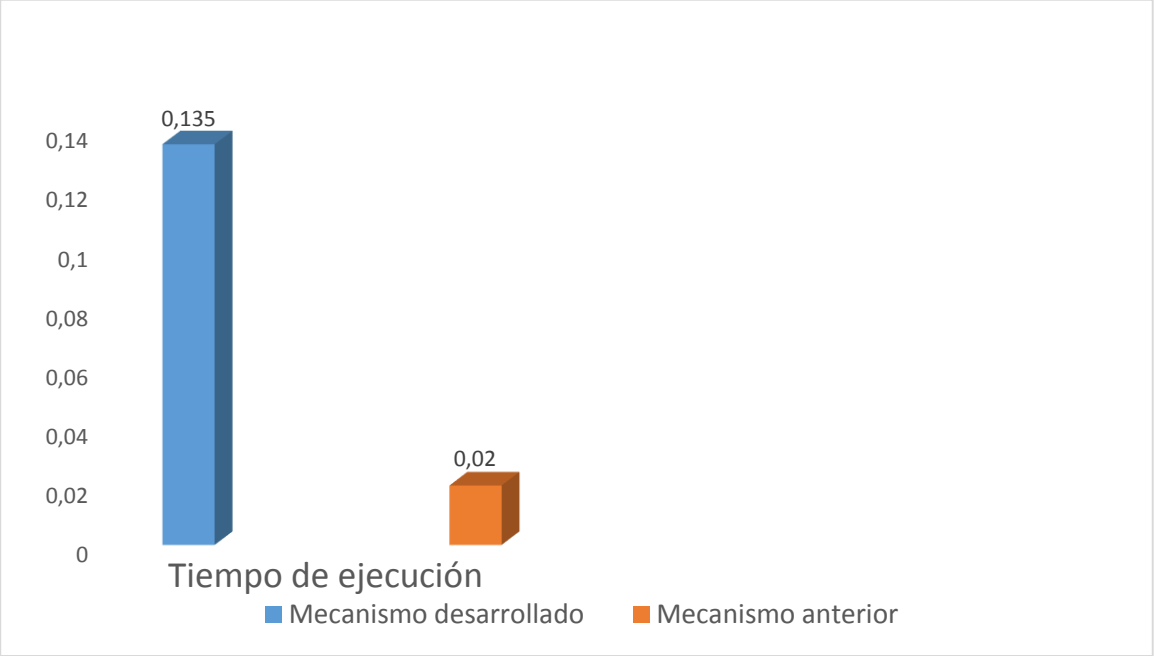



Ilustración 27: Gráfica de comparación de los tiempos de ejecución

## 3.5 Conclusiones parciales

- ✓ Se expuso el estándar de codificación utilizado en la implementación del mecanismo.
- ✓ Se elaboró un diagrama de componentes para lograr un mejor entendimiento de cómo estará físicamente distribuido el sistema.
- ✓ Se validó el diseño a través de las métricas Tamaño Operacional de Clases y Relaciones entre Clases, lo que permitió corroborar el buen comportamiento que mantienen todos los atributos de calidad.
- ✓ Se diseñaron los casos de prueba para rectificar a tiempo defectos que pudiera presentar el sistema.
- ✓ Se validó la solución propuesta a través del método de caja blanca, aplicando pruebas unitarias y la técnica de camino básico.

# Conclusiones generales

Con el desarrollo del presente trabajo de diploma se concluye que:

- ✓ Mediante el análisis de la literatura especializada fueron identificados los elementos teóricos que sustentan el desarrollo de la investigación.
- ✓ Se realizó un estudio de las tecnologías de captura de metadatos, lo que permitió definir que tecnologías utilizar en la implementación, en aras de desarrollar un mecanismo de captura más completo.
- ✓ Se logró diseñar e implementar un mecanismo para la captura de metadatos de una BD para todos los SGBD a los que el Replicador de datos REKO brinda soporte: ProstgreSQL, MySQL, Microsoft SQL Server y Oracle; el cual permite la captura de la información necesaria de la estructura de una BD para el proceso de réplica de estructura.
- ✓ Se realizaron pruebas de caja blanca durante el proceso de desarrollo de la solución propuesta, lo cual permitió mitigar los errores detectados en las iteraciones de pruebas, para garantizar un mejor funcionamiento de la solución, así como la factibilidad de su uso como parte del Replicador de datos REKO.

