Análisis de herramientas para la implementación de pruebas funcionales automatizadas a bases de datos relacionales

Juan Pablo Ríos Arenas

Informe Final Modalidad Práctica Empresarial

Asesor
Ronald Akerman Ortiz García, MSc

Universidad de Antioquia
Facultad de Ingeniería
Ingeniería de Sistemas
Medellín, Antioquia, Colombia
2023

Cita	(Ríos Arenas, 2023)
Referencia **Estilo APA 7 (2020)**	Ríos Arenas, J. P. (2023). *Análisis de herramientas para la implementación de pruebas funcionales automatizadas a bases de datos relacionales - 2023* [Trabajo de grado Profesional]. Universidad de Antioquia, Medellín.

Tabla de contenido

Lista de tablas

Lista de figuras

Siglas, acrónimos y abreviaturas

ANSI	American National Standards Institute
BDD	Behavior Driven Development
DSL	Domain Specific Language
IDE	Integrated Development Environment
ISO	International Organization for Standardization
JVM	Java Virtual Machine
POM	Page Object Model
SQL	Structured Query Language

Resumen

Las bases de datos han tomado gran importancia en los últimos años, puesto que son las bodegas donde se guarda la información crucial de nuestras aplicaciones. Por esta razón, en los ciclos de desarrollo de software, es importante incluir el aseguramiento de calidad a las bases de datos para garantizar su correcto funcionamiento. Por su parte, el área de aseguramiento calidad de Sofka no cuenta con los lineamientos que determinen una herramienta para la realización de pruebas funcionales automatizadas a bases de datos relacionales. Es por eso que se decide abarcar una investigación donde se identifiquen herramientas, de uso libre, que permitan y faciliten la realización de pruebas funcionales automatizadas a bases de datos relacionales. Para llevar a cabo el análisis de estas herramientas, se realizó la implementación del marco de trabajo SCRUM, definiendo las fases, objetivos y actividades necesarias para realizar el análisis. Se encontraron diferentes herramientas a partir de los criterios de búsqueda definidos, de entre ellas, se seleccionó Mintleaf como la herramienta con mejor perfil, a partir de la ejecución de unas pruebas de concepto a las herramientas que cumplían con los criterios. Finalmente se realizó la implementación de la herramienta bajo los lineamientos de automatización del área de calidad de Sofka.

Palabras clave: herramientas de automatización de pruebas de software, pruebas funcionales, bases de datos relacionales, aseguramiento de calidad.

Abstract

Databases have become very important nowadays since they are the warehouses where the crucial information of our applications is stored. Therefore, in software development cycles, it is important to include the quality assurance of the databases to guarantee their correct operation. On the other hand, Sofka's quality assurance area does not have the guidelines that determine the use of certain tool to carry out automated functional tests on databases. For this reason, it is decided to begin an investigation to identify open-source tools that allow and facilitate the implementation of automated functional tests on relational databases. The framework SCRUM was implemented to carry out the investigation and analysis of these tools. Different tools were found based on the defined search criteria. Mintleaf was selected as the most complete tool, based on the execution of some concept tests on the tools that met the criteria. Finally, the implementation of the tool was carried out under the automation guidelines of the Sofka quality area.

*Keywords***:** test automation tools, functional testing, relational database, quality assurance.

Introducción

Los sistemas de gestión de bases de datos son una parte fundamental en las infraestructuras digitales hoy en día, puesto que son las bodegas donde se guarda la información crucial de nuestras aplicaciones. Estos sistemas permiten almacenar, modificar y compartir información, y la cantidad de datos que almacenan es cada vez mayor. Por esta razón, se han convertido en un elemento fundamental en algunas áreas, tales como las investigaciones científicas, el desarrollo tecnológico, la medicina y la justicia (Tziatzios, 2019).

En las aplicaciones de software, las bases de datos cumplen un papel fundamental, sin importar si es una aplicación móvil, web o de escritorio, las bases de datos son requeridas en gran parte de las acciones y procesos ejecutados en dichos sistemas. A través de los años la complejidad de las soluciones tecnológicas ha ido evolucionando, especialmente las relacionadas a los productos de software. Entre más complejas se vuelven las interfaces de usuario, la lógica y el almacenamiento de datos se tornan más elaborados y crece la necesidad de implementar bases de datos seguras y robustas.

Por su parte, los ciclos de desarrollo de software que implementan metodologías ágiles se dividen en varias fases, siendo una de éstas, la fase de aseguramiento de calidad. Las pruebas de software permiten a los equipos de desarrollo asegurar la calidad del software a medida que este evoluciona (Gobert, et al., 2022). El uso y manipulación de bases de datos requiere una especial atención en este contexto, sin embargo, en el ámbito local, es común que se omita el aseguramiento de calidad de las bases de datos, lo cual puede repercutir en graves fallos en las aplicaciones.

Ahora bien, Sofka es una compañía de tecnología que brinda experiencias de servicio para la transformación tecnológica, creando soluciones de alto impacto al servicio de las personas y su entorno. Su portafolio incluye la prestación de servicios de aseguramiento de calidad y *testing*, área que tiene como propósito crear una propuesta de servicio, que permita reducir el riesgo de salidas a producción con errores y generar software con un alto nivel de calidad, por medio del uso de buenas prácticas y metodologías tales como el desarrollo guiado por comportamiento y la automatización de pruebas en todos los niveles.

Dentro de los servicios ofrecidos por el área de calidad de Sofka, se comprende la automatización de pruebas de software, servicio que se usa como pilar para generar eficiencia en los procesos de control de calidad, haciendo que las transacciones críticas de negocio se puedan ejecutar de forma más rápida y constante. Para este servicio se cuenta con lineamientos que

recomiendan el uso de ciertas herramientas para realizar pruebas funcionales a aplicaciones y servicios web, sin embargo, para la realización de pruebas funcionales a bases de datos no se cuenta con los lineamientos que sugieran el uso de una herramienta determinada.

