Revista Peruana de Computación y Sistemas 2022 4(2):31-40
https://doi.org/10.15381/rpcs.v4i2.24852

ISSN Online: 2617-2003
Facultad de Ingeniería de Sistemas e Informática
Universidad Nacional Mayor de San Marcos

Artículo de Contribución

Diseño de modelo para generación de código para C# y SQL Server

Model design for code generation for C# and SQL Server

Juan M. Rojas Torres [1], Hugo D. Calderon-Vilca [2]

[1] Universidad Líder Peruana, Depto. Sistemas e Informática, Cusco, Perú
E-mail: moises.rto@gmail.com, ORCID: https://orcid.org/0000-0002-9489-9066

[2] Universidad Nacional Mayor de San Marcos, Depto. Ingeniería de Software, Lima, Perú
Autor de correspondencia: hcalderonv@unmsm.edu.pe, ORCID: https://orcid.org/0000-0002-1177-4947

Resumen

En la actualidad existen herramientas que generan código fuente de forma automática. En este trabajo se propone un modelo de generación de código fuente enfocada en la arquitectura de 3 capas abarcando el lado del backend utilizando data driven, adicionalmente construimos una herramienta de generación de código fuente. La Herramienta permite cargar un esquema de base de datos y a partir de este genera el código fuente para el lenguaje SQL Server y C#, utilizamos un archivo .xls para importar o exportar el esquema de la base de datos. Hemos utilizado 5 requerimientos de generación de códigos y sus esquemas de base de datos para la prueba, obteniendo como resultado ha generado código fuente sin errores. Realizando el análisis de tiempo nuestro modelo genera 6,998 líneas de código por segundo, en total hemos generado 13 283 líneas de código fuente en 1472 milisegundos que corresponden a 5 requerimientos de generación de códigos fuente con sus respectivas bases de datos.

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

Abstract

Currently there are tools that generate source code automatically. We propose a source code generation architecture focused on the 3-layer architecture, additionally we build a source code generation tool. The Tool allows to load a database schema and from this it generates the source code for the SQL Server and C# language, we use an .xls file to import or export the database schema. We have used 5 code generation requirements and their database schemas for the test, obtaining as a result the generation of source code without errors. Carrying out the time analysis, our model generates 6,998 lines of code per second, in total we have generated 13,283 lines of source code in 1472 milliseconds that correspond to 5 requirements for generating source code with their respective databases.

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

Recibido 05/11/2022 - Aceptado 16/12/2021 - Publicado: 31/12/2022

Citar como:
Rojas, J. & Calderon-Vilca, H. (2022) Diseño de modelo para generación de código para C# y SQL Server. Revista Peruana de Computación y Sistemas, 4(2):31-52. https://doi.org/10.15381/rpcs.v4i2.24852

1. Introducción

Podemos encontrar diferentes caminos para desarrollar software, para el diseño de software se utilizan diferentes metodologías, existen herramientas para ayudar en el desarrollo de software y la mayoría de estas herramientas se basan en modelos preestablecidos. Uno de los problemas más resaltantes en el desarrollar software es el tiempo.

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. Una generación de código de prueba unitaria para acelerar el proceso y evitar la repetición es la propuesta del Institut Teknologi Bandung, desarrollaron el generador utilizando el lenguaje de programación LUA, que es un lenguaje de scripting rápido, liviano e integrable [2].

Los analistas y desarrolladores de la industria de TI tienen que dedicar mucho tiempo y esfuerzo en hacer manual de ingeniería directa e inversa. [5] Introduce 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.

También otros investigadores como [21] logran evaluar de forma automática el código fuente, aplicaron para los concursos de programación de estilo ACM/ICPC.

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. En una investigación realizada [4] aborda este inconveniente mediante la introducción de patrones de implementación de arquitectura, que se aplican automáticamente al código de la aplicación mediante un enfoque de generación incremental.

La generación de código totalmente automatizada a partir de las partes de comportamiento sigue siendo difícil, su funcionamiento es restringida a áreas de aplicación específicas que utilizan un lenguaje específico de dominio, DSL.

El problema que podemos 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.

Ingenieros de software, analistas programadores, analistas funcionales, etc. Están enfocados en resolver procesos poco eficientes de las organizaciones, para esto pueden desarrollar sistemas de información, aplicaciones móviles por mencionar algunas soluciones informáticas. Algunas de estas soluciones podrían terminar siendo magnos desarrollos de sistemas.