## Recomendaciones

- ✓ Capturar los metadatos de bases de datos para otros SGBD como *SQLite* y *FireBird*.
- ✓ Desarrollar una API propia del Replicador de datos REKO que permita capturar toda la información, sin tener que hacer consultas directas a la BD.

## Bibliografía Referenciada


1. CEJAROSU. *Mecanismos*. [Web Page] 2005 2005 [cited 2014 07-12]; Available from: http://concurso.cnice.mec.es/cnice2006/material107/maquinas/maq_mecanismos.htm.
2. Lapuente, M.J.L. *Hipertexto, el nuevo concepto de documento en la cultura de la imagen*. [Web Page] 2013 08/12/2013 [cited 2014 07-12]; Available from: www.hipertexto.info.
3. [Web Page] 2014 07-12-2014 [cited 2014 07-12]; Available from: http://www.mastermagazine.info/.
4. Héctor Andrés Melgar Sasieta, A.E.D.R., *Modelado de la Metadata para el Desarrollo de Herramientas de Productividad sobre Múltiples Manejadores de Base de Datos Relacionales*
5. Oracle. *Interface DatabaseMetaData*. [cited 2014 08-12]; Available from: http://docs.oracle.com/javase/7/docs/api/java/sql/DatabaseMetaData.html.
6. Center, I.K. *Acceder a la base de datos con el controlador JDBC*. Available from: IBM Knowledge Center.com.
7. Peña, N.M., *Desarrollo de un dialecto de SQL Server para la réplica de datos en el software Reko.* 2013, Universidad de las Ciencias Informáticas. p. 80.
8. Sardiña, Y.C., *REKO: UNA SOLUCIÓN DE RÉPLICA PARA SISTEMAS DE BASES DE DATOS RELACIONALES DISTRIBUIDOS.* 2007: IV Simposio Informática y Comunidad. p. 9.
9. PRESSMAN, R.S., *Ingenieria del Software, Un Enfoque Práctico*. 2002.
10. Nieves, I.I.S., *Metodologías Ágiles - Proceso Unificado Ágil.* 2014: p. 9.
11. Torrecilla, P. *mas sobre el-proceso unificado agil-fases y disciplinas*. 2013.
12. R.T.Sánchez, *Metodología de desarrollopara la actividad productiva de la UCI.* 2014.
13. *Visual Paradigm Features*. 23-12-2014 17-01-15]; Available from: http://www.visual-paradigm.com/features/.
14. Linda DeMichiel, B.S., *Java™ Platform, Enterprise Edition (Java EE) Specification, v7* Oracle, Editor. 2012, Oracle.
15. GROUP, O.M. *Introduction to OMG's Unified Modeling Language (UML)*. 2015 27/05/2015 18/05/2015]; Available from: http://www.omg.org/gettingstarted/what_is_uml.htm.
16. *Programación Java*. 2009 [cited 2015 21/06/2015]; Available from: http://www.lenguajes-de-programacion.com/programacion-java.shtml.
17. Eclipse. *About the Eclipse Foundation*. 2015 [cited 2015-04-28; Available from: http://www.eclipse.org/org/.
18. DUHARTE, F.R., *Pruebas Unitarias de Software en la Plataforma J2EE*. 2008.
19. MICROSOFT. *Microsoft SQL Server 2008*. 2013 24-06-2015; Available from: http://www.microsoft.com/latam/sql/2008/default.mspx.
20. HernanOkamura *Sistema Gestores de Bases de Datos*. 2012.
21. Larman, C., *UML y Patrones*.
22. PRESSMAN, R.S., *Ingeniería del Software, Un Enfoque Práctico*. 6ta ed. 2002.
23. Sommerville, I., *Ingenieria de SW* Séptima Edisión ed. Vol. Séptima Edisión. 2005.
24. *Normas ISO 25010* 2011.
25. technologies, q.-v., *Generación y Verificación de Historia de Usuario.* 2014.
26. Len Bass, P.C., Rick Kazman, *Software Architecture in Practice*. Vol. 2nd Edition. 2010: Department of Computer Science.
27. Microsoft. 2015 24-06-2015]; Available from: http://msdn.microsoft.com/es-es/library/bb972268.aspx.

