UNIVERSIDAD PRIVADA LÍDER PERUANA

ESCUELA PROFESIONAL DE INGENIERIA DE SISTEMAS E INFORMÁTICA

Trabajo de investigación

HERRAMIENTA CASE GXC PARA GENERAR CÓDIGO BAJO LA ARQUITECTURA DE PROGRAMACIÓN EN 3 CAPAS

PARA OBTENER GRADO ACADÉMICO DE BACHILLER EN INGENIERÍA DE SISTEMAS E INFORMÁTICA

Autor

Juan Moises Rojas Torres

Asesor

Mg. Hugo David Calderon Vilca

Santa Ana, La Convención, Cusco

2020

Línea de investigación:

Sistemas y tecnologías de la información

Título:

HERRAMIENTA CASE GXC PARA GENERAR CÓDIGO BAJO LA ARQUITECTURA DE PROGRAMACIÓN EN 3 CAPAS

RESUMEN

El problema que se muestra es que existen herramientas que generan código automáticamente, se basan en modelos, las herramientas no tienen adaptabilidad por parte de los programadores. La investigación está enfocada en generar código para la arquitectura de 3 capas. La herramienta genera código a partir del esquema de base de datos, que debe ser ingresado a la herramienta a través de la interfaz gráfica. Entre los objetivos de la herramienta esta la reducción el tiempo de programación. La presente investigación propone ingresar un esquema de base de datos y a partir de este genera el código para el lenguaje SQL Server y CSharp, para diferentes tipos de script. La herramienta está escrita en el lenguaje CSharp, utiliza un archivo .xls para importar o exportar el esquema de la base de datos. Para la obtención de los resultados se utilizó 5 esquemas de base de datos, obteniendo como resultado un código que compilo sin errores. Realizando el análisis de tiempo se obtuvo que la propuesta genera 9,023.78 líneas de código por segundo, también se obtuvo que para generar todos los script para las 5 estructuras de base de datos solo se utiliza 1472 milisegundos. También se obtuvo en total 13283 líneas de código.

Palabras clave: Ingeniería de Software, Herramientas CASE, Optimización de Tiempo, Tecnología CASE, Generación de código.

ABSTRACT

The problem shown is that there were tools that generate code automatically, they are based on models, the tools do not have adaptability on the part of the programmers. The research is focused on generating code for the 3-tier architecture. The tool generates code from the database schema, which must be entered into the tool through the graphical interface. Among the objectives of the tool is the reduction of programming time. The present investigation proposes to enter a database schema and from this generates the code for the SQL Server and CSharp language, for different types of script. The tool is written in the CSharp language, it uses an .xls file to import or export the schema from the database. To obtain the results, 5 database schemas were used, obtaining as a result a code that I compiled without errors. Carrying out the time analysis, it was obtained that the proposal generates 9,023.78 lines of code per second, it was also obtained that to generate all the scripts for the 5 database structures only 1472 milliseconds are used. A total of 13283 lines of code were also obtained.

Keywords: Software Engineering, CASE Tools, Time Optimization, CASE Technology, Code generation.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

Índice de Tablas

CAPÍTULO I. INTRODUCCIÓN

1.1. Antecedentes del problema

En la actualidad existen diferentes formas de desarrollar software, en la parte de diseño del proyecto se utilizan muchas metodologías, existen herramientas para ayudar en el desarrollo de software y la mayoría de estas herramientas utilizan modelos pre-establecidos. Al desarrollar software se tiene como uno de los problemas el tiempo. Hay diferentes investigaciones realizadas acerca de generación de código algunas se centran en el script de bases de datos, otras en las diferentes fases de desarrollo de software, etc.

(Wibowo, Hendradjaya, & Widyani, 2015) en su artículo publicado indica que las pruebas de software son un proceso importante en el ciclo de vida del desarrollo de programas y es uno de los principales procesos que toman mucho tiempo es el desarrollo de código de prueba. Frente a ello proponen una generación de código de prueba unitaria para acelerar el proceso y evitar la repetición. Desarrollaron el generador utilizando el lenguaje de programación LUA, que es un lenguaje de scripting rápido, liviano e integrable.

El problema principal de la investigación Evaluation of FED-CASE - A Tool to Convert Class Diagram into Structural Coding que motivo a escribir este artículo es la elaboración de un enfoque de desarrollo basado en modelos. Los analistas y desarrolladores de la industria de TI tienen que dedicar mucho tiempo y esfuerzo en hacer manual de ingeniería directa e inversa. Ellos introducen una herramienta CASE y la evalúan utilizando un criterio y un número de medidas para sugerir esta herramienta para el desarrollo automatizado basado en modelos. Utilizan el modelado UML porque consideran que es un estándar en el mundo de ingeniería y por ello a partir de un diagrama de clases generan el código (Sadaf, Athar, & Azam, 2016).

En el artículo Application of Architecture Implementation Patterns by incremental code generation dentro del problema indica que se han establecido arquitecturas de referencia en muchos dominios de ingeniería de software que permiten la reutilización del conocimiento experto de diseño, directrices y mucho más. Sin embargo, las arquitecturas también pueden conducir a un código repetitivo y una sobrecarga de implementación. Este trabajo aborda este inconveniente mediante la introducción de patrones de implementación

de arquitectura (AIM Patterns), que se aplican automáticamente al código de la aplicación mediante un enfoque de generación incremental (Brunnlieb & Poetzsch-Heffter, 2016).

Recursive Modeling for Completed Code Generation indica que la generación de código totalmente automatizada a partir de las partes de comportamiento sigue siendo difícil, si es que funciona, restringida a áreas de aplicación específicas que utilizan un lenguaje específico de dominio, DSL. Proponen un enfoque para modelar las partes de comportamiento de un sistema e integrarlas en los modelos estructurales. La idea subyacente son los refinamientos recursivos de elementos de actividad en un diagrama de actividad (Sulistyo & Prinz, 2009).

El problema que se puede evidenciar es que existen herramientas que generan código automáticamente basada en modelos, pero no hay una herramienta que genere código para la arquitectura de programación a 3 capas, y sea configurable por los programadores. Se propone una herramienta CASE capaz de generar código para la arquitectura de programación de 3 capas, a partir de la configuración del programador. La generación de código lo realizamos almacenando la sintaxis del lenguaje y los parámetros de las tablas.

Los ingenieros de sistemas, analistas de sistemas, desarrolladores de software, etc. Se ocupan en resolver procesos para las organizaciones ya sea con sistemas de información, aplicaciones informáticas u otra solución informática. Muchas de estas soluciones terminan en grandes desarrollos de sistemas o puede ser una aplicación informática. Con la herramienta CASE GXC, pretendemos ayudar reducir el tiempo de programación con la generación de código de algunos procesos, configurados por el programador que va a utilizar la herramienta.

1.2. Objetivos de la investigación

1.2.1. Objetivo general

- Diseñar la herramienta CASE GXC para generar código bajo la arquitectura de programación en 3 capas.

1.2.2. Objetivos específicos

- Analizar las formas de generación de código para la arquitectura de 3 capas.
- Diseñar el modelo de generación de código para la arquitectura de 3 capas.
- Validar el modelo de generación de código para la arquitectura de 3 capas.

1.3. Justificación de la investigación

Los ingenieros de sistemas, analistas de sistemas, desarrolladores de software, etc. Se ocupan en resolver procesos para las organizaciones ya sea con sistemas de información, aplicaciones informáticas u otra solución informática. Muchas de estas soluciones terminan en grandes desarrollos de sistemas o puede ser una aplicación informática. Con la herramienta CASE GXC, se puede ayudar reducir el tiempo de programación con la generación de código de algunos procesos, configurados por el programador que va a utilizar la herramienta.

En la actualidad existen herramientas CASE que se ocupan de ayudar en las diferentes fases de desarrollo de software. Pero no son muy utilizadas porque muchas de las herramientas CASE no están modeladas de acuerdo al gusto de los programadores de software, porque ya traen modelos pre definidos y estos no se pueden configurar, para poder generar el código. La herramienta GXC será configurable, esto quiere decir que los programadores pueden configurar la herramienta.

La herramienta GXC ayuda a reducir el tiempo de desarrollo de sistemas de información, con modelos de código establecidos por cada programador, y generar código para las diferentes capas de ser el caso (capa de datos, capa lógica de negocios, capa de presentación) y también interviene en el entorno de la construcción de la base de datos y procedimientos almacenados.

Otra finalidad de la herramienta CASE GXC esta aumentar la eficiencia y eficacia de los procesos, reducir los recursos necesarios y reducir los defectos en el desarrollo de software. Las herramientas CASE contribuyen con la economía en el desarrollo de software.

Según el libro Software Quality: Concepts and Practice indica que se presentaron resultados alentadores para AutoFix (Pei et al., 2011, 2015). En un experimento de 2015, AutoFix corrigió con éxito el 42% de las fallas del software (86 de 204 fallas), donde en la mayoría de los casos (51 de 86 casos) la calidad de la corrección fue comparable a una corrección realizada por un programador (Galin, 2018). Según esos resultados se puede evidenciar que una herramienta CASE puede corregir fallas del software.

