View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UPNorte Box Repositorio institucional

FACULTAD DE INGENIERÍA

Carrera de Ingeniería de Sistemas Computacionales

GENERADOR DE CÓDIGO DE FUNCIONALIDADES TIPO CRUD EN LA MANTENIBILIDAD DE SOFTWARE APLICADO A SISTEMAS DE INFORMACIÓN EMPRESARIALES

Tesis para optar el título profesional de:

Ingeniero de Sistemas Computacionales

Autor:
Julio César Becerra Urbina

Asesor:
Mg. Lain Jardiel Cárdenas Escalante

Trujillo - Perú

2019

DEDICATORIA

A mi madre Cecilia Urbina y a mi abuela María Silva, por darme la oportunidad de estudiar y apoyarme en todo momento desde que empecé este camino a convertirme en ingeniero, por quererme mucho y creer en mí, todo se lo debo a ellas.

A mi esposa y mi hijo, por darme la energía y la voluntad de salir adelante siempre, de pensar en un futuro lleno de éxitos donde se combinan la vida profesional y la familiar.

Todos aquellos familiares y amigos que recordé al momento de escribir esto. Ustedes saben quiénes son.

A la Universidad Privada del Norte, por proporcionarme una educación de calidad, acreditada y reconocida a nivel internacional. Que gracias a su equipo me ha formado para ser un profesional de éxito. En especial un agradecimiento mis docentes, en especial el Ingeniero Laín Cárdenas, que con su asesoría y enseñanzas me ayudaron a culminar con éxito el presente proyecto.

AGRADECIMIENTO

En primer lugar, deseo expresar mi agradecimiento a mi asesor de Tesis, Lain Jardiel Cárdenas Escalante, por la dedicación y apoyo que ha brindado a este trabajo, por la paciencia hacia mis tiempos de desarrollo de este informe. Gracias por la confianza ofrecida.

Al docente Isaí Carlos Abanto, por su asesoría para poder representar mis ideas en ecuaciones matemáticas.

A mis amigos, que siempre me han prestado un gran apoyo moral y humano, necesarios en los momentos difíciles de este trabajo y esta profesión.

Gracias por su presencia y apoyo, este trabajo es también el suyo.

A todos, muchas gracias.

Tabla de contenidos

DEDICATORIA 2

AGRADECIMIENTO 3

ÍNDICE DE TABLAS 5

ÍNDICE DE FIGURAS 6

ÍNDICE DE ECUACIONES 7

ABSTRACT 9

CAPÍTULO I. INTRODUCCIÓN 10

CAPÍTULO II. METODOLOGÍA 39

CAPÍTULO III. RESULTADOS 45

CAPÍTULO IV. DISCUSIÓN Y CONCLUSIONES 52

REFERENCIAS 55

DIMENSIONES 67

ÍNDICE DE TABLAS

Tabla1 Comparativo de Metodologías ……………………………………………….25
Tabla 2 Técnicas de Recolección de Datos 39
Tabla 3 Recolección de Datos del Módulo de Gestión de Entregas……………………….41
Tabla 4 Recolección de Datos del Módulo de Gestión de Tipo de Afectación IGV……….41
Tabla 5 Recolección de Datos del Módulo de Gestión de Entregas………………………..42
Tabla 6 Recolección de Datos del Módulo de Gestión de Tipo de Afectación IGV........... 42
Tabla 7 Variables de Hipótesis.. 45
Tabla 8 Comparación de Resultados de Complejidad Ciclomática 46
Tabla 9 Comparación de Resultados de Acoplamiento dentre Objetos 46
Tabla 10 Comparación de Resultados de la Densidad de Comentarios 47
Tabla 11 Comparación de Resultados del Volumen del Programa 47
Tabla 12 Resultados de Facilidad para hacer Pruebas Unitarias...................................... 478
Tabla 13 Resultados de Facilidad para hacer Cambios 479
Tabla 14 Resultados de Índice de Mantenibilidad.. 49

ÍNDICE DE FIGURAS

Figura 1. Comparación de Resultados de Indicadores (I Parte). Fuente: Elaboración propia 48
Figura 2. Comparación de Resultados de Indicadores (II Parte). Fuente: Elaboración propia 48
Figura 3. Comparación de Resultados de Índice de Mantenibilidad. Fuente: Elaboración propia 51

ÍNDICE DE ECUACIONES

Ecuación 1. Facilidad para hacer Pruebas Unitarias........ 43
Ecuación 2. Facilidad para hacer Cambios........ 43
Ecuación 3. Índice de Mantenibilidad........ 43

RESUMEN

En el presente trabajo de investigación se realizó con el objetivo de demostrar que el desarrollo y la aplicación de un software generador de código tipo CRUD favorece la mantenibilidad de código aplicado a sistemas de información empresariales, por esta razón sustentaremos que, escribiendo código con una baja complejidad, componentes desacoplados, respetando las convenciones de nombres y líneas de comentarios descriptivos para los métodos podemos lograr un alto índice de mantenibilidad de software.