28. CDAE, *ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE*. 2014, PROYECTO REPLICADOR DE DATOS REKO
29. campo, G.D., *Patrones de Diseño, Refactorización y Antipatrones.w*. 2009.
30. Saavedra Gutierrez, J.A., *El Mundo Informático*. 2007.
31. Grosso, A., *Patrones GRASP* 2011.
32. Navarro, J.M., *Diseño del Sistema de Tarjeta de Crédito Con UML*. Oficina General del Sistema de Bibliotecas y Biblioteca Central UNMSM.
33. Demién Gutierrez, *UML Diagramas de Paquetes*. 2009: Universidad de los Andes.
34. Microsoft. *Diagramas de clases de UML*. 2013 [cited 2015 25-06-2015]; Available from: https://msdn.microsoft.com/es-es/library/dd409437.aspx.
35. Marca Huallpara, H.M.y.Q.L., Nancy Susana, *Diagramas de Despliegue*. Universidad Salesiana Bolivia. p. 6.
36. Microsoft. *Diagramas de componentes de UML: Referencia*. 2015 [cited 2015 2015-05-10]; Available from: https://msdn.microsoft.com/es-es/library/dd409390.aspx.
37. System, S. *Diagrama de Componentes UML 2*. [cited 2015 25/06/2015]; Available from: http://www.sparxsystems.com.ar/resources/tutorial/uml2_componentdiagram.html.
38. *Codigo Fuente*. 2008 [cited 20015 21/06/2015]; Available from: http://definicion.de/codigo/.
39. Jerez, G.R.L., *ESTÁNDAR DE CODIFICACIÓN JAVA PROYECTO REPLICADO DE DATOS REKO V4.0*. 2014.
40. González, R.P., *Drivers de autenticación para el sistema de seguridad ACAXIA.* 2011, Universidad de las Ciencias Informáticas.
41. *Las pruebas de software*. 2005 [cited 2015 24-06-2015; Available from: http://pruebasdesoftware.com/laspruebasdesoftware.htm.
42. *Ingenieria de SW*, in *Disciplina Prueba*. 2013, Universidad de las Ciencias Informáticas.

# Anexos

## Anexo 1 Descripción de los requisitos no funcionales.

| | |
|---|---|
| **Atributo de Calidad** | Fiabilidad. |
| **Sub-atributos/Sub-característica** | Tolerancia a fallos y Recuperabilidad. |
| **Objetivo** | Capacidad del producto para operar según lo previsto en presencia de fallos de hardware o software.<br>Capacidad del producto para recuperar los datos directamente afectados y reestablecer el estado deseado del sistema en caso de interrupción o fallos. |
| **Origen** | Proveedor de requisitos. |
| **Artefacto** | El sistema y el código fuente. |
| **Entorno** | El sistema desplegado y existen fallos en la red y eléctricos. |
| **Estímulo** | **Respuesta: Flujo de eventos (Escenarios)** |
| **Recuperación ante un fallo del fluido eléctrico** | |
| Fallos eléctricos. | 1. El sistema se encuentra capturando los metadatos de la base de datos.<br>2. El sistema falla por ausencia del fluido eléctrico.<br>3. El sistema se reestablece antes el fallo eléctrico iniciando de forma automática con el sistema operativo y continua la captura de los metadatos.<br>4. El sistema concluye el proceso antes afectado satisfactoriamente. |
| **Medida de respuesta** | |
| Navegar en el sistema en presencia de fallos eléctricos. | |