Existen diferentes herramientas CASE que tiene funcionalidades para aportar en las diferentes fases de desarrollo de software. En la actualidad no son muy utilizadas porque muchas de las herramientas CASE no están modeladas para el gusto de los equipos de desarrollo de software, uno de los motivos es que trae modelos predefinidos y dicho software no se puede configurar, para poder generar el código.

Según el libro Software Quality: Concepts and Practice indica que se presentaron resultados alentadores para AutoFix. 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 [8]. Según esos resultados se puede evidenciar que una herramienta CASE puede corregir fallas del software.

La presente investigación tiene como objetivo, diseñar una modelo para generar código en la arquitectura de programación de 3 capas.

El diseño del modelo propone reducir el tiempo de desarrollo de sistemas de información, con modeladores de código determinados por cada programador y generar código en distintas capas de ser el caso (capa de datos, capa lógica de negocios, capa de presentación) y además interviene en el ambiente de la construcción de la base de datos y procedimientos almacenados.

La propuesta también permite aumentar la eficacia y eficiencia de los procesos, reducir los recursos inevitables y disminuir las deficiencias del software. Las herramientas CASE favorecen a la economía del desarrollo de software.

Con el diseño del modelo, 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.

2. Estado de arte

Wibowo, J. T. P. y compañia. proponen el generador de prueba de unidad Lua (LUTG), que se integra a ZeroBrane Studio, siendo un Plugin que conecta sin inconvenientes la codificación y prueba. El generador logra generar código para pruebas de unidad, Guarda datos de los casos de prueba en archivos de tipo Lua y XML, y genera datos de prueba en forma automática para ello utiliza una habilidad asentada en búsqueda, algoritmo genético, para lograr corduras de prueba de cobertura de rama completa [2].

La propuesta de Brunnlieb, M., & Poetzsch-Heffter, A. [4]. Presenta dos condiciones prioras su enfoque, primero, describir una arquitectura como referente que utilice estilo basado en patrones para extraer patrones

AIM y esta sea suficientemente fácil. Lo que se ha dicho incluye tener pautas sobre los grados de código, de manera óptima, fragmentos de código de muestra que muestren las mejores prácticas. En segundo lugar, la arquitectura asociada debe apuntar a suficientes convenciones de nomenclatura, además de una pila de tecnología para permitir la generación incremental para insertar código generado automáticamente en el código existente.

Automatizar el desarrollo apoyado en modelos resulta más eficiente en comparación con el desarrollo simple. El modelado UML es un patrón estándar para la ingeniería y para que los proyectos tengan éxito, este enfoque se está volviendo esencial. La sistematización del proceso ayuda a redactar código con errores reducidos y que ocupan menor tiempo y esfuerzo. La participación humana se minimiza, lo que lleva a que se elimine al máximo los errores ocasionados por múltiples factores, por ejemplo, el escaso conocimiento, falta de interés o fatiga. La precisión y eficiencia son primariamente causas que influyen a los lectores a utilizar el enfoque [5].

Se utiliza un enfoque combinado, de arriba hacia abajo que refina recursivamente elementos de actividad de un diagrama de actividad y se utiliza otro enfoque en sentido contrario de abajo hacia arriba desarrollando una herramienta que crea ejecutores de actividad y mecanismo de mapeo inferior de acciones en los ejecutores. Para poder generar código, los lenguajes de modelado requieren disociar operaciones lógicas, aritméticas básicas y operaciones de cadenas, como constructores de actividades primordiales [6].

La guía métrica propone incluir características métricas con un total de seis. Utilizando tres instrumentales automáticos de generación de código con el fin de computar el código experimental. En las conclusiones podemos encontrar resultados con respecto a la medición del análisis que muestran un modelo métrico de generación automática juicioso, lo que resulta ventajoso para medir la generación automática de código, las instrucciones para conseguir resultados son la utilización de CodeGeneration, java-codetool y aiXcoder con el fin de generar objetos experimentales. Se usa generación de código basada en plantilla y aprendizaje automático. Posteriormente el código se mide automáticamente. Entre el código generador por CodeGeneration y java-codetool la calidad del código fuente es aproximadamente la misma. Se emplean pocas líneas de código con las herramientas apoyadas en plantillas en comparación con las apoyadas en aprendizaje automático. Las herramientas que utilizan aprendizaje automático usan un espacio y tiempo superior en comparación con las herramientas que se apoyan en plantillas, esto con respecto a la cantidad [13].