Por lo anterior, y con el objetivo de fomentar el aseguramiento de calidad a bases de datos, se decide abarcar una investigación donde se identifiquen herramientas, de uso libre, que permitan y faciliten la realización de pruebas funcionales automatizadas a bases de datos relacionales. Se analizarán diferentes herramientas con el fin de seleccionar aquella con mejor desempeño basándose en determinados criterios, para finalmente realizar la implementación de la herramienta seleccionada, siguiendo las prácticas y lineamientos actuales de las automatizaciones realizadas en el área de calidad de Sofka, con el fin de que estas sirvan como flujo de trabajo para la configuración e implementación de la herramienta.

1. Objetivos

1.1 Objetivo general

Implementar una herramienta para la automatización de pruebas funcionales a bases de datos relacionales, a partir de un análisis de herramientas de uso libre disponibles en el mercado.

1.2 Objetivos específicos

- Analizar herramientas libres disponibles en el mercado para la implementación de pruebas funcionales automatizadas a bases de datos relacionales.

- Seleccionar una herramienta, en base a pruebas de concepto realizadas a las herramientas analizadas.

- Ejecutar pruebas funcionales automatizadas a una base de datos relacional de prueba por medio de la herramienta seleccionada.

2. Marco teórico

A continuación, se listan y se definen los conceptos importantes tratados en este trabajo:

Base de datos relacional: Es un tipo de base de datos que almacena y brinda acceso a conjuntos de datos que están relacionados entre sí. En una base de datos relacional, cada fila de una tabla es un registro único que cuenta con una clave única llamada llave primaria. Las columnas de la tabla contienen atributos de los datos, y cada registro suele tener un valor para cada atributo, lo que facilita establecer las relaciones entre los conjuntos de datos (Oracle, s.f.).

Gestor de bases de datos: Sistema que se usa para almacenar, administrar, consultar y recuperar datos en una base de datos. Los gestores de bases de datos proporcionan una interfaz entre las aplicaciones, los usuarios y la base de datos, así mismo, ofrecen funciones administrativas para la gestión del almacenamiento, acceso y rendimiento de los datos (Oracle, s.f.). Entre los gestores de bases de datos relacionales más utilizados, se encuentran Microsoft SQL Server, Oracle, MySQL, IBM DB2 y PostgreSQL.

Transact-SQL: Es un lenguaje basado en el estándar SQL. SQL es un lenguaje contemplado en los estándares ANSI e ISO y está basado en el modelo relacional, diseñado para consultar y gestionar datos en los sistemas de gestión de bases de datos relacionales (Ben-Gan, I., 2016). Además, Transact-SQL provee algunas extensiones y funcionalidades propias que no están consideradas en el estándar SQL, entre las cuales se destaca la posibilidad de crear y editar procedimientos almacenados.

Procedimientos almacenados: Son rutinas que almacenan código. Estos pueden tener parámetros de entrada y de salida, retornar conjuntos de resultados de datos, modificar datos y aplicar cambios a los esquemas. Traen beneficios como la posibilidad de aplicar lógica, mayor control de seguridad, capacidad de controlar los errores del código y beneficios en el rendimiento de los procesos propios de las bases de datos (Ben-Gan, I., 2016).

Java: Es un lenguaje de programación ampliamente usado para codificar aplicaciones web, desarrollo de juegos, aplicaciones móviles, aplicaciones de escritorio, automatización de pruebas, entre otras, gracias a su facilidad de uso, capacidad multiplataforma y características de seguridad. El lenguaje Java está orientado a objetos, lo cual permite crear programas modulares y de código reutilizable (IBM, 2023).

Intellij: Es un ambiente de desarrollo integrado, el cual está diseñado para la codificación, análisis estático y refactorización de código que puede ser compilado bajo la máquina virtual de Java. IntelliJ ofrece características como asistencia inteligente de codificación, herramientas de refactorización confiables, navegación a través del código fuente, herramientas de desarrollo integradas, entre otros (JetBrains, s.f.).

Gradle: Es una herramienta de compilación automatizada que permite la construcción de casi cualquier tipo de software. Gradle se ejecuta en la JVM y permite la configuración de dependencias, la adición de tareas personalizadas y la realización de otro tipo de configuraciones en la compilación del proyecto (Gradle, s.f.).

Cucumber: Es una herramienta que permite diseñar y ejecutar pruebas bajo las prácticas del desarrollo guiado por comportamiento (BDD). BDD es un proceso de software ágil que busca la colaboración y entendimiento entre desarrolladores, gestores de proyecto y equipo de negocio. Los procesos BDD hacen uso de lenguajes de dominio específico, los cuales facilitan la comunicación y entendimiento de los requerimientos entre los diferentes actores de un proyecto. Cucumber usa Gherkin como su lenguaje de dominio específico (Cucumber, s.f.).