| | |
|---|---|
| **Atributo de Calidad** | Fiabilidad. |
| **Sub-atributos/Sub-característica** | Madurez. |

| | |
|---|---|
| **Objetivo** | Capacidad del producto para satisfacer las necesidades de fiabilidad en condiciones normales. |
| **Origen** | Arquitecto de software. |
| **Artefacto** | El sistema. |
| **Entorno** | El sistema desplegado y funcionando en periodos de tiempos determinados. |
| **Estímulo** | **Respuesta: Flujo de eventos (Escenarios)** |
| **El sistema desplegado en los escenarios de réplica especificados por el cliente** | |
| El sistema funciona correctamente bajo condiciones y periodos de tiempos determinados por el cliente. | NA |
| **Medida de respuesta** | |
| Desplegar el sistema. | |

| | |
|---|---|
| **Atributo de Calidad** | Fiabilidad. |
| **Sub-atributos/Sub-característica** | Disponibilidad. |
| **Objetivo** | Capacidad del producto de estar operativo y accesible para su uso cuando se requiere. |
| **Origen** | Proveedor de requisitos. |
| **Artefacto** | El sistema. |
| **Entorno** | El sistema desplegado, funcionando las 24 horas del día. |
| **Estímulo** | **Respuesta: Flujo de eventos (Escenarios)** |
| **l sistema desplegado en los escenarios de réplica especificados por el cliente las 24 horas del día** | |
| El sistema funcionando en modo *background* como parte de un proceso demonio del sistema operativo. | NA |
| **Medida de respuesta** | |
| Desplegar el sistema. | |

| | |
|---|---|
| **Atributo de Calidad** | Mantenibilidad. |
| **Sub-atributos/Sub-característica** | Analizabilidad. |

| | |
|---|---|
| **Objetivo** | Facilidad con la que se puede evaluar el impacto de un determinando cambio sobre el resto del software, diagnosticar las deficiencias o causas de fallos en el software, o identificar las partes a modificar. |
| **Origen** | Arquitecto de software. |
| **Artefacto** | El código fuente. |
| **Entorno** | El ambiente de desarrollo del sistema. |
| **Estímulo**<br>**Impacto de un determinado cambio sobre el resto del sistema** | **Respuesta: Flujo de eventos (Escenarios)** |
| Se evalúa el impacto de un cambio en el sistema. | 1. El arquitecto de software identifica los paquetes y clases implicados en el cambio, así como las funciones que se reutilizarán o se crearán para introducir el cambio.<br>2. Se evalúa el impacto partiendo de los resultados que arrojó el análisis anterior con respecto a la arquitectura del sistema. |
| **Diagnosticar las deficiencias o causas de fallos en el sistema** | **Respuesta: Flujo de eventos (Escenarios)** |
| Se diagnostica las deficiencias o causas de fallos en el sistema. | 1. El arquitecto de software identifica las posibles deficiencias o causas de fallos que se pueden originar en el sistema.<br>2. Se diagnostican las deficiencias o causas de fallos partiendo de los resultados que arrojó el análisis anterior. |
| **Identificar las partes a modificar** | **Respuesta: Flujo de eventos (Escenarios)** |
| Se identifica las partes a modificar en el sistema. | 1. El arquitecto de software identifica los paquetes y clases implicados en el cambio, así como las funciones que se reutilizarán o se crearán para introducir el cambio. |
| **Medida de respuesta** | |
| Analizar el cambio en el sistema. | |

| | |
|---|---|
| **Atributo de Calidad** | Mantenibilidad. |
| **Sub-atributos/Sub-característica** | Modificabilidad. |