1.4. Propósito de la investigación

La herramienta CASE GXC debe generar código fuente para la arquitectura de programación de 3 capas, para los lenguajes de CSharp y SQL Server. Así mismo debe convertirse en una herramienta útil para los desarrolladores de software, reduciendo el fallo y también mejorando el tiempo en el desarrollo de software. Para lograr lo anterior evaluamos las formas de generación de código, para realizar una óptima generación de código.

1.5. Alcance y limitaciones de la investigación

El alcance de la investigación se considera la generación de código fuente para el lenguaje CSharp y SQL Server. La limitante de la investigación es que no es una herramienta completa, solo se limita a generar código para SQL Server, para generación de script de la base de datos, procedimientos almacenados insertar, eliminar, listar, modificar. Para CSharp genera, parámetros de las clases, métodos insertar, eliminar, listar, modificar, auto completado.

CAPITULO II MARCO TEÓRICO

2.1. Antecedentes de la investigación

Junno Tantra Pratama Wibowo, Bayu Hendradjaya y Yani Widyani en su artículo publicado en la "2015 International Conference on Data and Software Engineering" de nombre Unit Test Code Generator for Lua Programming Language proponen como producto final, un generador de prueba de unidad Lua (LUTG), integrado a uno de los IDE de Lua más populares, ZeroBrane Studio, como Plugin para conectar sin problemas el proceso de codificación y prueba. El generador de código puede generar código de prueba de unidad, guardar datos de casos de prueba en formato de archivo Lua y XML, y generar los datos de prueba automáticamente usando una técnica basada en búsqueda, algoritmo genético, para lograr criterios de prueba de cobertura de rama completa (Wibowo, Hendradjaya, & Widyani, 2015).

(Brunnlieb & Poetzsch-Heffter, 2016) En una de sus conclusiones indican que en resumen hay dos condiciones previas principales para utilizar el enfoque presentado. Primero, se debe describir una arquitectura de referencia en un estilo basado en patrones para hacer que la extracción de patrones AIM sea lo suficientemente fácil. Esto incluye la existencia de pautas sobre el nivel de código, de manera óptima, fragmentos de código de muestra que indican las mejores prácticas. En segundo lugar, la arquitectura de referencia debe prescribir suficientes convenciones de nomenclatura, así como la pila de tecnología para permitir que la generación incremental entreteja automáticamente el código generado en el código existente.

(Sadaf, Athar, & Azam, 2016) En su investigación, concluyen que su estudio demuestra claramente que la automatización del desarrollo basado en modelos es más eficiente que el desarrollo simple. El modelado UML se ha convertido en un estándar en el mundo de la ingeniería y para que su proyecto sea un éxito, este enfoque se está volviendo esencial. La automatización del proceso nos ayuda mucho en términos de escribir código libre de errores en menos tiempo y esfuerzo. La participación humana se minimiza, lo que lleva a la eliminación de casi todos los errores causados por muchos factores, como menos

conocimiento, poco interés y fatiga. La eficiencia y la precisión son dos causas principales que empujan a los lectores a utilizar este enfoque.

Recursive Modeling for Completed Code Generation, investigación desarrollada por Selo Sulistyo y Andreas Prinz, concluye que presentan un enfoque combinado para generar un código completo a partir de los modelos estructurales y de comportamiento. Usando el enfoque de arriba hacia abajo aplicando refinamientos recursivos de elementos de actividad en un diagrama de actividad, y usan el enfoque de abajo hacia arriba desarrollando una herramienta para la creación de ejecutores de actividad y para el mecanismo de mapeo del nivel más bajo de actividades en los ejecutores. En su contexto, un modelo que no incluye ejecutores es un modelo PIM. Con respecto a la generación de código, consideran que el modelado es solo el cambio de enfoque de lenguajes orientados a la implementación a lenguajes orientados a gráficos (modelos). Por lo tanto, para poder generar código, los lenguajes de modelado necesitan abstraer operaciones aritméticas básicas, operaciones lógicas básicas y manipulaciones de cadenas, como ejecutores de actividades básicas (Sulistyo & Prinz, 2009).

The Metric For Automatic Code Generation presentado en la 3rd International Conference on Mechatronics and Intelligent Robotics(ICMIR-2019) por Zhen Li, Ying Jiang, Xiao Jiang Zhang y Hai Yan Xu, consideran que existen muchas investigaciones en la métrica acerca de código fuente estático y que existen pocos acerca de generación automática de código. Proponen el modelo métrico que incluye seis características métricas. Utilizan tres herramientas automáticas de generación de código para medir el código fuente experimental. Dentro de sus conclusiones señalan que los resultados de medición del análisis pueden mostrar que el modelo métrico de generación automática de código es razonable, lo que puede ser útil para medir la generación automática de código, Dentro de los procedimientos para obtener resultados se puede decir que para verificar su modelo métrico utilizan CodeGeneration, java-codetool y aiXcoder para generar los objetos experimentales. Utilizan generación de código basada en plantillas y los programadores predefinen algunos elementos de forma experimental. También utilizan generación de código basada en el aprendizaje automático. Luego el código fuente generado se mide automáticamente. La calidad del código fuente generado por CodeGeneration y java-codetool es aproximadamente la misma. Las herramientas basadas en plantillas generan menos líneas de

código que las basadas en aprendizaje automático. Las herramientas de aprendizaje automático emplean un tiempo y espacio mayor que las herramientas basadas en plantilla, esto con respecto a la cantidad (Li, Jiang, Zhang, & Xu, 2020).

(Han, Zhao, Yu, Shi, & Huang, 2019) Analizan que debido a la gran similitud entre módulos en la arquitectura B/S, a menudo se realiza un trabajo repetitivo de copiar y pegar en el desarrollo. Proponen un diseño e implementación del sistema de generación automática de código de arquitectura B / S. Presentan las tecnologías relacionadas utilizadas su sistema y elaboran la función de cada módulo de su sistema. Definen un formato XML para describir entidades de información, implementan módulos generales y módulos de funciones del sistema. En una de sus conclusiones manifiestan que su sistema de generación automática de código puede mejorar la eficiencia del desarrollo. El proceso que siguen es el modelo de plantilla lee la información del atributo y reemplaza las etiquetas de la variable y luego genera el código objetivo final. Con la práctica han demostrado que el usuario puede generar rápidamente aplicaciones de arquitectura B/S mediante el archivo de descripción de entidad que proporciona el usuario utilizando el sistema de generación automática de código.

Marcos Antonio Possatto y Daniel Lucrédio en su investigación Automatically propagating changes from reference implementations to code generation templates proponen un mecanismo para detectar y propagar semiautomáticamente los cambios del código de referencia a las plantillas, manteniéndolos sincronizados con menos esfuerzo. En una de sus conclusiones manifiestan que todavía hay un margen de mejora, sus resultados indican que la automatización se puede utilizar para reducir el esfuerzo y el costo en el mantenimiento y la evolución de una generación de código basada en plantillas de infraestructura. Como resultados observaron que el mecanismo desarrollado puede conducir a una reducción del 50% en el esfuerzo necesario para realizar la plantilla de mantenimiento/evolución, en comparación con un enfoque manual. Así mismo también observaron que ese efecto depende de la naturaleza de la tarea de evolución/mantenimiento, ya que para una de las tareas no había una ventaja observable en el uso del mecanismo (Possatto & Lucrédio, 2015).

En la investigación A novel syntax-aware automatic graphics code generation with attention-based deep neural network proponen un modelo llamado HGui2Code que integra características GUI con atención visual (extraída por CNN) con características semánticas

con capacidad DSL (extraída por LSTM). Adicionalmente también proponen SGui2Code, un modelo novedoso que utiliza una red ON-LSTM para generar código DSL que sea correcto en sintaxis. Dentro de sus conclusiones manifiestan lo siguiente la estructura atómica de su programa en sí tiene un significado inherente, y la entrada del modelo no es un espacio de dominio continuo, el marco de generación de código GUI automático original no puede hacer un uso completo de la información oculta en GUI y Código DSL, por lo tanto, proponen el modelo HGui2Code utilizando una red neuronal profunda basada en la atención para alinear el código DSL con los píxeles GUI correspondientes mediante un mecanismo de atención híbrido. Dentro de sus resultados y análisis de modelo se puede ver que para la generalidad de sus modelos, que códigos generan y que tan cerca está el código real, analizan en detalle los resultados del experimento de sus dos modelos. Al comparar la tasa de error de las líneas de base incluido los modelos pix2pix y ABHD, demostraron que sus modelos son más efectivos (Pang, y otros, 2020).

(She & Zheng, 2020) Proponen un método automático de generación de código de página basado en la plantilla de Excel y la tecnología POI dentro del cual tienen dos ventajas. Dentro de las conclusiones se tiene que los desarrolladores de software no necesitan prestar atención a los detalles de la lógica empresarial. Para páginas de contenido complejas, los usuarios pueden diseñar / importar directamente un documento relativamente excelente y describir los contenidos de Excel basados en diferentes marcos web (como easyUI, BootStrap, etc.). En su aplicación práctica demuestran que su método realiza la generación rápida y automática de interfaces. Con esto logran la reutilización de código, mejoran la eficiencia de desarrollo y reducen costos de desarrollo.