Por la similitud de módulos de arquitectura B/S, se ejecuta un trabajo redundante a menudo que involucra copiar y pegar en la programación. Un diseño/implementación del sistema de automático código de arquitectura B/S es la propuesta. Presenta tecnologías relacionadas que utilizan en su sistema y elaboran las funcionalidades por módulo de sistema. Definen el XML a fin de referir entidades, se implementa funciones del sistema y módulos generales. Como conclusión se observa una manifestación que el sistema de generación automática de código mejora la eficiencia en desarrollo. Se sigue el proceso basado en plantilla para leer información de atributos y sustituye las etiquetas de las variables y posteriormente genera código final. Mediante la práctica demuestran como los usuarios crean velozmente aplicaciones con arquitectura B/S a través del archivo que describe la entidad que provee el usuario manipulando el generador de código [14].

Un mecanismo para descubrir y trascender semiautomáticamente modificaciones en el código en plantillas, sincronizados con menor esfuerzo es la propuesta de [15], los resultados señalan que utilizando automatizaciones es posible disminuir costo y esfuerzo en el mantenimiento y evolución de generación de código determinada en plantillas de infraestructura, se observa el mecanismo elaborado consigue arrastrar a una disminución del 50% de esfuerzo requerido para elaborar la plantilla de mantenimiento y evolución, comparado con una elaborada a mano. También se observa el resultado depende del ambiente de la tarea de evolución y mantenimiento, en vista de que algunas tareas tenían una ventaja visible en el uso del mecanismo.

Un modelo llamado HGui2Code que integra características GUI con atención visual (extraída por CNN) con características semánticas con capacidad DSL (extraída por LSTM), proponen, un modelo novedoso SGui2Code que utiliza una red ON-LSTM para generar código DSL que sea correcto en sintaxis [16], utilizan una red neuronal profunda basada en la atención para alinear el código DSL con los píxeles GUI correspondientes mediante un mecanismo de atención híbrido, al comparar la tasa de error de las líneas de base incluido los modelos pix2pix y ABHD, demostraron que sus modelos son más efectivos.

Un procedimiento automatizado de generación de código de página asentado en plantillas Excel y tecnología POI adentro del cual tienen dos ventajas, señalan que el método propuesto efectúa la generación automática y rápida de interfaces, consiguen lograr la reutilización de código, disminuyen costos de desarrollo y también mejoran la eficiencia de desarrollo [17].

Un rumbo para la generación automática, un modelo (aplicación web) que elabore métodos comerciales usando la especificación restringida de lenguaje natural, tiene un proceso de creación veloz para ejemplares para aplicaciones web, enfocados a métodos comerciales manejando una descripción restrictiva de lenguaje natural, obtuvieron finalmente código fuente preparado para ser integrarse al producto final y esto disminuye los problemas que se asocian a las etapas de elaboración de requisitos y diseño[18].

La práctica de desarrollo ISML-MDE que es un ambiente establecido por modeladores que alcanza componentes principales: un marco de generación de código (ISML-GEN), lenguaje de modelado para aplicaciones corporativo (ISML) y LionWizard. Muestra el enriquecimiento practico para la elaboración de ISML-MDE en un ambiente establecido por modeladores que existieron para automatizar y abstraer la elaboración de aplicaciones corporativas, resulto que la abstracción en ISML es adecuadamente alto, para esconder los pormenores de consumación, pero inferior para detallar la mayoría en aplicaciones corporativas, los ingenieros asimilan progresivamente el idioma y lo afilian fácilmente, por las destrezas de modelado fragmentado proporcionado por generación, la palabra clave oriunda y mutación de código modular automatiza gran parte de la ejecución de la aplicación y favorece a la omisión de detalles de la infraestructura de destino [19].

La propuesta JDriver, marco de generación automática de controllers para herramientas de fuzzing fundadas en AFL, que consigue crear código de controlador hacia los registros de proceso de métodos, métodos comunes que consiguen procesar registros, su metodología de generación automatizada de controllers utiliza análisis de dependencia con el fin de asemejar a qué campos ingresa el método para posteriormente montar secuencias de métodos con el fin de transformar los valores de campos a los que ingresa de forma que las sucesiones de métodos generados logren cambiar el estado de instancias de clase. JDriver se evaluó en commons-Imaging, que es una librería de imágenes que se usa considerablemente y es brindada por la organización JDriver y Apache para obtener resultados y generaron exitosamente 149 métodos auxiliares que utilizan en la creación de 110 instancias de clases. Al mismo tiempo, crean 99 controllers para abarcar 422 métodos [20].