| | |
|---|---|
| **Objetivo** | Capacidad del producto que permite que sea modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño. |
| **Origen** | Desarrollador del mecanismo. |
| **Artefacto** | El código fuente. |
| **Entorno** | El ambiente de desarrollo del sistema. |
| **Estímulo** | **Respuesta: Flujo de eventos (Escenarios)** |
| **Capacidad de modificación del sistema** | |
| La arquitectura del sistema está diseñada para brindar facilidades a la hora de introducir modificaciones en el sistema. Esto permite que se puedan introducir cambios o modificaciones, que no afecten el correcto desempeño del resto de la solución. | NA |
| **Medida de respuesta** | |
| Introducir una modificación al sistema. | |

| | |
|---|---|
| **Atributo de Calidad** | Mantenibilidad. |
| **Sub-atributos/Sub-característica** | Capacidad para ser probado. |
| **Objetivo** | Facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios. |
| **Origen** | Arquitecto de software y el analista |
| **Artefacto** | Diseños de casos de pruebas. |
| **Entorno** | El sistema desplegado. |
| **Estímulo** | **Respuesta: Flujo de eventos (Escenarios)** |
| **Facilidad con la que se pueden establecer criterios de prueba para el sistema** | |
| Desde la concepción inicial del sistema se definen y delimitan las funciones según los requisitos funcionales y no funcionales a implementar especificando los argumentos o | NA |

| | |
|---|---|
| variables de entrada y salida del sistema, elementos que favorecen el proceso de identificación y elaboración de los distintos escenarios de prueba. | |
| **Medida de respuesta** | |
| Escenario de prueba del sistema. | |

| | |
|---|---|
| **Atributo de Calidad** | Eficiencia en el rendimiento. |
| **Sub-atributos/Sub-característica** | Comportamiento en el tiempo. |
| **Objetivo** | Grado en que los tiempos de respuesta, procesamiento y las tasas de rendimiento de un producto o sistema, al realizar sus funciones cumplen con los requisitos. |
| **Origen** | Arquitecto de software. |
| **Artefacto** | El sistema. |
| **Entorno** | El sistema desplegado. |
| **Estímulo** | **Respuesta: Flujo de eventos (Escenarios)** |
| **Para escenario 1** | |
| Tiempo de respuesta. | 1. El sistema funcionando correctamente. |
| **Estímulo** | **Respuesta: Flujo de eventos (Escenarios)** |
| **Para escenario 2** | |
| Tiempo de respuesta. | 1. El sistema funcionando correctamente. |
| **Medida de respuesta** | |
| Navegar en el sistema. | |

| | |
|---|---|
| **Atributo de Calidad** | Eficiencia en el rendimiento. |
| **Sub-atributos/Sub-característica** | Utilización de recursos. |
| **Objetivo** | Grado en el que las cantidades y tipos de recursos utilizados por un producto o sistema, al realizar sus funciones cumplen con los requisitos. |
| **Origen** | Arquitecto de software. |

| **Artefacto** | El sistema. |
|---|---|
| **Entorno** | El sistema desplegado. |
| **Estímulo**<br>**Características de Hardware** | **Respuesta: Flujo de eventos (Escenarios)** |
| **Servidor de aplicación**<br>• Cantidad: 2 servidores.<br>• CPU: 4x2.33GHz.<br>• RAM: 4Gb.<br>• HDD: 200 Gb RAID 5.<br>• LAN: 2 x NIC (10/100MBbit).<br>• Fuentes de Alimentación: 2x500W. | 1. El sistema funcionando correctamente. |
| **Estímulo**<br>**Características de Software** | **Respuesta: Flujo de eventos (Escenarios)** |
| • Tener disponible al menos 2 puertos para el uso de la aplicación.<br>• Instalador de la aplicación de REKO.<br>• Instalador de la aplicación ActiveMQ.<br>• Un navegador web (Firefox). | 1. El sistema funcionando correctamente. |
| **Medida de respuesta** | |
| Navegar en el sistema. | |

## Anexo 2 Criterios de evaluación de la métrica TOC [40]

| Atributo de Calidad | Categoría | Criterio |
|---|---|---|
| Responsabilidad | Baja | <=Promedio |
| | Media | Entre Promedio y 2*Promedio |
| | Alta | >Promedio |
| Complejidad de implementación | Baja | <=Promedio |
| | Media | Entre Promedio y 2*Promedio |
| | Alta | >2*Promedio |
| Reutilización | Baja | >2*Promedio |
| | Media | Entre Promedio y 2*Promedio |
| | Alta | <=Promedio |