Jean Pierre Alfonso Hoyos y Felipe Restrepo Calle en la investigación Automatic Source Code Generation for Web-based Process-oriented Information Systems, proponen un enfoque para generar automáticamente un prototipo de aplicación web que ejecute procesos comerciales utilizando una especificación de lenguaje natural restringida. En el cual concluyen que tienen un método de creación rápida de prototipos para aplicaciones web, que ejecutan procesos comerciales utilizando una especificación de lenguaje natural restringido, como resultado obtuvieron un código fuente listo para ser parte del producto final y esto reduce los problemas asociados con las etapas de obtención de requisitos y diseño (Alfonso & Restrepo, 2017).

La proposición de la investigación ISML-MDE: A Practical Experience of Implementing a Model Driven Environment in a Software Development Organization es la experiencia práctica de desarrollar ISML-MDE, que viene a ser un entorno basado en modelos que comprende tres componentes principales que son: un lenguaje de modelado para aplicaciones empresariales (ISML), un marco de generación de código (ISML-GEN) y LionWizard. Concluyeron que presentaron la experiencia práctica de la creación de ISML-MDE en un entorno basado en modelos que fueron para abstraer y automatizar el desarrollo de aplicaciones empresariales. Como resultados obtuvieron que el nivel de abstracción en ISML es suficientemente alto, para ocultar los detalles de implementación, pero bajo para especificar la mayoría de una aplicación empresarial. Los ingenieros aprenden gradualmente el idioma y lo adoptan fácilmente, por las facilidades de modelado parcial proporcionadas por la palabra clave nativa. La generación y transformación de código modular automatiza la mayor parte de la implementación de la aplicación y también contribuye al olvido de los detalles de la plataforma de destino (Franky, y otros, 2016).

JDriver: Automatic Driver Class Generation for AFL-Based Java Fuzzing Tools es una investigacion que propone JDriver que es un marco de generación automática de clases de controladores para herramientas de fuzzing basadas en AFL, que puede generar código de controlador para los archivos de procesamiento de métodos, así como métodos ordinarios que no procesan archivos. En conclusión, su método de generación automática de clases de controladores emplea análisis de dependencia para analizar a qué campos accede el método y luego construir secuencias de métodos para mutar los valores de los campos a los que se accede de modo que las secuencias de métodos generadas puedan mutar el estado de las instancias de clase. Para obtener los resultados evaluaron a JDriver en commons-Imaging, una biblioteca de imágenes ampliamente utilizada proporcionada por la organización Apache y JDriver ha generado con éxito 149 métodos auxiliares que se pueden usar para crear instancias para 110 clases. Además, se crean 99 clases de controladores para cubrir 422 métodos (Huang & Wang, 2018).

2.2. Bases teóricas

2.2.1. Conceptos y teorías herramienta CASE de generación de código bajo la arquitectura de 3 capas

2.2.1.1. Programación en N Capas

El estilo de la arquitectura de programación en n capas utiliza una distribución jerárquica de roles y responsabilidades que proporcionan una división efectiva para resolver problemas. Los roles nos dan el tipo y forma de la interacción entre capas y responsabilidades, así como la funcionalidad que implementan (De La Torre, Zorrilla, Calvarro, & Ramos, 2010). Si nos enfocamos en la arquitectura tradicional de N-Capas (Forma Lógica), tenemos a la capa de acceso a datos (Fuentes de datos, Servidores externos), capa de negocio y la capa de presentación.

Según el diseño básico de capas se debe separar los componentes de la solución en capas, las capas están acopladas al resto de capas de primer nivel, por ejemplo: la capa 1 es la capa de la base del sistema y la capa 2 debe estar referenciada con la capa 1 y así sucesivamente hasta a la capa N (De La Torre, Zorrilla, Calvarro, & Ramos, 2010).
Vamos a enfocarnos a la programación en tres capas y son las siguientes:

2.2.1.1.1. Capa de presentación

Esta capa presenta visualmente la aplicación o sistema. Se encarga de enviar mensajes a los objetos de la capa de negocio, quien responde directamente o a su vez solicita a la capa de acceso a datos la respuesta, la cual sería la que proporciona los datos que da como respuesta la capa de presentación (Presuman, 2005).

Se puede decir que esta es la cara bonita de una aplicación o sistema, con la que el usuario va a interactuar y mediante la cual hará todo lo que su perfil le permita.

2.2.1.1.2. Capa de negocio

Esta capa se encarga de procesar las solicitudes de la capa presentación, de ser necesario realiza la solicitud a la capa de acceso a datos. Esta capa contiene todos los objetos de la base de datos. Esta capa tiene la capacidad de mantenimiento y reutilización.

Contiene objetos definidos por las clases reutilizables, estas se pueden reutilizar una y otra vez en otras aplicaciones o sistemas. Estos objetos de negocios contienen la gama normal de constructores, métodos para establecer y obtener variables, métodos que realizan cálculos, métodos privados y métodos de comunicación con la base de datos (Presuman, 2005).

En esta capa se establece las reglas que debe cumplirse al momento de realizar una solicitud desde la capa de presentación.

2.2.1.1.3. Capa de acceso a datos

Esta capa se encarga de acceder a los datos de la base de datos, se utiliza esta capa para almacenar datos y recupera toda la información necesaria en una solicitud desde la base de datos (De La Torre, Zorrilla, Calvarro, & Ramos, 2010).

En esta capa se implementa la conexión al servidor y base de datos, se invoca a los procedimientos almacenados, los cuales reciben las solicitudes desde la capa de negocio.

2.2.1.2. Teoría sobre Programación orientada a objetos

2.2.1.2.1. Programación orientada a objetos

La programación orientada a objetos es la programación estructurada y modular, las instrucciones y datos se encapsulan en entidades llamadas clases y luego se crean objetos. Los lenguajes modernos permiten la programación orientada a objetos, y es un estilo que predomina en la industria de programación (Vozmediano, 2017).

2.2.1.2.2. Objeto

Es una persona, animal o cosa, tiene una característica en particular que la distingue de los demás, la cual puede ser representada dentro de una clase. Cada objeto puede tener diferentes campos. Por ejemplo, si hablamos de un cuadrado, este tiene un ancho, alto, color, etc. mientras que un círculo tiene un centro, radio, diámetro, etc (Vozmediano, 2017).

2.2.1.2.3. Clase

La clase es la categoría que agrupa objetos similares, las clases se crean a partir de agrupar objetos de la misma categoría. Por ejemplo, si hablamos de un objeto circulo, cuadrado, triangulo estaríamos hablando de una categoría de figuras geométricas, en este caso mi clase se llamará figuras geométricas. Otro ejemplo, seria si hablamos de los objetos, camioneta, automóvil, ómnibus estaríamos hablando de la categoría vehículo y la clase seria vehículo (Vozmediano, 2017).

2.2.1.2.4. Método

Los métodos son básicamente las operaciones que se puede realizar con un objeto, por ejemplo, si tengo un triángulo algunos métodos serian poder pintar el triángulo, girar, cambiar tamaño, mover, etc. estos métodos son llamados o invocados.

2.2.1.2.5. Procedimientos almacenados

Un procedimiento o subrutina es un subprograma que se encarga de ejecutar un proceso o procesos específicos. Los procedimientos se llaman o invocan por su nombre, por ejemplo, el procedimiento ActualizarColor, nos indica que se va actualizar el color de un objeto.

2.2.1.2.6. Tipos de dato

El tipo de dato especifica de cómo se va a tratar al dato y como se le va a pasar el parámetro. Por ejemplo, un mes del año se puede tratar como int (número) o como string (texto).

2.3. Marco conceptual

CASE: Ingeniería de software asistida por computadora.

Interfaz: Hace referencia a la interfaz gráfica del usuario.

SQL: Lenguaje de consulta estructurada.

CSharp: Lenguaje de programación desarrollada y estandarizada por Microsoft.

Script: Secuencia de comandos, escritas en orden lógico.

LUA: Lua es un lenguaje de programación multiparadigma, imperativo, estructurado y bastante ligero, que fue diseñado como un lenguaje interpretado con una semántica extendible.

UML: Lenguaje Unificado de Modelado.

GXC: Nombre de la herramienta propuesta, proviene de generador xtremo de codigo.

Métrica: Referencia a la forma de realizar la medición de los datos.

Módulos: Es una porción de código de un programa.

GUI: Interfaz gráfica de usuario.

Bootstrap: es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web.

CAPÍTULO III. METODOLOGÍA DE LA INVESTIGACIÓN

3.1. Tipo o enfoque de la investigación

3.1.1. Investigación cuantitativa

La presente investigación es cuantitativa, se utilizó el algoritmo de generación de código para recopilar la cantidad de líneas generadas y también nos muestra el tiempo que demoro en la generación de código.