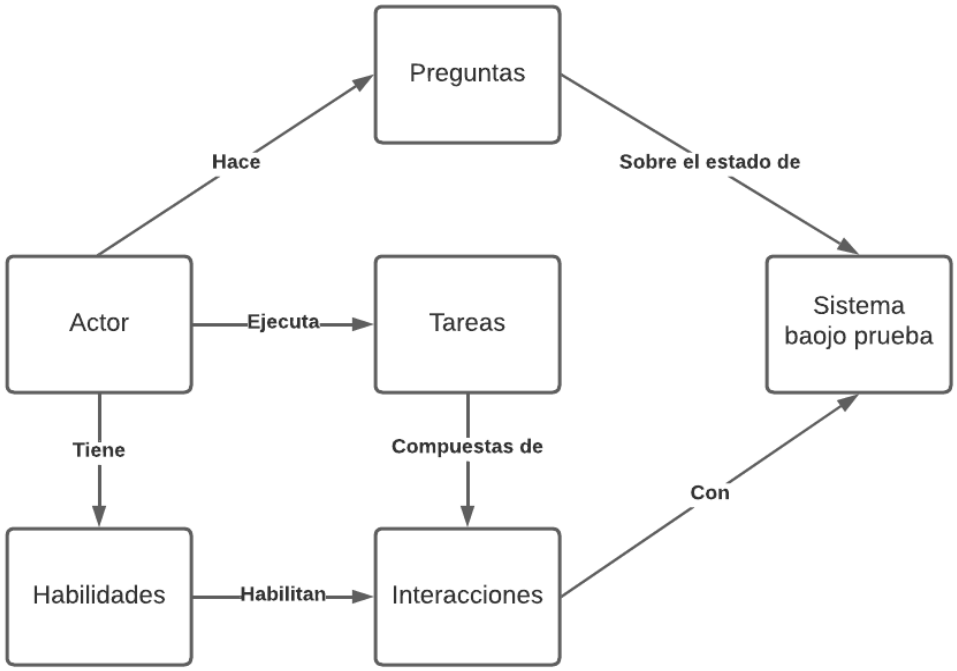
Gherkin: Es un DSL que permite describir comportamientos de negocio sin la necesidad de incluir detalles técnicos de implementación. Gherkin permite desglosar una necesidad de negocio en 3 partes, las cuales son las precondiciones, las acciones y los resultados esperados, para esto, define un conjunto de reglas gramaticales que le dan estructura a un texto plano y que pueden ser interpretadas por la herramienta Cucumber. De esta manera, por medio de Gherkin y Cucumber se tiene la capacidad de diseñar casos de prueba para automatización, ejecutar especificaciones sin ambigüedades y generar documentación activa sobre el comportamiento del sistema bajo prueba (Cucumber, s.f.).

POM: Es un patrón de diseño de automatización, creado e implementado para automatizaciones *frontend*, cuyo objetivo es mejorar la mantenibilidad de los casos de prueba y reducir la duplicidad de código. En este patrón, cada página de una aplicación web es representada en una clase, estas clases son llamadas *Page Object*. Un *Page Object* es una clase que representa los elementos de una página web como una serie de objetos, y encapsula las características y acciones de los objetos en métodos (Leotta, et al., 2020).

Screenplay: Es un patrón de diseño de automatización que fue creado como una versión mejorada del patrón POM, como resultado de la aplicación de los principios SOLID de desarrollo.

El enfoque del patrón Screenplay es la creación de pruebas automatizadas de alta calidad, donde el elemento principal es el actor. El actor tiene habilidades y puede ejecutar tareas, las cuales están compuestas de interacciones con el sistema bajo prueba. Una interacción se entiende como cualquier acción que se puede ejecutar dentro del sistema. El actor tiene la capacidad de hacer preguntas acerca del estado del sistema bajo prueba, por medio de estas preguntas puede hacer las validaciones pertinentes según el enfoque de la prueba (Serenity, s.f.). El diseño de este patrón se puede observar en la Figura 1.

Figura 1. Diagrama de patrón de diseño de automatización Screenplay.



SCRUM: Es un marco de trabajo que permite la solución de trabajos complejos, manteniendo una alta productividad para entregar productos de alta calidad. SCRUM hace parte del desarrollo de software bajo metodologías ágiles. La palabra SCRUM viene del deporte Rugby, la cual es una formación donde se adoptan estrategias por parte de los miembros del equipo, cada uno con un rol específico (Zayat, W., & Senvar, O., 2020).

SOLID: Son principios con un enfoque de diseño de código de software estructurado que garantiza que este sea modular, fácil de mantener, entender, depurar y refactorizar (Educative, s.f.).

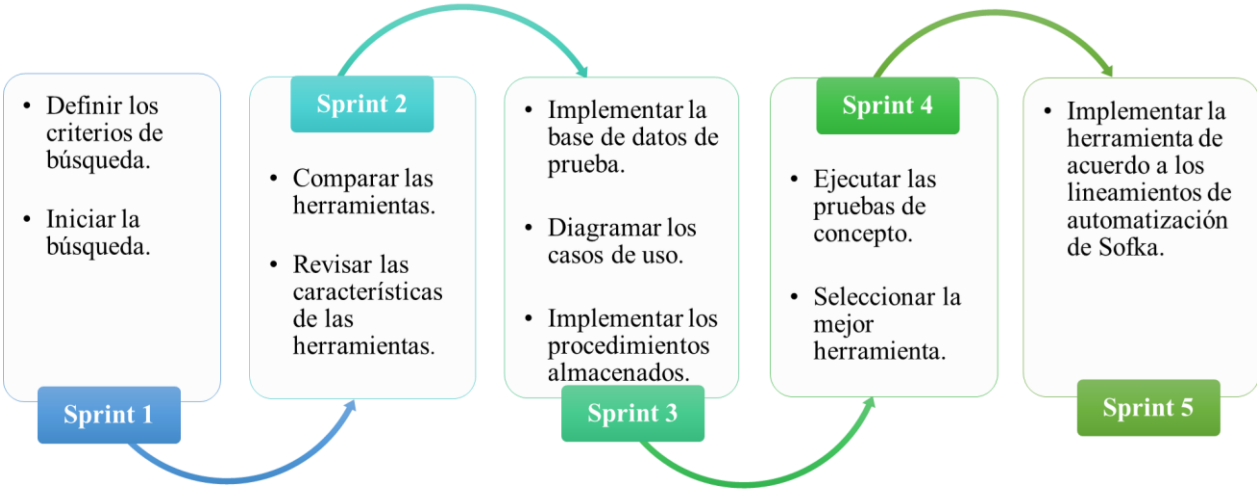
3. Metodología

Para el desarrollo del proyecto se utilizó el marco de trabajo SCRUM, el cual es ampliamente conocido en los desarrollos de software que implementan metodologías ágiles. Se hicieron entregas regulares y parciales de la investigación para ir observando su avance. Bajo el marco trabajo SCRUM, se estructura el desarrollo en ciclos de trabajo llamados *Sprints,* que son iteraciones de uno a cuatro semanas, y se van sucediendo uno detrás de otro. Los *Sprints* son de duración fija, terminan en una fecha específica, aunque no se haya terminado el trabajo, y nunca se alargan.