## Anexo 3 Criterios de evaluación de la métrica RC [40]

| Atributo de Calidad | Categoría | Criterio |
|---|---|---|
| Acoplamiento | Ninguno | 0 |
| | Baja | 1 |
| | Media | 2 |
| | Alta | >2 |
| Complejidad de mantenimiento | Baja | <=Promedio |
| | Media | Entre Promedio y 2*Promedio |
| | Alta | >2*Promedio |
| Reutilización | Baja | >2*Promedio |
| | Media | Entre Promedio y 2*Promedio |
| | Alta | <=Promedio |
| Cantidad de pruebas | Baja | <=Promedio |
| | Media | Entre Promedio y 2*Promedio |
| | Alta | >2*Promedio |

## Anexo 4 Instrumento de evaluación de la métrica TOC

| Clase | Cantidad de procedimientos | Responsabilidad | Complejidad de implementación | Reutilización |
|---|---|---|---|---|
| Check | 8 | Baja | Baja | Alta |
| Column | 14 | Baja | Baja | Alta |
| Constraint | 2 | Baja | Baja | Alta |
| DatabaseStructure | 14 | Baja | Baja | Alta |
| DataBaseStructures | 2 | Baja | Baja | Alta |
| MetadataManager | 43 | Alta | Alta | Baja |
| Exclusion | 11 | Baja | Baja | Alta |
| ExclusionColumnData | 10 | Baja | Baja | Alta |
| ForeignKey | 18 | Media | Media | Media |
| Function | 37 | Alta | Alta | Baja |
| FunctionArgs | 8 | Baja | Baja | Alta |
| Index | 21 | Media | Media | Media |
| IndexColumnData | 10 | Baja | Baja | Alta |
| PrimaryKey | 9 | Baja | Baja | Alta |
| Rule | 4 | Baja | Baja | Alta |
| Sequence | 14 | Baja | Baja | Alta |
| Table | 33 | Media | Media | Media |
| Trigger | 33 | Media | Media | Media |
| Unique | 13 | Baja | Baja | Alta |
| View | 12 | Baja | Baja | Alta |
| DialectUtils | 19 | Media | Media | Media |
| HibernateDialect | 19 | Media | Media | Media |
| PostgresDialect | 19 | Media | Media | Media |
| MySqlServerDialect | 19 | Media | Media | Media |

| | | | | |
|---|---|---|---|---|
| Oracle9iDialect | 19 | Media | Media | Media |
| SqlServerDialect | 19 | Media | Media | Media |
| **Promedio de métodos por clases** | **16,53846** | | | |

## Anexo 5 Instrumento de evaluación de la métrica RC

| Clase | Cantidad de relaciones de uso | Acoplamiento | Complejidad de mantenimiento | Reutilización | Cantidad de pruebas |
|---|---|---|---|---|---|
| Check | 1 | Baja | Baja | Alta | Baja |
| Column | 1 | Baja | Baja | Alta | Baja |
| Constraint | 6 | Alta | Alta | Baja | Alta |
| DatabaseStructure | 5 | Alta | Alta | Baja | Alta |
| DataBaseStructures | 1 | Baja | Baja | Alta | Baja |
| MetadataManager | 2 | Media | Baja | Alta | Baja |
| Exclusion | 2 | Media | Baja | Alta | Baja |
| ExclusionColumnData | 1 | Baja | Baja | Alta | Baja |
| ForeignKey | 1 | Baja | Baja | Alta | Baja |
| Function | 2 | Media | Baja | Alta | Baja |
| FunctionArgs | 1 | Baja | Baja | Alta | Baja |
| Index | 2 | Media | Baja | Alta | Baja |
| IndexColumnData | 1 | Baja | Baja | Alta | Baja |
| PrimaryKey | 1 | Baja | Baja | Alta | Baja |
| Rule | 1 | Baja | Baja | Alta | Baja |
| Sequence | 1 | Baja | Baja | Alta | Baja |
| Table | 7 | Alta | Alta | Baja | Alta |
| Trigger | 1 | Baja | Baja | Alta | Baja |
| Unique | 1 | Baja | Baja | Alta | Baja |
| View | 1 | Baja | Baja | Alta | Baja |
| DialectUtils | 2 | Media | Baja | Alta | Baja |
| HibernateDialect | 5 | Alta | Alta | Baja | Alta |
| PostgresDialect | 4 | Alta | Media | Media | Media |
| MySqlServerDialect | 4 | Alta | Media | Media | Media |
| Oracle9iDialect | 4 | Alta | Media | Media | Media |
| SqlServerDialect | 4 | Alta | Media | Media | Media |
| **Promedio** | **2,38461538** | | | | |