3.2. Diseño de la investigación

3.2.1. Investigación aplicada

Esta investigación es de tipo aplicada porque, es de conocimiento que el tiempo en el desarrollo de software, siempre debe ser el menor. Así mismo los errores deben de ser nulos o lo más mínimo posible. Para dar respuesta a las interrogantes, se presenta la herramienta CASE GXC para generar código para la arquitectura de 3 capas y contribuir a mejorar el tiempo y reducir los errores.

3.2.1.1. Diseño de arquitectura de la herramienta CASE GXC para generación de código de programación en 3 capas

En la actualidad existen diferentes herramientas CASE para generar código, sin embargo, la mayoría de estas herramientas se basa en la generación desde un modelo fijo. En esta investigación diseñamos una herramienta CASE configurable para generar código para la arquitectura de tres capas. Para la herramienta se utilizó la menor cantidad de formularios y componentes (Botones, labels, datagridview, etc) para que se tenga y cumpla con la sencillez que buscamos en el diseño de la herramienta, sin dejar que sea efectiva y eficaz.

El usuario de la herramienta CASE GXC debe diseñar su base de datos en la interface de la herramienta o en una hoja de cálculo con formato .xls, respetando la estructura requerida por la herramienta para cargar las tablas, campos y tipo de dato, la cual se carga con un primer formulario antes de entrar al formulario de generación de código. Para almacenar la base de datos ingresada en la interface de generación vamos a exportar

a un archivo de formato .xls, para poder recuperar en un futuro la base de datos y generar código.

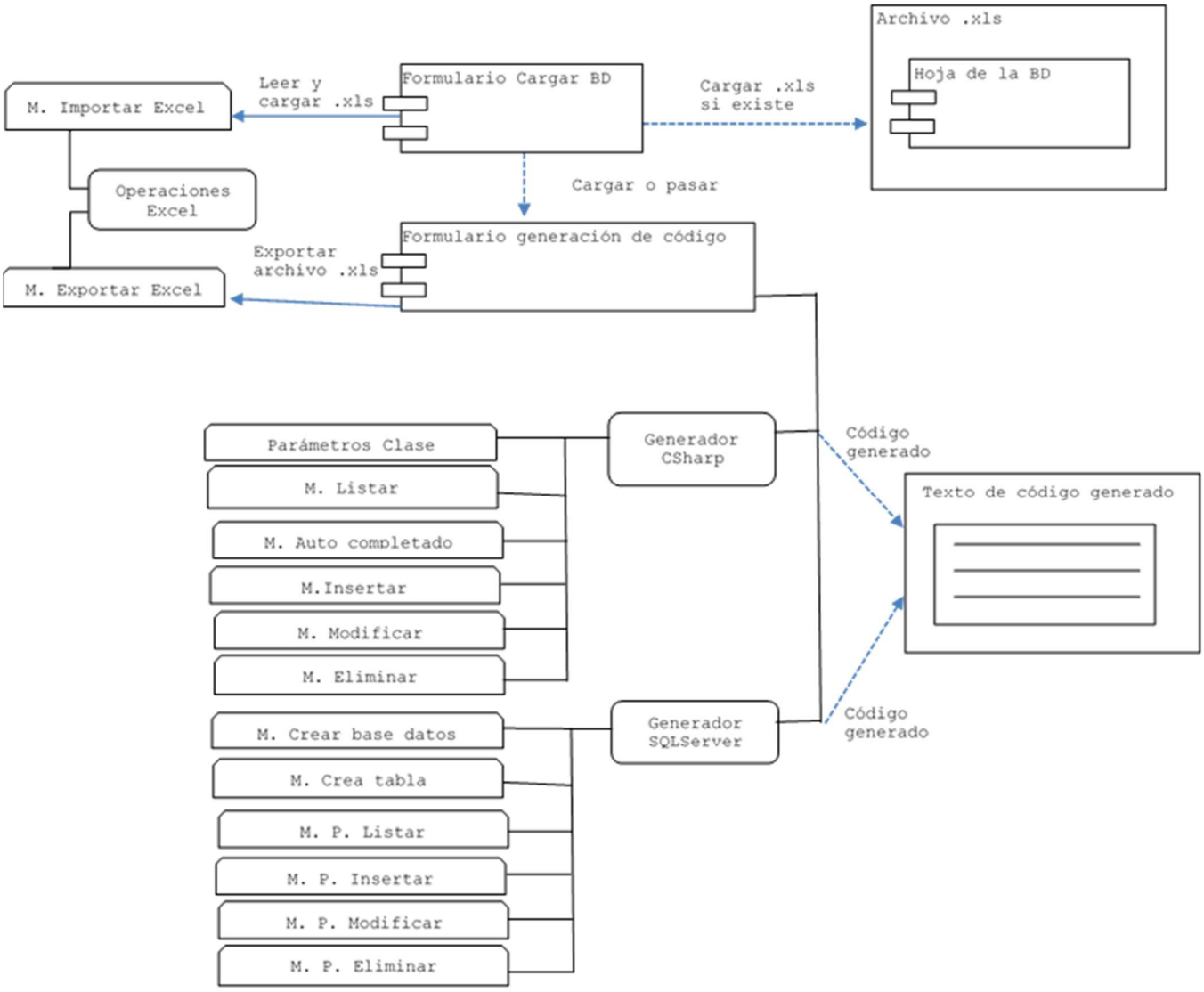

Fig. 1 Diseño arquitectura de la Herramienta CASE GXC

Para la capa de acceso a datos el código que se genera, es el script de creación de la base de datos, script de las tablas, script procedimientos almacenados (listar, insertar, eliminar y modificar).

Con respecto a la capa de negocio se genera código para los parámetros de las clases, los métodos listar, listar autocompletando, insertar, modificar y eliminar.

Toda la generación de código va enlazada, ósea se pone nombres definidos para que se pueda realizar el llamado desde los procedimientos o métodos.

3.2.1.1.1. Componentes

Formulario de inicio: Este formulario se encarga de cargar los archivos .xls, si ya se tuviera un archivo .xls exportado con el formato pre definido. En el caso de que no se cuenta con ningún archivo de .xls, se procede a pasar al formulario generación de código.

Para cargar el archivo .xls la herramienta recupera la ruta del archivo, y necesita el nombre de la base de datos que se tiene que ingresar a través de un textbox. Antes de pasar al formulario generación de código, se debe pre visualizar la BD recuperada desde el archivo .xls para ver que todo este correcto, posteriormente se pasa al formulario antes mencionado.

Formulario generación de código: En este formulario escribimos el nombre de la base de datos como requisito para poder genera código, agregamos las tablas y las columnas de tablas con sus tipos de dato. Adicionalmente el usuario debe identificar cada columna con el tipo de llave (Primary key, Foreign Key o Ninguna).

La otra parte de este formulario se encarga de generar el código, se elige el lenguaje y el tipo de script que desea generar.

Para poder generar el código debemos de tener la base de datos y las tablas ya agregadas dentro del formulario generación de código, posteriormente se debe seleccionar el lenguaje y el tipo de script, seguidamente se genera el código para todas las tablas en bloque.

Este formulario nos da la opción de exportar la BD en un archivo .xls, para tener almacenada la BD y recuperar en un futuro.

Fig. 2 Caso de uso Registrar Tablas

Archivo .XLS: Este archivo se utiliza principalmente por la herramienta para almacenar la BD, con el formato pre establecido.

La estructura de este archivo es las primeras filas de cada columna son las tablas, las siguientes filas son las columnas y estas tienes la estructura:

Nombre de columna[Tipo de dato / Tipo de llave-Tabla Referencia]

Tabla 1 Descripción de archivo .xls

Descripción de la estructura del archivo .xls	
Tabla	Tabla viene a ser una tabla de la base de datos, se ubica en la primera fila del archivo .xls. Es decir las primeras filas de cada columna vienen a ser el nombre de las tablas
Columna	Columna viene a ser el nombre de la columna de una tabla, puede ser de la columna 1 a la columna N. Las columnas se ubican desde la segunda fila de cada columna.
Tipo de dato	Tipo de dato viene a ser el tipo de dato de la columna, por ejemplo, puede ser varchar, int, bool, etc. Para poner el tipo de dato debe existir antes un corchete de apertura ”[”
Tipo de llave	El tipo de llave indica si una columna es clave primaria, clave foránea o ninguna. El Tipo de llave va después del tipo de datos antecedido de un espacio. Si es clave foránea va junto a un guion “-” y junto a la tabla de referencia, al final se cierra con un corchete de cierre “]”

Tabla de referencia	Viene a ser la tabla de la cual se está referenciando la clave foránea

3.2.1.2. Generación de código de la herramienta

En esta parte se puede visualizar cómo trabaja la herramienta para generar el código y que sintaxis se utiliza.

Para generar el código tenemos las clases, y es aquí donde se conecta las tablas de la BD y se determina y configura las sintaxis de los script.

Fig. 3 Conexión para generar código

En la Fig. 3 se puede ver gráficamente la conexión que tiene los elementos tabla, clase y script. El proceso que se realiza para generar código, primero se identifica la tabla para la cual se va a generar código, segundo se llama al método de la clase, este método ya tiene los parámetros porque son obtenidos desde la tabla, tercero el método de la clase genera el script de la tabla y esto se muestra en el formulario de generación de código.