Para el desarrollo de esta investigación se recolectó la información a través de un análisis de código minucioso, donde gracias al uso de herramientas de software y métricas de código se pudo establecer cuantitativamente que el proyecto de software desarrollado aumenta la facilidad para realizar pruebas unitarias y realizar cambios al código generado.

Los resultados obtenidos demostraron que la facilidad para hacer pruebas unitarias mejoró sustancialmente, validándose un cambio de FPU >23 (pre-test) a FPU<=8.8 (post-test) y la facilidad para hacer cambios de la misma forma, validándose valores de FC>12000 (pre-test) a valores de FC<4650 (post-test). Estos valores reflejaron que la Mantenibilidad mejoró significativamente obteniendo un incremento del Índice de Mantenibilidad desde un IM<=49.81 (pre-test) a un IM>=82.43.

Con base en lo mencionado, se llegó a la conclusión que utilizar un software que genere una arquitectura de código que respete los estándares de mantenibilidad favorece este criterio de calidad de manera significativa, pero ya depende del programador continuar bajo la línea de buenas prácticas al momento de realizar cualquier tipo de mantenimiento al software.

Palabras clave: CRUD, Generador de Código, Mantenibilidad de Software, Métricas de Software.

ABSTRACT

In the present research work it was carried out with the objective of demonstrating that the development and application of a CRUD code generating software favors the maintainability of code applied to business information systems, for this reason we will support that, writing code with a low complexity, decoupled components, respecting the conventions of names and descriptive comment lines for the methods we can achieve a high index of software maintainability.

For the development of this investigation, the information was collected through a thorough code analysis, where thanks to the use of software tools and code metrics it was quantitatively established that the software project developed increases the ease to perform unit tests and perform Changes to the generated code.

The results showed that the ease of doing unit tests improved, validating a change from FPU> 23 (pre-test) to FPU <= 8.8 (post-test) and the facility to make changes in the same way, validating FC > 12000 (pretest) at FC<4650 (posttest). These values correspond to the Maintainability significantly improved obtaining an increase in the Maintainability Index from an IM <= 49.81 (pre-test) to an IM> = 82.43.

Based on the aforementioned, it was concluded that using software that generates a code architecture that respects the maintainability standards favors this quality criterion significantly, but it is up to the programmer to continue under the line of good practices at the moment to perform any type of maintenance to the software.

Keywords: CRUD, Code Generator, Software Maintainability, Software Metrics.

CAPÍTULO I. INTRODUCCIÓN

1.1. Realidad problemática

Hoy en día el desarrollo del software no se detiene cuando un sistema se entrega, sino que continúa a lo largo de la vida de éste. Es así, que después de distribuir un sistema, inevitablemente debe modificarse, con la finalidad de mantenerlo útil, así como lo menciona (Sommerville, 2011). Se debe tener en cuenta que tanto los cambios empresariales como los de las expectativas del usuario generan nuevos requerimientos para el software existente y es posible que tengan que modificarse partes del software para corregir errores encontrados durante su operación, y de esta forma poder adaptarlo a los cambios en su plataforma de software y hardware, y mejorar su rendimiento u otras características no funcionales.

Dentro de este proceso de mantenimiento se deben realizar cambios en el código, ya sea para agregar nuevas características o para corregir errores, la facilidad para poder identificar el fragmento de código que se desea actualizar y para realizar los cambios depende directamente de la mantenibilidad de código, un tema al que se le ha estado restando importancia, ya que según la Comisión de Promoción del Perú para la Exportación y el Turismo, solo el 10% del software que se desarrolla en el Perú comprende de un área de gestión de la calidad (Promperú, 2019), es decir, un 90% de empresas no contemplan la gestión de Mantenimiento de Software dentro de su proceso de construcción.

Esto se debe principalmente a las formas en que la mayoría de las organizaciones aplican sus presupuestos. Invertir en la mantenibilidad conduce a aumentos de costo a corto plazo, los cuales son mensurables. Por desgracia, las ganancias a largo plazo no pueden medirse al mismo tiempo, de modo que las compañías son renuentes a gastar dinero en un rendimiento futuro incierto

(Sommerville, 2011). El riesgo por falta de mantenibilidad en un sistema puede generar que los tiempos de mantenimiento se alarguen o que no se puedan agregar nuevas características al sistema y en consecuencia que el software sea dado de baja porque ya no puede cumplir con las nuevas expectativas del usuario.

Se han considerado los siguientes estudios como antecedentes sobre los Generadores de Código y la Mantenibilidad de Software:

Los autores Sagrado, Águila, Bosch y Chicano (2017), en la investigación "Impacto de las métricas CK en la refactorización" tuvieron el objetivo de Estimar el impacto de las métricas CK de calidad en la refactorización basándose en la reducción de la entropía. Por ello se recaudó información y datos sobre refactorización, para valorar sobre ellos las métricas respecto a la decisión de re factorizar en función de la reducción en la entropía. Las valoraciones obtenidas se emplean como punto de partida para obtener una lista ordenada de métricas aplicando distintos métodos, incluso en situaciones en las que no todas las métricas pueden valorarse o su valoración no es representativa. Se llegó a la conclusión que de cara a decidir cuándo refactorizar una clase dada se debería tener en cuenta el número de clases con las que se relaciona directamente, así como el número de métodos de la clase y la complejidad ciclomática de estos métodos.