## Anexo6 Descripción de las clases.

| **Descripción de la clase Table** | |
|---|---|
| **Nombre:** Table | |
| **Tipo de clase:** Entidad | |
| **Atributo** | Tipo |

| | |
|---|---|
| name | private |
| columsMap | private |
| oid | private |
| inherits | private |
| ruleMap | private |
| triggerMap | private |
| constraintsMap | private |
| indexsMap | private |
| **Responsabilidades** | |
| **Nombre:** addColumns(column: Column) | |
| **Descripción:** Adiciona una columna a la tabla. | |
| **Nombre:** addIndex(index: Index). | |
| **Descripción:** Adiciona un index a la tabla. | |
| **Nombre:** addConstraint(constraint: Constraint) | |
| **Descripción:** Adiciona un constraint a la tabla. | |
| **Nombre:** addRule(rule: Rule) | |
| **Descripción:** Adiciona una regla a la tabla. | |
| **Nombre:** setName(name: String) | |
| **Descripción:** cambia el valor actual de name por el pasado por parámetro. | |
| **Nombre:** getName(): String | |
| **Descripción:** devuelve el nombre de la tabla. | |
| **Nombre:** setInherit(inherit: String) | |
| **Descripción:** cambia el valor actual de inherit por el pasado por parámetro. | |
| **Nombre:** getInherit():String | |
| **Descripción:** devuelve el valor de inherit. | |
| **Nombre:** getEngine(): int | |
| **Descripción:** devuelve el valor de engine. | |
| **Nombre:** setEngine(String) | |
| **Descripción:** Cambia el valor del engine por el pasado por parámetro. | |
| **Nombre:** getRowFormat():String | |
| **Descripción:** devuelve el valor del rowformat. | |
| **Nombre:** setRowFormat(String rowFormat) | |

| |
|---|
| **Descripción:** Cambia el valor del rowformatpor el pasado por parámetro. |

| **Descripción de la clase Column** | |
|---|---|
| **Nombre:** Column | |
| **Tipo de clase:** Entidad | |
| **Atributo** | Tipo |
| name | private |
| dataType | private |
| typeName | private |
| defaultValue | private |
| admitNull | private |
| size | private |
| ordinalPosition | private |
| **Responsabilidades** | |
| **Nombre:** setName(name: String) | |
| **Descripción:** cambia el valor actual de name por el pasado por parámetro. | |
| **Nombre:** getName():String | |
| **Descripción:** devuelve el nombre de la columna. | |
| **Nombre:** setDataType (inherit: int) | |
| **Descripción:** cambia el valor actual de dataType por el pasado por parámetro. | |
| **Nombre:** getDataType (): int | |
| **Descripción:** devuelve el valor de dataType. | |
| **Nombre:** getTypeName ():String | |
| **Descripción:** devuelve el valor de typeName. | |
| **Nombre:** setTypeName (String: typeName) | |
| **Descripción:** Cambia el valor del typeName por el pasado por parámetro. | |
| **Nombre:** getDefaultValue (): int | |
| **Descripción:** devuelve el valor del defaultValue. | |
| **Nombre:** setAdmitNull (boolean: defaultValue) | |
| **Descripción:** Cambia el valor del admitNull por el pasado por parámetro. | |
| **Nombre:** isAdmitNull (): boolean | |
| **Descripción:** devuelve el valor del admitNull. | |
| **Nombre:** setSize (int: size) | |
| **Descripción:** Cambia el valor del size por el pasado por parámetro. | |
| **Nombre:** getSize (): int | |
| **Descripción:** devuelve el valor del size. | |