Implementamos diferentes clases con sus respectivos métodos, para poder generar el código (aquí está la sintaxis de los scripts). Para determinar que clase se va utilizar, el usuario selecciona el lenguaje. De ahí se llama al método según la selección de tipo de script, al seleccionar el tipo de script estamos eligiendo el método de la clase.

Para entender un poco mejor el proceso de generación vamos a mostrar el siguiente diagrama de flujo:

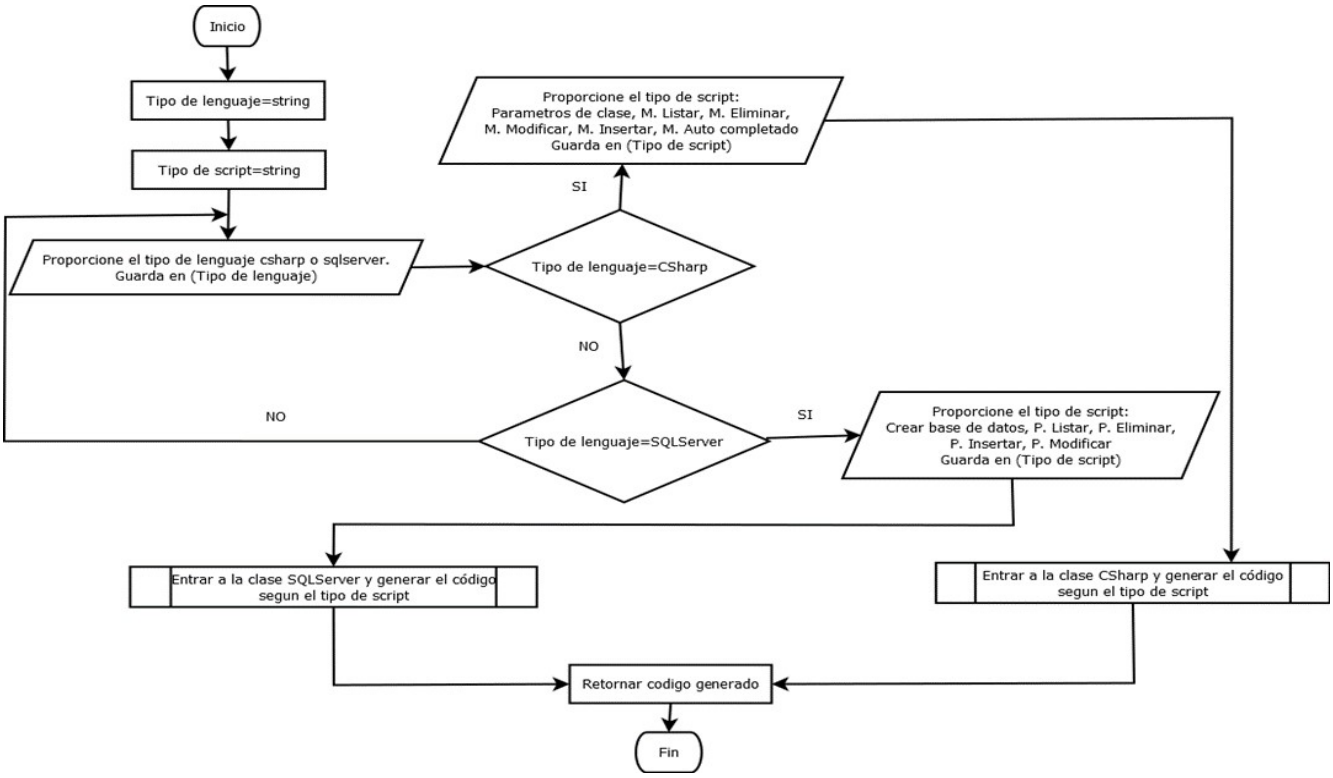

Fig. 4 Diagrama de flujo de generación de código

Vamos a mostrar un poco de la clase CSharp y mostraremos un fragmento del código:

```
public class CSharp
{
    1 referencia
    public string ParametrosClase(string parametros, string ParametrosRefactorizados)
    {
        string Resultado = parametros+ "\r\n\r\n" + ParametrosRefactorizados;
        return Resultado + "\r\n";
    }
    1 referencia
    public string ProcedimientosListar(string Tabla, string NombreProcedimiento)
    {
        string Resultado = "public DataTable Listar_" + Tabla + "()" + "\r\n" +
            "{\r\n" +
            "   try\r\n" +
            "   {\r\n" +
            "       DataTable oTabla = oDatos.Get_DataTable("+ NombreProcedimiento + ");\r\n" +
            "       return oTabla;\r\n" +
            "   }\r\n" +
            "   catch (SqlException ex)\r\n" +
            "   {\r\n" +
            "       MessageBox.Show(ex.Message, Error, MessageBoxButtons.OK, MessageBoxIcon.Error);\r\n" +
            "       return null;\r\n" +
            "   }\r\n" +
            "}\r\n";
        return Resultado + "\r\n";
    }
```

Fig. 5 Fragmento de código

En la Fig. 6 se puede ver cómo es que se genera el código, si se ve el método ParametrosClase puede ver que necesita parámetros y ParametrosRefactorizados, y este método devuelve una cadena de texto. Si vemos el método ProcedimientosListar, también recibe parámetros, dentro del mismo se puede ver la sintaxis que se utiliza en el ejemplo, la misma que puede ser configurada, posteriormente devuelve una cadena de texto.

Básicamente todos los métodos de las clases van a devolver cadenas de texto.

Describiremos brevemente la forma de utilización de la herramienta para la capa de datos y para la capa lógica de negocio.

3.2.1.2.1. Capa de datos

Para esta capa se propone la generación del script de la base de datos con la sintaxis "create database Nombre_BD". Para generar el script de las tablas dependerá del lenguaje seleccionado pero principalmente, se realiza la generación de una tabla o de todas las tablas.

A partir de la BD, también se propone que se pueda crear procedimientos almacenados propios de un CRUD, como insertar, modificar, listar, eliminar. Esto se genera para una sola tabla o para todas las tablas en bloque.

3.2.1.2.2. Capa lógica de negocio

Los script propuestos para generar código son generar las clases, parámetros de clases, métodos insertar, modificar, listar y eliminar.