3. Diseño del modelo para generación de código para C# y SQL Server

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 un modelo para generar código fuente para la arquitectura de tres capas. Para la herramienta utilizamos 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 del modelo 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 aplicar al formulario de generación de código lo podemos ver en la Fig. 1. Para almacenar la base de datos ingresada en la interface de generación se exporta

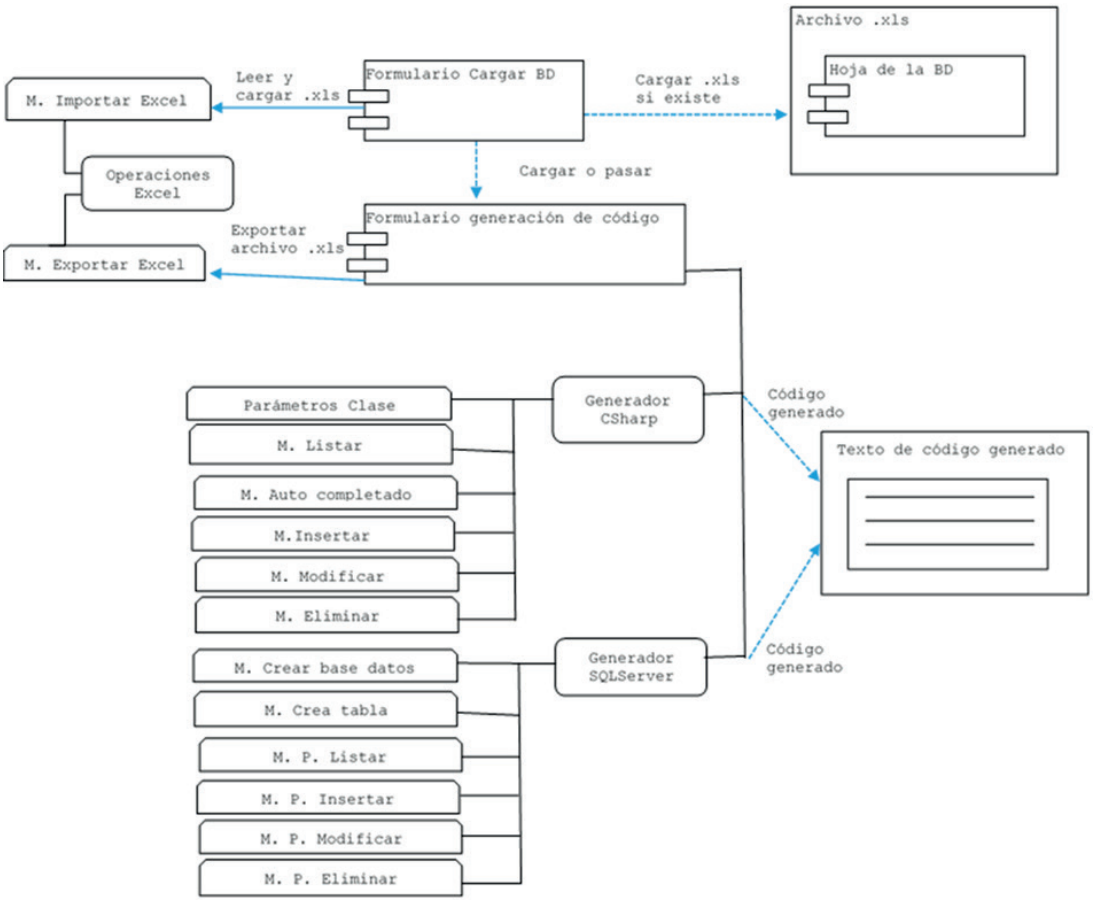

Fig. 1. Diseño del Modelo

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

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 se proporciona los nombres definidos para que se pueda realizar el llamado desde los procedimientos o métodos.

3.1. Componentes

Formulario de inicio

Este formulario se encarga de cargar los archivos .xls como entrada, el archivo tiene el contenido estructura de base de datos, también se puede proporcionar directamente en el formulario la estructura de la base de datos describiendo cada tabla y sus tipos de datos de los atributos, también es posible guardar en un archivo .xls exportando la estructura de la base de datos para su posterior uso en este mismo formulario (Ver Fig. 2).