SCRUM propone tres roles principales: El Dueño de Producto, el Equipo y el Scrum Máster. El Dueño de Producto es el responsable de identificar las necesidades del producto, poniéndolas en una lista priorizada de funcionalidades, decidiendo cuáles deberían ir al principio de la lista para el siguiente Sprint, y priorizando y refinando continuamente la lista. El Equipo construye el producto solicitado por el cliente. El equipo en SCRUM es "multifuncional" y tiene todas las competencias y habilidades necesarias para entregar un producto potencialmente distribuible en cada Sprint. El Scrum Máster ayuda al grupo del producto a aprender y aplicar SCRUM.

Para este trabajo se modificó y se adaptó el marco de trabajo según los requerimientos, por lo cual los roles de dueño de producto y Scrum máster fueron interpretados por el asesor externo y el rol del equipo fue interpretado por el estudiante encargado de la práctica. Los *Sprints* contaron con una duración de tres semanas y en cada uno se hizo una planeación y una entrega. Se omitieron las ceremonias conocidas como "*dailys*", y se reemplazaron por sesiones semanales o agendadas solo cuando había una necesidad. En la Figura 2 se observa el esquema de las fases implementadas a través del marco de trabajo SCRUM y los *Sprints* definidos.

Figura 2. Esquema de la metodología implementada a través de los *Sprints.*



Sprint 1: Se definieron los criterios para la búsqueda de herramientas de pruebas funcionales a bases de datos, para lo cual se realizaron espacios entre el *product owner* y el equipo de trabajo para acordar y justificar la selección de dichos criterios en base a los lineamientos de automatización del área de calidad de Sofka. Una vez identificados los criterios, se inició la búsqueda de las herramientas.

Sprint 2: Se empezó el proceso de selección de herramientas, donde en un principio, se listaron las herramientas gratuitas para pruebas a bases de datos relacionales encontradas y fueron filtradas a partir de los criterios definidos en el sprint anterior. Aquellas herramientas que cumplían con los criterios más importantes fueron seleccionadas.

Sprint 3: Una vez seleccionadas las herramientas, se inició la implementación de la base de datos de prueba en el gestor de bases de datos relacionales SQL Server. En base a las relaciones de esta base de datos, se hizo el diseño y creación de un diagrama de casos de uso, y a partir de este diagrama, se hizo la construcción de los procedimientos almacenados.

Sprint 4: Se planteó la realización de unas pruebas de concepto, con el objetivo de comprobar la viabilidad técnica de las herramientas seleccionadas. Dichas pruebas también se ejecutaron con el objetivo de determinar y seleccionar la herramienta más completa y que ofreciera más características.

Sprint 5: Se crearon dos proyectos base de implementación de la herramienta, uno haciendo uso de Serenity con el patrón de diseño Screenplay y otro haciendo uso del *framework* de automatización de Sofka.

Para la búsqueda de herramientas, se realizó una investigación de carácter exploratorio, por tanto, se implementó una investigación cualitativa, que sugiere métodos de análisis de contenido cualitativo. La recolección de la información se realizó a partir de una búsqueda de revisión de literatura a partir de diferentes fuentes, tales como: bases de datos bibliográficas como Scopus y Google Scholar, además, de una búsqueda de herramientas a partir de libros, artículos, portales oficiales de herramientas y sitios web. El análisis de los datos se realizó a partir de las herramientas seleccionadas, las cuales fueron implementadas y analizadas según los criterios que se establecieron en las etapas iniciales del proyecto.

4. Resultados y Análisis

En la primera etapa del trabajo se definieron los criterios de búsqueda que se tendrán en cuenta para la búsqueda y análisis de las herramientas. Estos criterios fueron escogidos a partir de los lineamientos, prácticas y herramientas utilizadas en las automatizaciones del área de calidad de Sofka. A continuación, se listan los criterios:

Herramienta de libre uso: Es un criterio importante ya que las automatizaciones en sofka se construyen a partir de herramientas de libre uso, con el principal objetivo de reducir costos a los clientes. Solo se hace uso de herramientas de pago cuando la circunstancia así lo requiere.

Integración con lenguaje Java: Java es el principal lenguaje de programación utilizado en las automatizaciones de Sofka, por tanto, la herramienta seleccionada debe contar con una dependencia que permita su uso bajo este lenguaje.

Comunidad y documentación amplia: Estos dos criterios son importantes para la implementación de la herramienta, ya que, al ser una herramienta desconocida y nueva por nosotros, requiere de una curva de aprendizaje y de la adquisición de conocimientos que solo se pueden obtener a partir de la documentación oficial o de tutoriales y/o blogs elaborados por la comunidad. Para la evaluación de estos dos criterios, existe flexibilidad en el cumplimiento, ya que la herramienta evaluada puede cumplir con ambos criterios o cumplir con solo uno de los dos para que sea seleccionable.

Compatibilidad con SQL Server, MySQL y Oracle: Se busca que la herramienta sea compatible con estos gestores de bases de datos ya que son de los más usados en la industria del desarrollo del software, sin embargo, para efectos de este trabajo, la prioridad es el gestor SQL Server, ya que es usado en diferentes proyectos de los clientes de Sofka, por tanto, si la herramienta no es compatible con este gestor, esta herramienta no podrá ser seleccionada.

Herramienta para pruebas funcionales: Se define este criterio ya que se pueden encontrar herramientas con diferentes enfoques de pruebas a bases de datos, tales como pruebas unitarias, pruebas de rendimiento, pruebas de estrés o pruebas de seguridad.