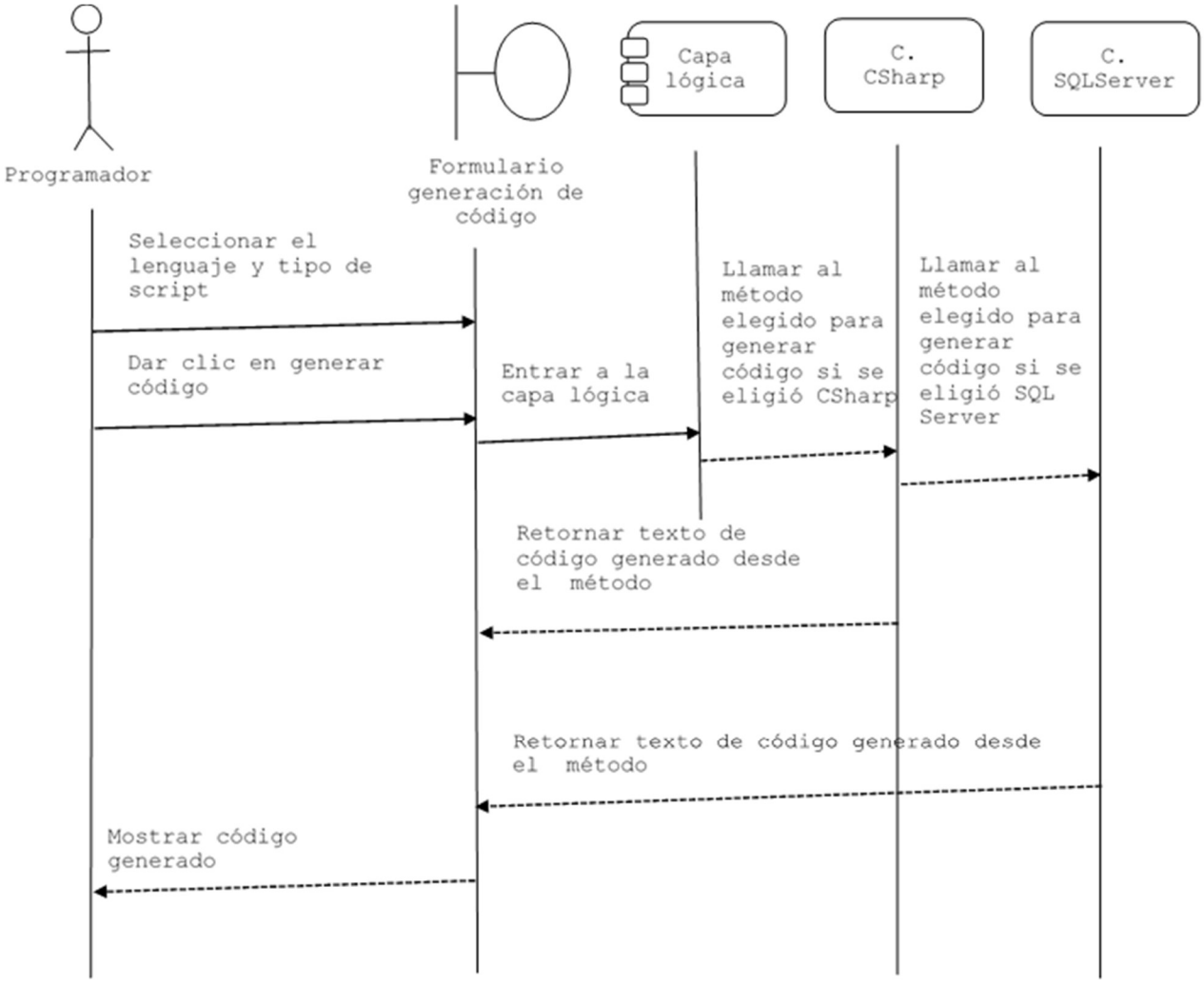

Fig. 6 Diagrama de secuencia de generación de código

3.3. Recolección de datos

Para validar la herramienta, se utilizó 5 entradas que vienen a ser esquemas de base de datos, los cuales son introducidos en la interface de generación de código y a partir de este generamos código para SQL Server, para el script de base de datos, procedimientos almacenados (listar, modificar, eliminar e insertar), también generamos código para el lenguaje Csharp, para los parámetros de las clases, métodos (listar, listar auto completando para un combobox, insertar, modificar y eliminar).

Tabla 2 Esquemas de base de datos

Abreviación	Nombre de esquema de base de datos	Cantidad de tablas
E1	Adventurewords	10
E2	Hospital	6
E3	Hotel	15
E4	Northwind	13
E5	Pubs	7
	Total de tablas	**51**

En la tabla 2 se puede ver que para validar la herramienta en total vamos a utilizar 51 tablas. A continuación, les mostramos los esquemas de base de datos de la tabla 1.

En la Fig. 7 Tenemos el esquema de base de datos de AdventureWorks, que viene a ser la base de datos de una empresa ficticia, en total tiene 10 tablas con las cuales realizamos las pruebas, dentro de la herramienta CASE GXC.

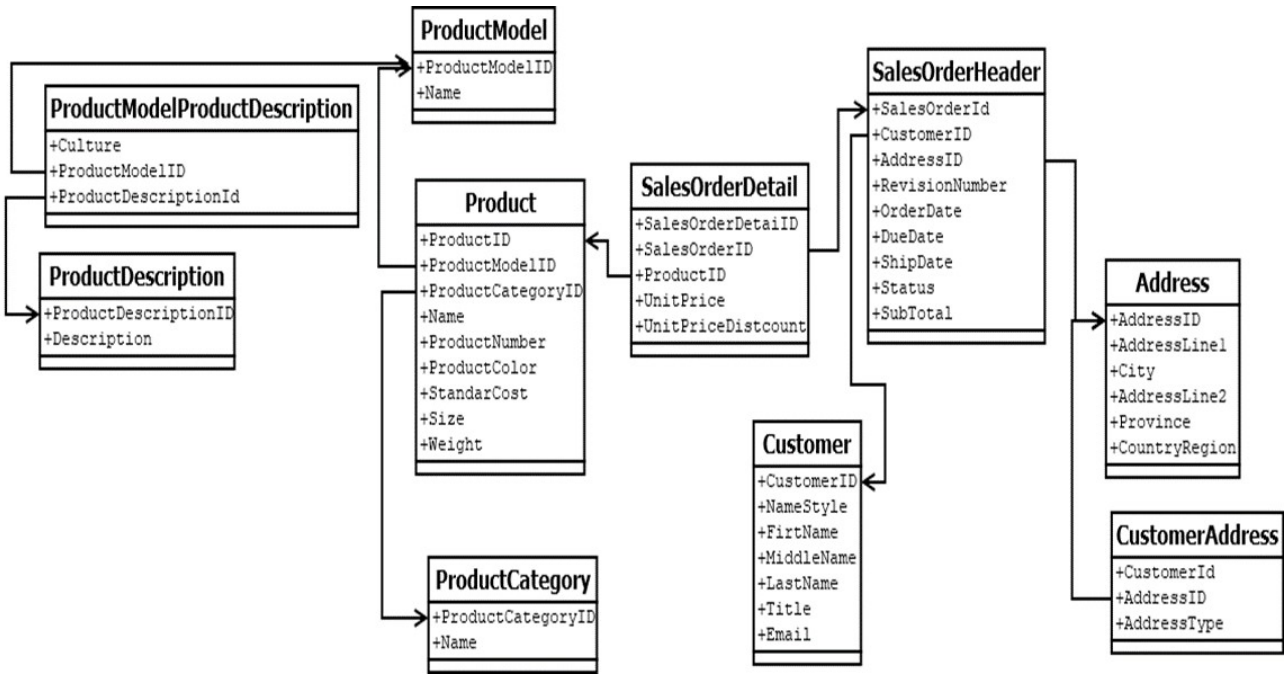

Fig. 7 Diagrama de base de datos AdventureWorks

El esquema de base de datos hospital de la fig. 8, nos muestra un hospital ficticio de un nivel principiante, en vista que una base de datos de un hospital es más compleja y tiene muchas más tablas de las que se muestra a continuación. Este esquema tiene

6 tablas con las cuales realizamos las pruebas.

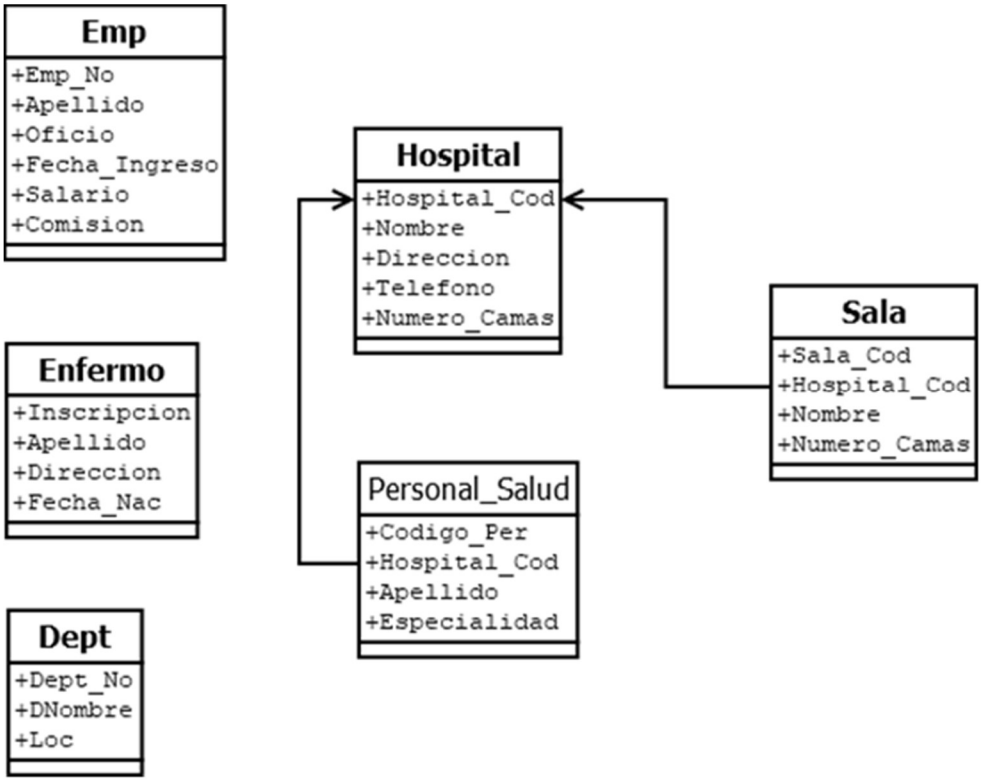

Fig. 8 Diagrama de base de datos hospital

El esquema de base de datos hotel (fig. 9), es de una organización ficticia, pero se asemeja a la realidad de un hotel, en vista que el diagrama contempla al cliente, modo de pago, empleado, tipo de empleado, estancia, habitación, tipo de habitación, mini restaurante, mini bar, bebidas, tipo de marca, tipo de bebidas. En total tenemos 15 tablas para generar código.