Formulario generación de código

En este formulario nombramos la base de datos como requisito para poder generar código, agregamos las tablas y las columnas de tablas con sus tipos de dato, ver Fig. 3 y Fig.4. 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 se necesita la base de datos y las tablas ya agregadas dentro del formulario generación de código, posteriormente se debe seleccionar

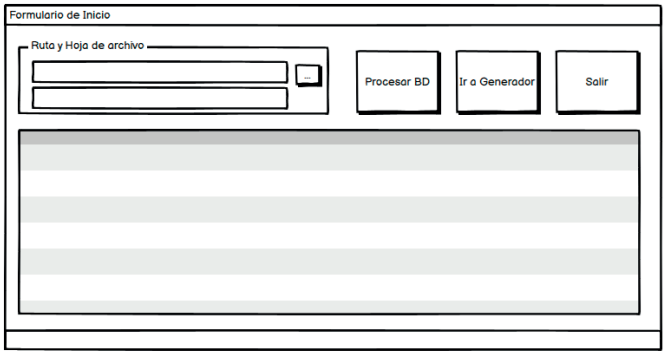

Fig. 2. Formulario de inicio

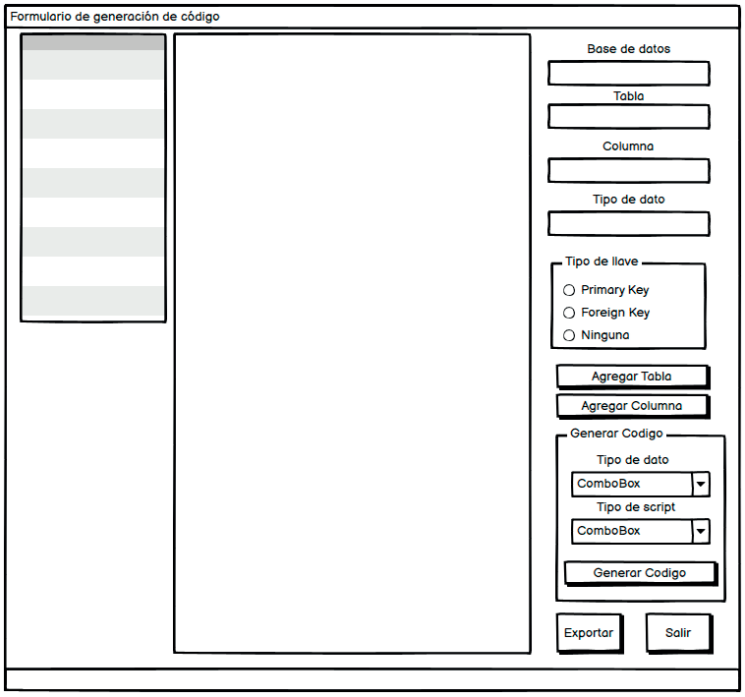

Fig. 3. Formulario generación de código

Fig. 4. Caso de Uso Registrar Tablas

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.

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 tienen la estructura:

Nombre de columna [Tipo de dato / Tipo de llave-Tabla Referencia] (Ver Tabla 1).

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. Generación de código de la herramienta

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

En la Fig. 5 podemos 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.

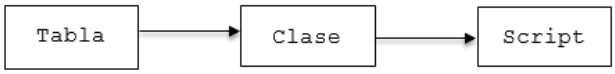

Fig. 5. Conexión para generar código

Implementamos diferentes clases con sus respectivos métodos, para poder generar el código (sintaxis de los scripts).

El proceso de generación presentamos en la figura 6 como diagrama de flujo.

Describiremos brevemente acerca del diseño de la herramienta sobre la capa de datos y la capa lógica de negocio

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.

Capa lógica de negocio

Los scripts propuestos para generar código son generar las clases, parámetros de clases, métodos insertar, modificar, listar y eliminar (Ver Figura 7).

4. Resultados

Para validar nuestra herramienta, hemos utilizado 5 entradas que vienen a ser esquemas de base de datos (Ver Tabla 2), 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 C#, para los parámetros de las clases, métodos (listar, listar auto completando para un combobox, insertar, modificar y eliminar).