Una vez definidos los criterios, se emprendió la búsqueda de las herramientas disponibles e identificadas a partir de material encontrado en el motor de búsqueda de Google. En la Tabla 1 se observa el comparativo de las herramientas con mayor potencial para ser seleccionadas e implementadas.

Tabla 1. Comparativo de herramientas identificadas.

Herramienta	Uso libre	Integra con lenguaje Java	Comunidad Amplia	Documentación Amplia	Compatible con SQL Server	Compatible con MySQL	Compatible con Oracle	Tipo de prueba
EvoSQL	Sí	Sí	No	No	Sí	Sí	Sí	Generación de datos
JDBCSlim	Sí	Sí	No	No	Sí	Sí	Sí	Funcionales
JDBDT	Sí	Sí	No	Sí	Sí	Sí	Sí	Funcionales
Mintleaf	Sí	Sí	No	Sí	Sí	Sí	Sí	Funcionales
Database Rider	Sí	Sí	No	Sí	Sí	Sí	Sí	Unitarias
utPLSQL	Sí	Sí	No	Sí	No	No	Sí	Automatización de paquetes, funciones, procedimientos.
Unitils	Sí	Sí	No	Sí	Sí	Sí	Sí	Unitarias y de integración
DBUnit	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Unitarias
Irontest	Sí	Sí	No	No	Sí	Sí	Sí	Integración

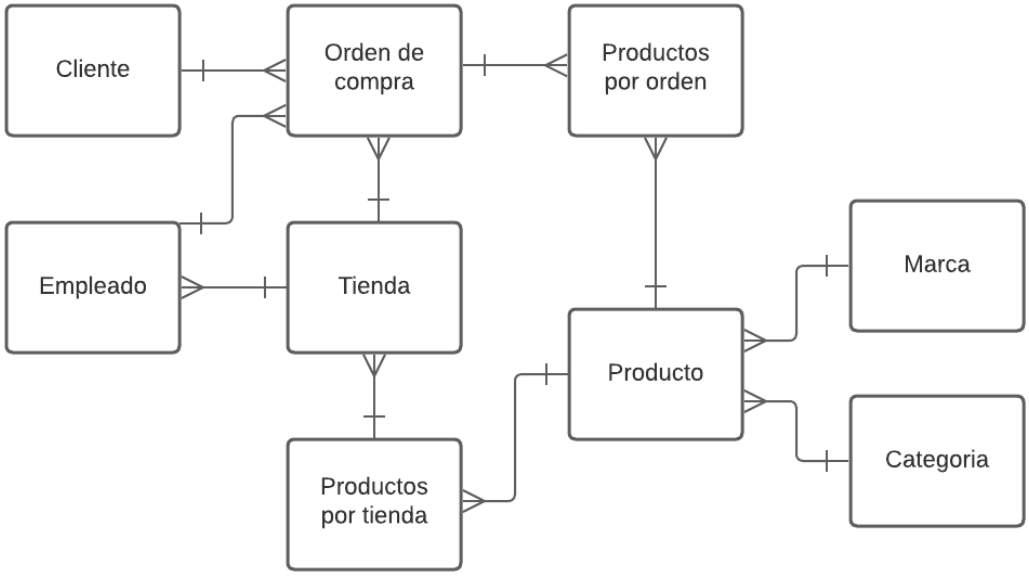
En la búsqueda, se encontró un gran número de herramientas gratuitas para la implementación de pruebas a bases de datos, sin embargo, se fueron descartando todas aquellas que no se pudieran implementar en el lenguaje Java o que no fueran de uso libre. Por lo tanto, estas herramientas no fueron incluidas en la tabla comparativa mostrada anteriormente. De las herramientas que se muestran en la Tabla 1, se puede observar que la única herramienta que cuenta con una comunidad amplia es la herramienta DBUnit, la cual, junto a las herramientas Unitils y DatabaseRides, fueron descartadas ya que su principal objetivo es facilitar la implementación de pruebas unitarias y este no es el objetivo del trabajo de investigación. Las herramientas EvoSQL, JDBCSlim y Irontest también fueron descartadas ya que no cuentan con una comunidad o documentación amplia, lo cual hace casi imposible el aprendizaje e implementación de estas. Así mismo, la herramienta utPSSQL fue descartada ya que no cuenta con soporte al gestor de bases de datos relacionales SQL Server.

Después del análisis realizado para la selección de las herramientas que se pueden implementar para el desarrollo de pruebas funcionales a bases de datos, se encontró que sólo dos herramientas cumplen con los criterios necesarios para el aprendizaje de la herramienta y realización de las pruebas funcionales. Las dos herramientas seleccionadas fueron: JDBDT y

Mintleaf, las cuales, a pesar de no cumplir con el criterio de comunidad amplia, cumplen con el criterio de documentación, que permite el aprendizaje de éstas.

Una vez seleccionadas las herramientas, se dio paso a la implementación de la base de datos de prueba, para lo cual, como se mencionó anteriormente, se hizo uso del gestor de bases de datos relacionales SQL Server. Se decidió implementar la base de datos de un sistema de ventas, ya que es un sistema de común y fácil entendimiento, esto para facilitar la implementación de las pruebas funcionales. En la Figura 3, se puede ver el diagrama entidad-relación de la base de datos implementada.

Figura 3. Diagrama entidad-relación de la base de datos de ventas.



En el diagrama se observa la entidad producto, un producto pertenece a una marca y de una marca puede haber varios productos. De igual forma un producto pertenece a una categoría y de una categoría puede haber varios productos. Un producto puede estar disponible en distintas tiendas y en una tienda puede haber muchos productos, de esta relación de muchos a muchos surge la nueva entidad productos por tienda. A una tienda pertenecen distintos empleados y un empleado sólo puede pertenecer a una tienda. A una orden de compra se le relaciona con un cliente, un empleado y una tienda. De igual forma en una orden de compra se relacionan varios productos y un producto puede estar presente en varias órdenes de compra, de esta relación de muchos a muchos surge la nueva entidad productos por orden.