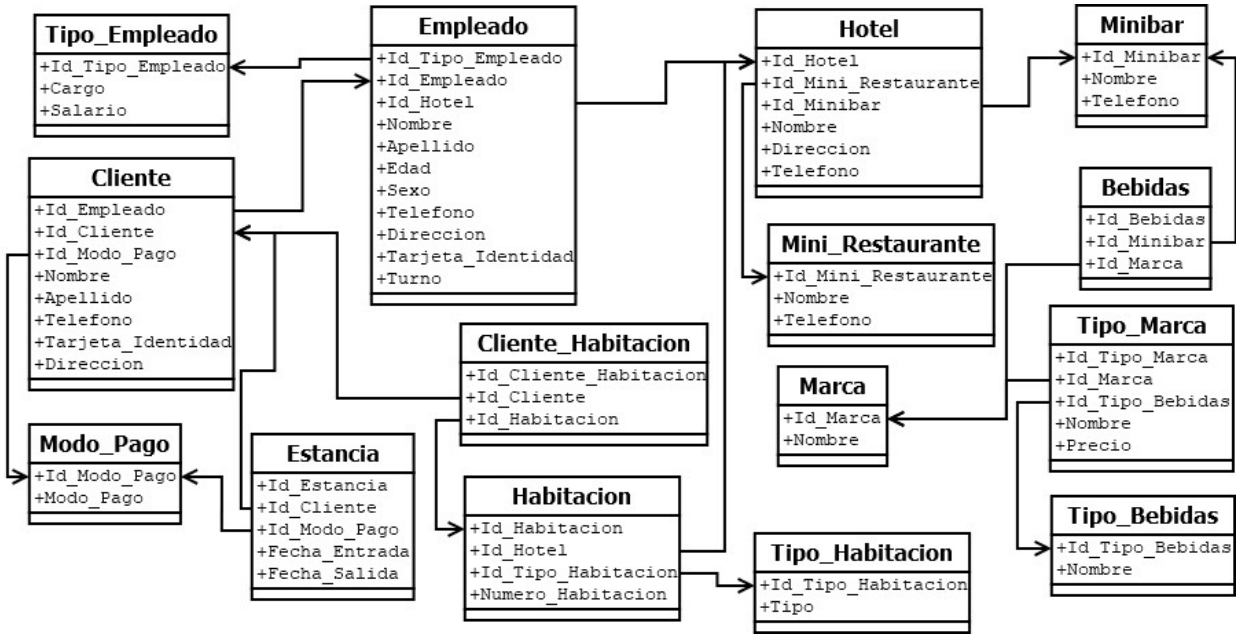

Fig. 9 Diagrama de base de datos hotel

El esquema de la base de datos Northwind (fig. 10), es de una organización ficticia que gestiona pedidos, productos, clientes, proveedores y otros de una pequeña empresa. El diagrama cuenta con 13 tablas para las pruebas que realizamos.

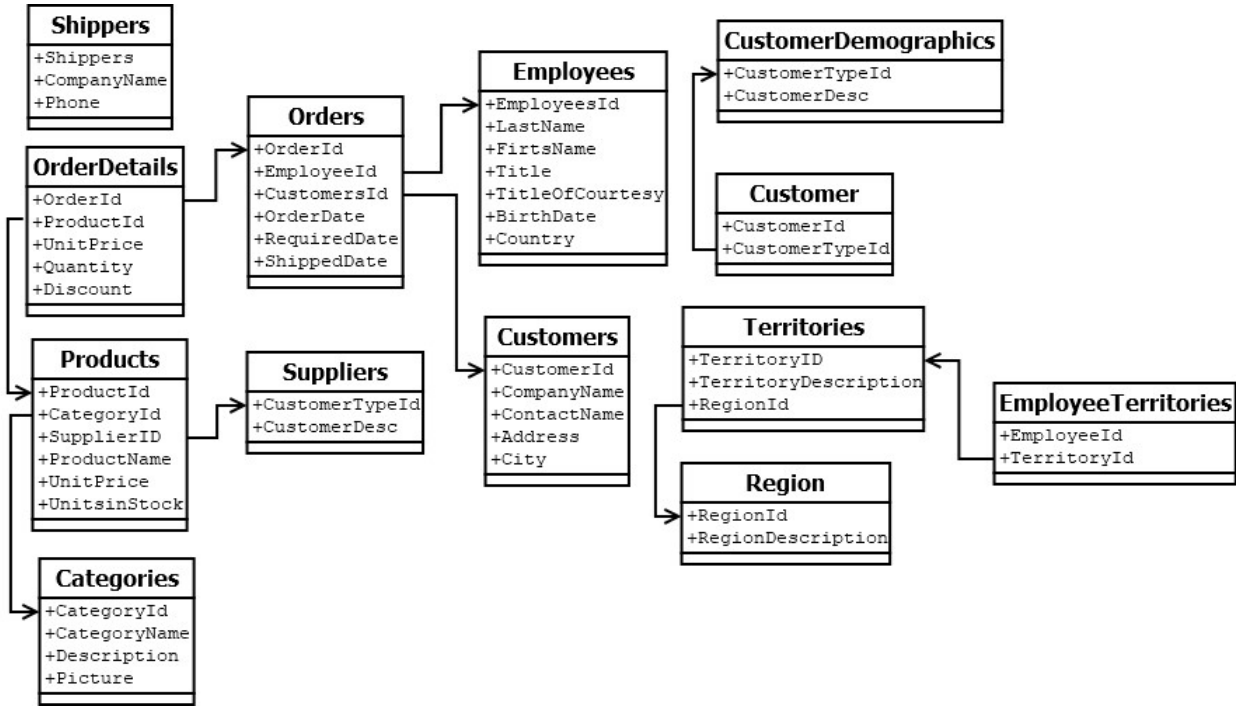

Fig. 10 Diagrama de base de datos Northwind

El esquema de base de datos Pubs es de una organización de publicaciones, en total para las pruebas consideramos 7 tablas de este esquema.

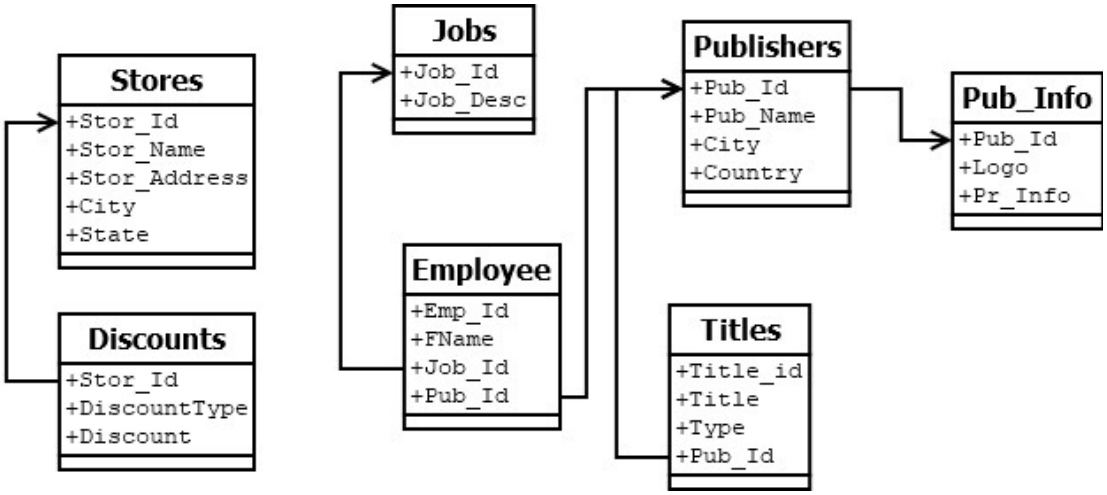

Fig. 11 Diagrama de base de datos Pubs

3.4. Resultados

Para realizar las pruebas, ingresamos cada uno de los esquemas de base de datos (tablas y columnas), en el formulario de generación de código. Seguidamente realizamos la generación de código por cada tipo de código, donde recuperamos el tiempo, la tasa de error y las líneas de código. A continuación, les presentamos la tabla 3 generación de código libre de errores.

Tabla 3 Generación de código libre de errores

Código generado	E1	E2	E3	E4	E5
Script de base de datos	√	√	√	√	√
Procedimientos listar	√	√	√	√	√
Procedimientos insertar	√	√	√	√	√
Procedimientos modificar	√	√	√	√	√
Procedimientos eliminar	√	√	√	√	√
Parámetros de clases	√	√	√	√	√
Métodos listar	√	√	√	√	√
Métodos listar auto completando	√	√	√	√	√

Métodos insertar	√	√	√	√	√
Métodos modificar	√	√	√	√	√
Métodos eliminar	√	√	√	√	√

En la siguiente tabla que se presenta, vamos a evidenciar las líneas de código generadas con la herramienta CASE GXC, de igual manera veremos el tiempo que lleva generar cada script.

Tabla 4 Resultados de medición de código generado

Script generado	**Tiempo de generación en milisegundos**					**Líneas de generación de código**				
	E1	**E2**	**E3**	**E4**	**E5**	**E1**	**E2**	**E3**	**E4**	**E5**
Script de base de datos	18	8	37	24	7	164	96	233	193	109
Procedimientos listar	18	5	27	20	9	151	91	226	196	106
Procedimientos insertar	21	11	48	32	9	209	123	303	258	138
Procedimientos modificar	25	8	44	35	10	219	129	318	271	145
Procedimientos eliminar	16	7	40	27	9	171	103	256	222	120
Parámetros de clases	45	20	83	60	16	693	377	899	713	365
Métodos listar	16	6	34	24	7	151	91	226	196	106
Métodos listar auto completando	35	13	73	55	18	241	145	361	313	169
Métodos insertar	27	10	60	42	13	241	145	361	313	169
Métodos modificar	29	10	56	43	12	241	145	361	313	169
Métodos eliminar	28	9	56	43	14	241	145	361	313	169
TOTALES	278	107	558	405	124	2722	1590	3905	3301	1765

Los esquemas de base de datos en la tabla vienen a ser E1, E2, E3, E4, E5. Donde E1 tiene 10 tablas, E2 tiene 6 tablas, E3 tiene 15 tablas, E4 tiene 13 tablas, E5 tiene 7 tablas.