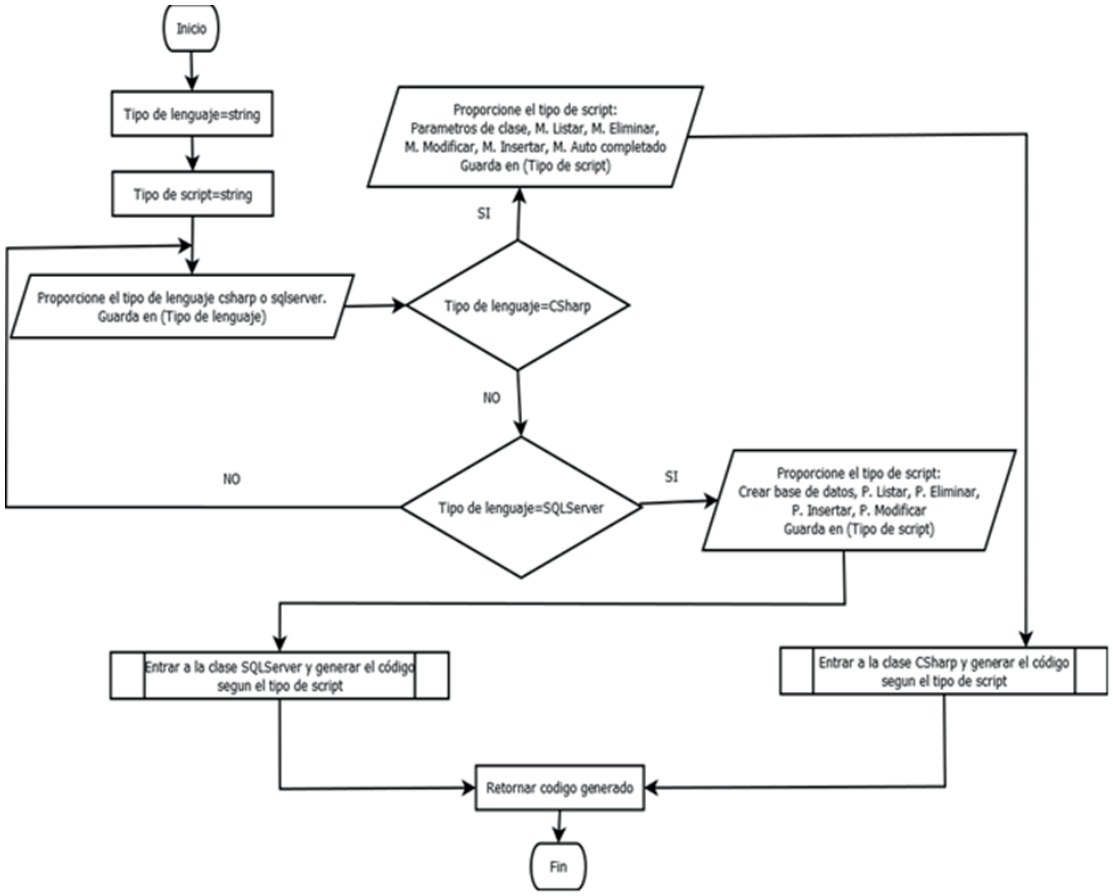

Fig. 6. Diagrama de flujo de generación de código

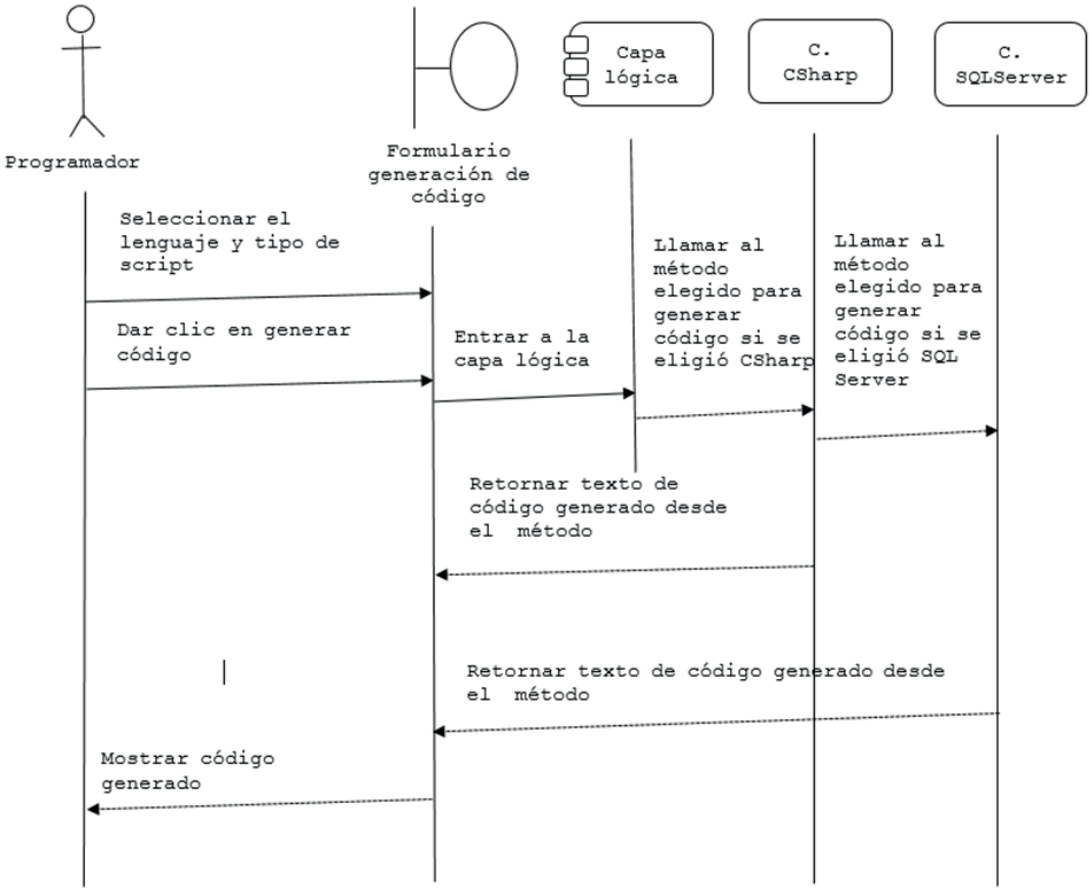

Fig. 7. Diagrama de secuencia de generación de código

En total Utilizamos 51 tablas. A continuación, les mostramos los esquemas de base de datos de la tabla 1.

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, s presenta la tabla 3, generación de código libre de errores.

En la tabla 4 que se presenta, se evidencia las líneas de código generadas con nuestra herramienta CASE GXC, de igual manera se ve el tiempo que lleva generar cada script.

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, presentado de acuerdo a la tabla 4

En la tabla 3 de resultados podemos 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, disminuye 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

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

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	√	√	√	√	√

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

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 (Ver Tabla 5), de todos los esquemas de base de datos y los resultados en global en tiempo en milisegundos y líneas de código.

La tabla 5 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 podemos decir que nuestra 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.

Tabla 5. *Tiempo en milisegundo 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***

5. Discusión

La tasa de error de la investigación [16], muestra 6.00 para la web, 34.24 para androide y 35.20 para iOS. Mientras que nuestra generación de código, 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 nuestra tasa de error es mínima.

La investigación [17], 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 nuestra propuesta la herramienta CASE GXC genera el código un promedio de 6,998 líneas de código por milisegundo, si realizamos la operación de líneas de código generadas por minuto tendríamos 6,998 líneas de código X 60000 milisegundos (un minuto) resulta 419880 líneas de código, por ello nuestra herramienta CASE GXC saca una ligera ventaja en cuestión de tiempo.