A partir de la base de datos y sus entidades, se diseñó un diagrama de casos de uso, donde se identificaron dos actores, el cliente y el empleado, y a cada uno se le identificaron algunas de las acciones que pueden ejecutar en el sistema. El cliente puede hacer búsqueda de los productos filtrando por nombre, marca o categoría y puede realizar una compra. El empleado puede generar una orden de compra, añadir nuevas marcas, categorías o productos, consultar la disponibilidad del producto por tienda y actualizar el inventario (Ver Figura 4).

Figura 4. Diagrama de casos de uso.

En base a los casos de uso, se construyeron los procedimientos almacenados que serán utilizados para realizar las pruebas funcionales a la base de datos implementada. Una vez implementado el sistema bajo prueba, se procedió a realizar las pruebas de concepto a las

herramientas seleccionadas, con el fin de verificar su viabilidad técnica y las características que estas ofrecen. Para realizar las pruebas de concepto, se decidió utilizar como objeto de prueba el procedimiento almacenado referente a la búsqueda de producto por marca, esto debido a su baja complejidad. Al implementar estas pruebas, se encontró que ambas herramientas cuentan con funciones propias para lograr la conexión con la base de datos, sin embargo, la herramienta Mintleaf ofrece funciones propias que permiten la ejecución de procedimientos almacenados, mientras que para ejecutar procedimientos almacenados en la herramienta JDBDT se debe hacer por medio de un *query* SQL convencional.

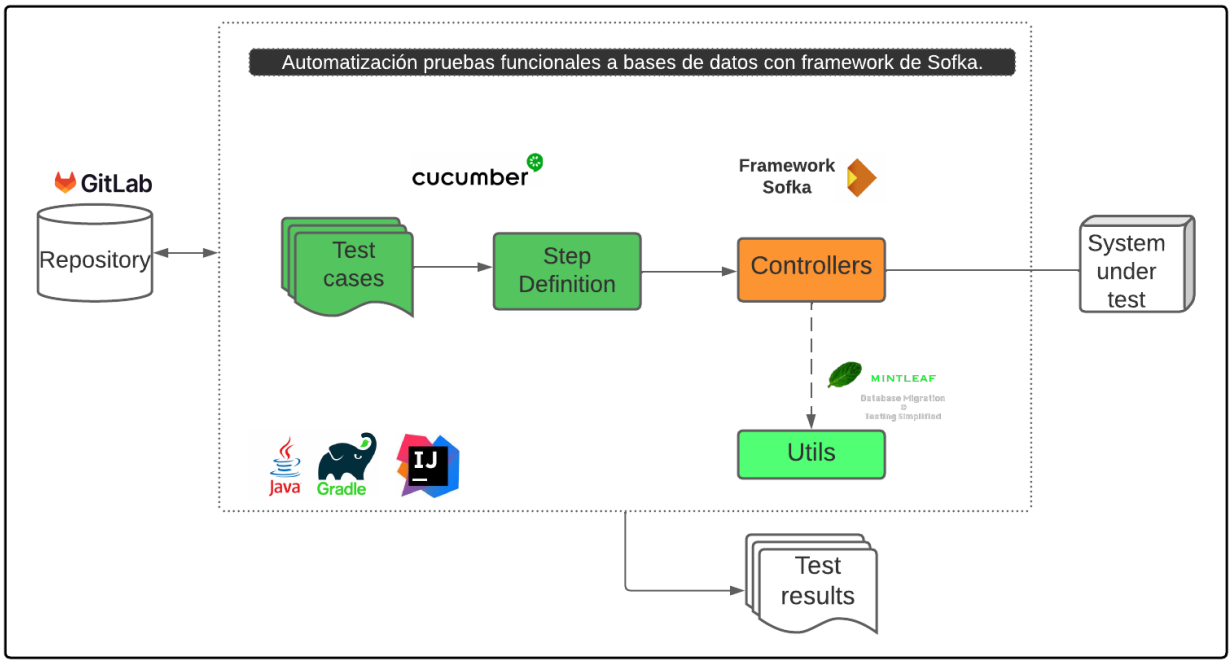
Por medio de las pruebas de concepto, se observó que la herramienta JDBDT es más eficiente en cuanto al número de líneas de código requeridas para su implementación en comparación con la herramienta Mintleaf. Sin embargo, los resultados obtenidos con la herramienta JDBDT no son los esperados, ya que el tipo de variable encargada de guardar el resultado es de tipo *DataSet*, lo que dificulta su manipulación. Esto debido a que la variable guarda el resultado de la consulta en formato XML, y ésta a su vez, no permite la iteración de los resultados para su manipulación y validación, algo que es indispensable al momento de hacer cualquier tipo de prueba. Así mismo, este tipo de variable no es tan conocida en comparación con el tipo de variable encargada de guardar los resultados en la herramienta Mintleaf, la cual es de tipo *ResulSet*. Este tipo de formato almacena los resultados de la consulta en una lista iterable de objetos, lo cual facilita la manipulación y la validación de los resultados obtenidos al ejecutar el procedimiento almacenado.

Agregando a lo anterior, se hizo una inspección profunda a la documentación disponible de ambas herramientas, donde se encontró que la herramienta Mintleaf cuenta con más características, tales como la migración de bases de datos, importación y exportación de datos a partir de archivos CSV, la creación de contextos de conexión donde se pueden hacer transacciones sin afectar los datos reales de la base de datos, la comparación entre diferentes fuentes de datos y el soporte a los gestores de bases de datos relacionales SQL Server, Oracle, H2 y MySQL. Es por eso qué, se seleccionó la herramienta Mintleaf como la herramienta gratuita más completa para implementar pruebas funcionales automatizadas a bases de datos relacionales a partir de la ejecución de procedimientos almacenados.