En la tabla 4 de resultados se puede ver que, a mayor cantidad de tablas, mayor es el tiempo de generación y mayor son las líneas de código generadas.

La generación de código en relación al tiempo, se puede decir que va a disminuir el tiempo de desarrollo, puesto que, para generar código de 15 tablas en todos los scripts vistos en la tabla, nos lleva un tiempo de 558 milisegundo, que viene a ser poco más de la mitad de un segundo. En ese tiempo se genera 3905 líneas de código incluyendo los saltos de línea. Si a partir de estos resultados queremos ver cuantas líneas de código se generan por cada milisegundo, vendría a ser 6.998, aproximadamente 7 líneas por milisegundo.

Para una mejor vista de los resultados, presentamos a continuación una tabla compacta, de todos los esquemas de base de datos y los resultados en global en tiempo en milisegundos y líneas de código.

Tabla 5 Tiempo en milisegundos y líneas de código por entradas

Esquema de base de datos	Tiempo en milisegundos	Líneas de código
Adventurewords	278	2722
Hospital	107	1590
Hotel	558	3905
Northwind	405	3301
Pubs	124	1765
Totales	**1472**	**13283**

La tabla 5 no muestra un resultado de que, para generar código para todos los esquemas de base de datos, solo se emplea 1472 milisegundos y en total se genera 13283 líneas de código.

Frente a estos resultados se puede decir que la herramienta CASE GXC para generar código bajo la arquitectura de programación en 3 capas es eficiente y va a contribuir a reducir errores en el desarrollo, reducción de esfuerzo, eficiencia en el desarrollo y tiempo de desarrollo.

3.5. Discusión de resultados

La tasa de error de la investigación (Pang, y otros, 2020), muestra 6.00 para la web, 34.24 para androide y 35.20 para iOS. Mientras que la generación de código de la presente

investigación, tiene error cuando el usuario se equivoca al ingresar la tabla, columna o tipo de dato incorrecto. En caso de que el usuario no cometa errores, no se evidencia tasa de error puesto que está correctamente pre configurado para generar el código. Por lo antes mencionado consideramos que la tasa de error de la presente investigación es mínima.

La investigación (She & Zheng, 2020), dentro de la aplicación práctica de su algoritmo, refiere que para generar al menos 72,000 lo puede hacer en tan solo unos minutos, mientras que la herramienta CASE GXC genera el código un promedio de 9.024 líneas de código por milisegundo, si realizamos la operación de líneas de código generadas por minuto tendíamos 9.024 líneas de código X 60,000 milisegundos (un minuto) resulta 541,440 líneas de código, por ello la herramienta CASE GXC saca una ligera ventaja en cuestión de tiempo.

(Possatto & Lucrédio, 2015), Señala que con su herramienta consiguieron la reducción de esfuerzo en cuanto a la programación de código, con resultados buenos, la presente investigación también obtuvo buenos resultados en cuanto a la reducción de esfuerzo, porque ya no se tendrá que escribir código para la BD, procedimientos almacenados (listar, insertar, modificar, eliminar), parámetros de clase, métodos (listar, eliminar, insertar, modificar).

La investigación (Sadaf, Athar, & Azam, 2016) genera el código a partir de un diagrama de base de datos de forma gráfica y lo transforma a código de clases. Mientras que en esta investigación se necesita que el usuario ingrese su esquema de base de datos dentro de la herramienta CASE GXC y a partir de ello se genera código, para la capa de negocios y script para la base de datos.

(Egea & Dania, 2017) Genera código SQL compatible con MySQL, MariaDB, PostgreSQL y SQL Server, mientras que en la presente investigación genera código para SQL Server y código para clases en el lenguaje CSharp. Por lo tanto la investigación (Egea & Dania, 2017) enfoca su generación de código a varios motores de base de datos y está delante de la presente investigación en ese enfoque.

(Wibowo, Hendradjaya, & Widyani, 2015) Enfoca la generación de código de prueba de unidad y se integra a ZeroBrane Studio, la presente investigación se enfoca en

generar código para la programación en 3 capas y para ello se utiliza el lenguaje de programación CSharp, como herramienta para poder programar.

Xiangei y Yongehun (She & Zheng, 2020) generan código para la interfaz web basado en la tecnología Java POI. La investigación que presentamos genera código para la base de datos, la capa lógica de negocios. Son enfoques diferentes, donde la investigación (She & Zheng, 2020) se basa en el diseño y la presente investigación se basa en la funcionalidad, también se puede decir que (She & Zheng, 2020) se basa en frontend y la presente investigación se basa en backend.

CAPÍTULO IV. ASPECTOS ADMINISTRATIVOS

4.1. Cronograma de actividades

Actividad	Meses del año 2020											
	Junio					Julio				Agosto		
	1	2	3	4	5	1	2	3	4			
Revisión de papers y referencias												
Redacción del proyecto de investigación												
Metodología de investigación												
Análisis de requerimientos y procesos herramienta												
Diseño de interfaces de la herramienta												
Programación de la herramienta												
Pruebas de rendimiento												
Elaboración de informe final												

4.2. Recursos humanos y materiales

La investigación va a necesitar de recursos materiales y conocimiento sobre temas de programación, así como de gestores de base de datos y modos de almacenamiento de datos.

4.2.1. Recursos humanos

Nombres y Apellidos	**Institución perteneciente**	**Función en el proyecto**
Juan Moises Rojas Torres	Universidad Líder Peruana	Investigador principal

4.3. Presupuesto

Descripción	Unidad de medida	Costo Unitario (S/.)	Cantidad	Costo total (S/.)
Honorario del programador	1	2000.00	1	2000.00
Insumos y materiales	1	600.00	1	600.00

Servicios de terceros	1	200.00	2	400.00
Gastos administrativos	1	200.00	1	200.00
Personal científico	1	1500.00	1	1500.00
			Total	**4700.00**

REFERENCIAS BIBLIOGRÁFICAS

Alfonso, J., & Restrepo, F. (2017). Automatic Source Code Generation for Web-based Process-oriented Information Systems. *ACM*, 11.

Brunnlieb, M., & Poetzsch-Heffter, A. (2016). Application of Architecture Implementation Patterns by Incremental Code Generation. *ACM*, 12.

De La Torre, C., Zorrilla, U., Calvarro, J., & Ramos, M. (2010). *Guía de Arquitectura N-Capas Orientada al Dominio con .NET 4.0.* España: Krasis Consulting.

Egea, M., & Dania, C. (2017). SQL-PL4OCL: An Automatic Code Generator from OCL to SQL Procedural Language. *IEEE*, 1.

Franky, M. C., Pavlich-Mariscal, J. A., Acero, M. C., Zambrano, A., Olarte, J. C., Camargo, J., & Pinzón, N. (2016). ISML-MDE: A Practical Experience of Implementing a Model Driven Environment in a Software Development Organization. *Emerald Insight*, 18.

Galin, D. (2018). Software Quality: Concepts and Practice. *IEEE*, 17.

Han, J., Zhao, T., Yu, Z., Shi, W., & Huang, M. (2019). Design and Implementation of B/S Architecture Code Automatic Generation System. *IEEE*, 4.

Huang, Z., & Wang, Y. (2018). JDriver: Automatic Driver Class Generation for AFL-Based Java Fuzzing Tools. *Symmetry*, 16.

Li, Z., Jiang, Y., Zhang, X. J., & Xu, H. Y. (2020). The Metric for Automatic Code Generation. *ACM*, 8.

Pang, X., Zhou, Y., Li, P., Lin, W., Wu, W., & Wan, J. Z. (2020). A novel syntax-aware automatic graphics code generation with attention-based deep neural network. *ELSEVIER*, 12.

Possatto, M. A., & Lucrédio, D. (2015). Automatically propagating changes from reference implementations to code generation templates. *ELSEVIER*, 14.

Presuman, R. (2005). *Ingeniería de software un enfoque práctico.* España: Pearson Educación S.A.

Sadaf, S., Athar, A., & Azam, F. (2016). Evaluation of FED-CASE - A Tool to Convert. *IEEE*, 4.

She, X., & Zheng, Y. (2020). An Automatic Page Code Generation Method Based On Excel Template And Poi Technology. *IEEE*, 5.

Sulistyo, S., & Prinz, A. (2009). Recursive modeling for completed code generation. *ACM*, 7.

Vozmediano, A. M. (2017). *Java para novatos: Cómo aprender programación orientada a objetos con Java sin desesperarse en el intento.* Safe creative.

Wibowo, J. T., Hendradjaya, B., & Widyani, Y. (2015). Unit test code generator for lua programming language. *IEEE*, 5.