En la investigación [15] señala que con su herramienta consiguieron la reducción de esfuerzo en cuanto a la programación de código, con resultados buenos, nuestra 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 [5] 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.

En la propuesta [7] Genera código SQL compatible con MySQL, MariaDB, PostgreSQL y SQL Server, mientras que nosotros generamos código para SQL Server y código para clases en el lenguaje CSharp.

En la investigación [2] enfocan la generación de código de prueba de unidad y se integra a ZeroBrane Studio, nosotros nos enfocamos en generar código para la programación en 3 capas y para ello utilizamos el lenguaje de programación C#, como herramienta para poder programar.

Xiangei y Yongehun [17] generan código para la interfaz web basada 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 [17] se basa en el diseño y nuestra investigación se basa en la funcionalidad, también podemos decir que [17] se basa en frontend y nuestra investigación se basa en backend.

6. Conclusiones y trabajos en futuro

La investigación que se presenta, muestra la generación de código para la arquitectura de 3 capas, mediante la herramienta CASE GXC. Permite generar script para la base de datos, procedimientos almacenados, parámetros de clase y métodos de clases, desde la interface de generación de código, también puede guardar la base de datos en un archivo .xls el cual será restaurado desde la interface principal.

Se muestran resultados favorables en cuanto a las métricas de tiempo, tasa de error y líneas de código.

La herramienta permitirá reducir esfuerzo en tiempo y costos de desarrollo de una arquitectura de 3 capas.

En futuro se trabajará el modelo para otros lenguajes de programación, también proponemos que se debe extender otras funcionalidades en la generación de código fuente y se debe enfocar en otras arquitecturas de software.

Referencias bibliográficas

[1] J. Canós, P. Letelier y C. Penadés, 13 Marzo 2012. [En línea]. Available: http://roa.ult.edu.cu/bitstream/123456789/476/1/TodoAgil.pdf.

[2] Wibowo, J. T. P., Hendradjaya, B., & Widyani, Y. (2015). Unit test code generator for lua programming language. 2015 International Conference on Data and Software Engineering (ICoDSE). doi:10.1109/icodse.2015.7437005.