Como último paso del trabajo y ya con la herramienta seleccionada, se procedió a hacer la implementación de esta bajo los lineamientos de automatización del área de calidad de Sofka, por

lo cual se hicieron dos implementaciones, una haciendo uso del *framework* de automatización de Sofka y otra haciendo uso de Serenity con el patrón de Screenplay. Ambas implementaciones hacen uso del lenguaje de programación Java, la programación orientada a objetos, el IDE IntelliJ, el gestor de dependencias Gradle y la herramienta Cucumber junto al DSL Gherkin.

Figura 5. Implementación de la herramienta con *framework* de automatización de Sofka.



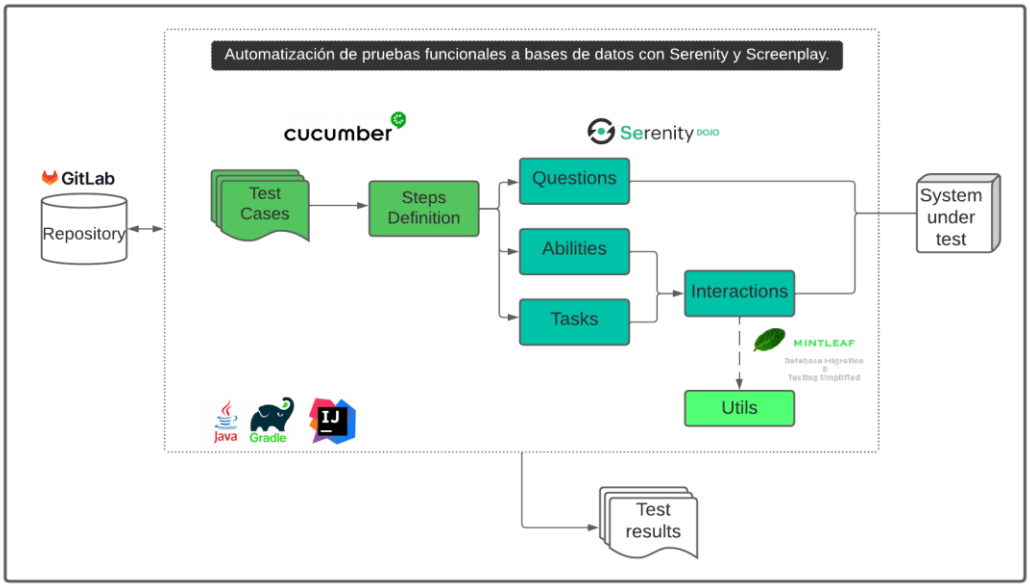
Como se observa en la Figura 5, el flujo inicia con la definición de los casos de prueba (*Test Cases*), en este módulo se almacenan los archivos que contienen los pasos que describen el comportamiento del sistema, escritos en DSL Gherkin, el cual, está integrado en la herramienta Cucumber. Estos archivos son conocidos con el nombre de *features*, su extensión es de tipo .feature y deben cumplir con la sintaxis definida en DSL Gherkin, ya que, a partir de esta sintaxis, Cucumber convierte los pasos que describen el comportamiento del sistema en métodos de una clase Java.

Estas clases creadas por Cucumber a partir de los archivos *feature*, son conocidas como *Steps Definition* y son almacenadas en el módulo del mismo nombre. Dentro de cada una de estas clases, se crea un método para cada uno de los pasos definidos en DSL Gherkin, a partir del archivo *feature* al que están relacionadas. La función principal de estos métodos es hacer un llamado a las

clases contenidas en el módulo *Controllers*. Las clases contenidas en el módulo *Controllers*, son definidas por el encargado de realizar el proyecto de automatización. El objetivo de las clases del módulo *Controllers*, es contener la lógica y las reglas de negocio del sistema bajo prueba, al mismo tiempo que interactúan con éste. Para lograr dicho objetivo, se apoyan de las clases contenidas en el módulo *Utils.*

En el módulo *Utils* se almacenan las clases que generan las utilidades requeridas por la prueba a ejecutar, como por ejemplo, conectar a una base de datos, leer archivos a partir de una ruta, crear archivos, generar datos de prueba, entre otros. Para este caso, se realizó la implementación de la herramienta Mintleaf por medio de una clase creada en el módulo *Utils*, esta clase contiene los métodos para la conexión, ejecución de procedimientos almacenados y consultas a la base de datos. En cada ejecución de un proyecto de pruebas automatizadas, se interactúa con el sistema bajo prueba (*System under test*) y se genera un reporte con los resultados de las pruebas (*Test results*). El código necesario para desarrollar y ejecutar el proyecto de automatización debe ser versionado y guardado en un repositorio (*Repository*).

Figura 6. Implementación de la herramienta con Serenity.



En la Figura 6 se observan los módulos *Test Cases*, *Steps Definition* y *Utils*, los cuales tienen el mismo comportamiento y definición que en la implementación con el *framework* de Sofka, mostrado anteriormente en la Figura 5. La diferencia radica en las clases que son llamadas

en los métodos del módulo *Steps Definition*, ya que no se llaman desde el módulo *Controllers*, sino que son llamadas desde los módulos *Questions, Abilities* y *Tasks*, que son incorporados en la implementación con Serenity. En el módulo *Questions*, se almacenan las clases que interactúan con el sistema bajo prueba y preguntan sobre su estado actual, con el objetivo de hacer las comparaciones entre los resultados esperados y los resultados obtenidos de la prueba. El módulo *Abilities* contiene las clases que proporcionan habilidades como navegar por la web, consumir servicios o interactuar con bases de datos. En el módulo *Tasks*, se almacenan las clases que contienen las tareas que se ejecutan en el sistema bajo prueba, como por ejemplo autenticarse en una aplicación web, diligenciar un formulario, insertar valores en una base de datos, entre otras.