| |
|---|
| **Nombre:** getOrdinalPosition (): int |
| **Descripción:** devuelve el valor de ordinalPosition. |
| **Nombre:** setTypeName (int: ordinalPosition) |
| **Descripción:** Cambia el valor del ordinalPosition por el pasado por parámetro. |
| **Descripción de la clase DialectUtils** |
| **Nombre:** DialectUtils |
| **Tipo de clase:** Entidad |
| **Responsabilidades** |
| **Nombre:** getUniqueDef(dialect: HibernateDialect, schema: String, tableName: String): String |
| **Descripción:** devuelve una sentencia SQL que captura los atributos de un *Unique.* |
| **Nombre:** getInheritTables(dialect: HibernateDialect, schema: String, tableName: String): String |
| **Descripción:** devuelve una sentencia SQL que captura el atributo *Inherit* de una tabla. |
| **Nombre:** getSequenceNames(dialect: HibernateDialect, schema: String): String |
| **Descripción:** devuelve una sentencia SQL que captura todos los nombres de las *sequence*. |
| **Nombre:** hasTableOid(dialect: HibernateDialect, schema: String, tableName: String): String |
| **Descripción:** devuelve una sentencia SQL que captura el atributo *Oid* de una tabla. |
| **Nombre:** getViewDef(dialect: HibernateDialect, schema: String, viewName: String): String |
| **Descripción:** devuelve una sentencia SQL que captura los atributos de una *View.* |
| **Nombre:** getSequenceDef(dialect: HibernateDialect, schema: String, sequenceName: String): String |
| **Descripción:** devuelve una sentencia SQL que captura los atributos de una S*equence.* |
| **Nombre:** getPrimaryKeyDef(dialect: HibernateDialect, schema: String, primaryKeyName: String): String |
| **Descripción:** devuelve una sentencia SQL que captura los atributos de una P*rimaryKey.* |
| **Nombre:** getCheckDef(dialect: HibernateDialect, schema: String, tableName: String): String |
| **Descripción:** devuelve una sentencia SQL que captura los atributos de un *Check.* |
| **Nombre:** getForeignKeyDef(dialect: HibernateDialect, schema: String, tableName: String, foreignKeyName: String): String |
| **Descripción:** devuelve una sentencia SQL que captura los atributos de una *ForeingKey.* |
| **Nombre:** getExclusionDef(dialect: HibernateDialect, schema: String, tableName: String): String |
| **Descripción:** devuelve una sentencia SQL que captura los atributos de una *Exclusion.* |
| **Nombre:** getFunctionDef(dialect: HibernateDialect, schema: String, functionName: String): String |
| **Descripción:** devuelve una sentencia SQL que captura los atributos de una *Function.* |
| **Nombre:** getTriggerDef(dialect: HibernateDialect, schema: String, tableName: String): String |
| **Descripción:** devuelve una sentencia SQL que captura los atributos de un *Trigger.* |
| **Nombre:** getRuleDef(dialect: HibernateDialect, schema: String, tableName: String): String |
| **Descripción:** devuelve una sentencia SQL que captura los atributos de una *Rule.* |
| **Nombre:** getIndexDef(dialect: HibernateDialect, schema: String, tableName: String, indexName: String): String |

| |
|---|
| **Descripción:** devuelve una sentencia SQL que captura los atributos de una *Index.* |
| **Nombre:** getTableAttributes(dialect: HibernateDialect, schema: String, tableName: String): String |
| **Descripción:** devuelve una sentencia SQL que captura los atributos de una *Table.* |