[3] I. Sommerville, Ingeniería de software, séptima edición, Madrid: Pearson Educación S.A., 2005.

[4] Brunnlieb, M., & Poetzsch-Heffter, A. (2016). Application of Architecture Implementation Patterns by Incremental Code Generation. Proceedings of the 10th Travelling Conference on Pattern Languages of Programs - VikingPLoP ’16. doi:10.1145/3022636.3022647.

[5] Sadaf, S., Athar, A., & Azam, F. (2016). Evaluation of FED-CASE - A Tool to Convert Class Diagram into Structural Coding. 2016 International Symposium on Computer, Consumer and Control (IS3C). doi:10.1109/is3c.2016.57.

[6] Sulistyo, S. & Prinz A. (2009). Recursive modeling for completed code generation. In Proceedings of the 1st Workshop on Behaviour Modelling in Model-Driven Architecture (BM-MDA ’09). Association for Computing Machinery, New York, NY, USA, Article 6, 1–7. DOI:https://doi.org/10.1145/1555852.1555858.

[7] Egea, M., & Dania, C. (2017). SQL-PL4OCL: An Automatic Code Generator from OCL to SQL Procedural Language. 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS). doi:10.1109/models.2017.34.

[8] Galin D. (2018). Software Quality: Concepts and Practice, CASE Tools and IDEs - Impact on Software Quality, 544–560. doi:10.1002/9781119134527.ch26.

[9] De La Torre C., Zorrilla U., Calvarro J. & Ramos M. (2010). Guía de arquitectura n-capas orientada al dominio con .net 4.0. Microsoft Ibérica S.R.L. ISBN: 978-84-936696-3-8.

[10] Presuman R. (2010). Ingeniería del software un enfoque práctico. Septima edición, McGraw-Hill, ISBN: 978-607-15-0314-5.

[11] Vozmediano A. (2017). Java para novatos: Cómo aprender programación orientada a objetos con Java sin desesperarse en el intento. ISBN: 9781521386330.

[12] Joyanes L. (2008). Fundamentos de programación Algoritmos, estructura de datos y objetos. Cuarta edición. ISBN:978-84-481-6111-8.

[13] Li, Z., Jiang, Y., Zhang, X. J., & Xu, H. Y. (2020). The Metric for Automatic Code Generation. Procedia Computer Science, 166, 279–286. doi:10.1016/j.procs.2020.02.099.

[14] J. Han, T. Zhao, Z. Yu, W. Shi and M. Huang, "Design and Implementation of B/S Architecture Code Automatic Generation System," 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2019, pp. 1-4, doi: 10.1109/ICSESS47205.2019.9040840.

[15] Possatto, M. A., & Lucrédio, D. (2015). Automatically propagating changes from reference implementations to code generation templates. Information and Software Technology, 67, 65–78. doi:10.1016/j.infsof.2015.06.009.

[16] Pang, X., Zhou, Y., Li, P., Lin, W., Wu, W., & Wang, J. Z. (2020). A novel syntax-aware automatic graphics code generation with attention-based deep neural network. Journal of Network and Computer Applications, 102636. doi:10.1016/j.jnca.2020.102636.

[17] She, X., & Zheng, Y. (2020). An Automatic Page Code Generation Method Based on Excel Template and Poi Technology. 2020 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS). doi:10.1109/icitbs49701.2020.00123.

[18] Alfonso, J. & Restrepo, F. (2017). Automatic Source Code Generation for Web-based Process-oriented Information Systems.In Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: ENASE, ISBN 978-989-758-250-9, pages 103-113. DOI: 10.5220/0006333901030113

[19] Franky, M. C., Pavlich-Mariscal, J. A., Acero, M. C., Zambrano, A., Olarte, J. C., Camargo, J., & Pinzón, N. (2016). ISML-MDE. International Journal of Web Information Systems, 12(4), 533–556. doi:10.1108/ijwis-04-2016-0025

[20] Huang, Z., & Wang, Y. (2018). JDriver: Automatic Driver Class Generation for AFL-Based Java Fuzzing Tools. Symmetry, 10(10), 460. doi:10.3390/sym10100460.

[21] Julca-Mejia, W., & Calderon-Vilca, H.D. & Cárdenas-Mariño, F.C. 2018, "Evaluation of source code in ACM ICPC style programming and training competitions", Avances en Ingenieria de Software a Nivel Iberoamericano, CIbSE 2018, pp. 298.