Tanto las clases del módulo *Abilities* como las clases del módulo *Tasks*, hacen uso de las clases almacenadas en el módulo *Interactions*. Las clases del módulo *Interactions* contienen las diferentes interacciones que se pueden tener con el sistema bajo prueba, ejemplo de estas interacciones son: dar clic a un elemento, ingresar texto a un elemento, consumir un servicio REST, hacer una consulta a la base de datos, etc. Para lograr estas interacciones, las clases del módulo *Interactions* se apoyan de las clases contenidas en el módulo *Utils*. Al igual que la implementación con el *framework* de Sofka, las ejecuciones de la implementación con Serenity generan un reporte con los resultados de las pruebas y el código desarrollado debe ser versionado y guardado en un repositorio.

Ambas implementaciones quedaron almacenadas en repositorios de acceso público, con el objetivo de que sirvan como ejemplo de implementación, tanto de la herramienta como de las pruebas funcionales automatizadas a bases de datos relacionales. La implementación con el *framework* de Sofka se recomienda usar en proyectos pequeños y de baja complejidad, dada su facilidad de uso y entendimiento. Por su parte la implementación con Serenity y patrón de diseño Screenplay, se recomienda en proyectos grandes y de alta complejidad, ya que la estructuración de los módulos y las clases permiten el cumplimiento de los principios SOLID de programación. El cumplimiento de estos principios garantiza que el desarrollo del código se haga de manera que éste sea fácil de mantener, entender, depurar y refactorizar a medida que el proyecto crece.

Dentro del área de aseguramiento de calidad de Sofka no se cuenta con una herramienta para la automatización de pruebas funcionales a bases de datos relacionales, además, estas pruebas se realizan de manera manual a estos sistemas. Esto impacta directamente en el tiempo requerido para la ejecución de estas pruebas, así como también, las pruebas pueden verse afectadas debido a

los errores que se pueden presentar dado el factor humano. Por esto, se resalta la importancia de la inclusión de la herramienta Mintleaf, que permite llevar a cabo una metodología para la implementación de proyectos de automatización que incluyan la ejecución de pruebas funcionales automatizadas a bases de datos, para la reducción de tiempos y costos, generando valor agregado a los clientes finales de Sofka.

5. Conclusiones

- El uso del marco de trabajo SCRUM tuvo un impacto positivo en este proyecto, ya que permitió definir de manera clara las diferentes fases del proyecto, los objetivos y actividades en cada una de éstas, así como también, la duración de las mismas. Esto permitió tener una visión clara de los objetivos y del estado del proyecto en cada una de sus fases.

- A partir de la búsqueda de revisión de literatura, se observó que en el mercado se cuenta con pocas herramientas de uso libre para implementar pruebas funcionales automatizadas a bases de datos relacionales haciendo uso del lenguaje java, en comparación con la cantidad de herramientas de uso libre disponibles para otros tipo de pruebas, como por ejemplo herramientas para pruebas unitarias a bases de datos, pruebas de rendimiento a bases de datos, pruebas automatizadas de servicios web y pruebas automatizadas de aplicaciones web.

- De las herramientas identificadas para la implementación de pruebas funcionales automatizadas a bases de datos relacionales, se puede decir que no cuentan con una comunidad amplia y cuentan con poca documentación, lo cual dificulta el aprendizaje e implementación de estas.

- La ejecución de las pruebas de concepto a las herramientas seleccionadas fue un paso fundamental, ya que permitió validar la viabilidad técnica de las herramientas, y al mismo tiempo, los resultados de estas pruebas sirvieron como criterio de selección de la herramienta que contará con las mejores funciones y características.

- La implementación de la herramienta bajo los lineamientos de automatización de Sofka, generará un gran impacto tecnológico, ya que puede ser utilizada a futuro por la compañía para afrontar retos similares de manera eficaz y automatizar procesos de pruebas que son realizados típicamente de manera manual, reduciendo tiempos y costos a los clientes finales de la compañía.

6. Referencias

Ben-Gan, I. (2016). T-Sql Fundamentals. 3rd Edition. Microsoft Press.

Cucumber. (s.f.). What is Cucumber?. Recuperado de: https://cucumber.io/docs/guides/overview/

DesRivieres, J., & Wiegand, J. (2004). Eclipse: A platform for integrating development tools. IBM Systems Journal, 43(2), 371–383. https://doi.org/10.1147/sj.432.0371

Educative. (s.f). What are the SOLID principles in Java?. Recuperado de: https://www.educative.io/answers/what-are-the-solid-principles-in-java

Gobert, M., Nagy, C., Rocha, H., Demeyer, S. & Cleve, A. (2022). Best practices of testing database manipulation code. Information Systems, 111, 102105. https://doi.org/10.1016/j.is.2022.102105

Gradle. (s.f.). What is Gradle?. Recuperado de: https://docs.gradle.org/current/userguide/what_is_gradle.html

IBM. (2023). Advantages of Java. Recuperado de: https://www.ibm.com/docs/en/aix/7.1?topic=monitoring-advantages-java

JetBrains. (s.f). What is IntelliJ. Recuperado de: https://www.jetbrains.com/idea/features/

Leotta, M., Biagiola, M., Ricca, F., Ceccato, M., & Tonella, P. (2020). A Family of Experiments to Assess the Impact of Page Object Pattern in Web Test Suite Development. 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST). https://doi.org/10.1109/ICST46399.2020.00035

Oracle. (s.f.). What is a Relational Database (RDBMS)?. Recuperado de: https://www.oracle.com/database/what-is-a-relational-database/

Santamaría, J., & Hernández, J. (2016). Microsoft SQL Server. SQL SER vs MY SQL, 1-6.

Serenity. (s.f.). The Screenplay Pattern. Recuperado de: https://serenity-js.org/handbook/design/Screenplay-pattern.html

Tziatzios, D. (2019). Model-based testing for SQL databases.

Zayat, W., & Senvar, O. (2020). Framework Study for Agile Software Development Via SCRUM and Kanban. International Journal of Innovation and Technology Management. https://doi.org/10.1142/s0219877020300025