Los autores Pérez, Martínez, Nava, Núñez, Vásquez y Flores (2015), en la investigación "Analizando la Mantenibilidad de Software Desarrollado Durante la Formación Universitaria", tuvieron como objetivo verificar y analizar los programas que realizan los estudiantes de la Universidad Autónoma de San Luis para determinar la mantenibilidad de los SW y con qué facilidad estos pueden ser modificados. Por lo tanto, se analizaron 315 programas que en conjunto estaban constituidos por 573,220 líneas de código que desarrollaron los estudiantes llegando a la conclusión que existe

evidencia empírica de la relación directa entre la formación académica de los estudiantes y las características de sus programas. Esto se deriva del hecho de que una vez que las estructuras académicas son modificadas, los índices de mantenibilidad se ven sumamente afectados a pesar de que las métricas básicas de tamaño, volumen y complejidad permanecen similares.

Los autores López, Cabrera y Valencia (2008), en la investigación "Introducción a la calidad de Software", tuvieron como objetivo describir la importancia que tienen hoy en día la calidad de software y su impacto en las organizaciones para disminuir costos, optimizar los recursos y aumentar la satisfacción general del cliente. Se llegó a la conclusión que el estado actual de la industria del software requiere la aplicación de estándares, modelos y propuestas que se han venido desarrollando en el mundo, es por ello que la adopción de un modelo de calidad por parte de las empresas es un importante indicador de que ésta nueva habilidad ya no es opcional sino obligatoria.

Los autores Erazo, Flores y Pino (2016), en la investigación "Generando productos software mantenibles desde el proceso de desarrollo: El modelo de referencia MANTuS" tuvieron como objetivo incluir atributos de mantenibilidad al producto software durante el proceso de desarrollo con el fin de potenciar esta característica y que sean aplicables en el contexto de empresas desarrolladoras de software. Es por ello que se realizó un estudio de caso que permitió la aplicación del proceso de Construcción de software desde la perspectiva de mantenibilidad. Este estudio fue realizado con ocho desarrolladores de software que ejecutaron cambios a dos versiones del mismo programa: una versión que evidencia la utilización de algunas prácticas base del proceso de construcción del modelo de referencia MANTuS (código B) y otra que no las utiliza (código A). Por lo tanto, llegaron a la conclusión que los

participantes que trabajaron con el código A manifestaron que este no tiene alta mantenibilidad por diferentes aspectos como: la falta de documentación, que dificulta la modificación del código; la gran cantidad de condicionales, que impide realizar cambios mayores de forma sencilla; la ausencia de métodos y funciones. Por el contrario, todos los participantes que trabajaron con el código B concluyeron que este sí tiene alta mantenibilidad debido, a que las funcionalidades estaban separadas correctamente y el código estaba documentado.

Los autores Erazo, Flores y Pino (2016), en la investigación "Análisis y clasificación de atributos de mantenibilidad del software: una revisión comparativa desde el estado del arte" tuvieron como objetivo identificar los atributos que influyen sobre la mantenibilidad de software para conocer qué factores se podrían incluir durante el proceso de desarrollo con el fin de conseguir un producto altamente mantenible. Como resultado de la investigación realizada se obtuvieron atributos clasificados de acuerdo a las sub-características de mantenibilidad, los cuales describen diferentes aspectos que se deben considerar cuando se pretende desarrollar un producto altamente mantenible. Llegando a la conclusión que los atributos que influyen sobre el mayor número de sub-características de mantenibilidad son: acoplamiento, cohesión, complejidad, documentación, encapsulamiento, facilidad de entendimiento, herencia, simplicidad y tamaño.

Los autores Truica, Radulescu, Boicea y Bucur (2015), en la investigación "Performance evaluation for CRUD operations in asynchronously replicated document oriented database" tuvieron como objetivo encontrar la diferencia entre el rendimiento del tiempo para operaciones CRUD para diferentes implementaciones de NoSQL, sistemas orientados a documentos en un entorno distribuido. Por ello se realizó pruebas del tiempo de ejecución de las operaciones CRUD para una sola

instancia de base de datos y para un entorno distribuido con dos nodos se tiene en cuenta y los resultados se comparan con los resultados de pruebas obtenidos para tres sistemas de gestión de bases de datos relacionales: Microsoft SQL Servidor, MySQL y PostgreSQL. Teniendo como resultado las bases de datos orientadas a documentos, CouchDB tiene en general, muy buen tiempo de rendimiento para insertar, actualizar y borrar operaciones, pero se queda atrás cuando se trata modelar los datos. Asimismo, Los sistemas de gestión de bases de datos son aplicaciones que podrían cambiar, con el tiempo, la forma en que manejan las operaciones CRUD y esto podría mejorar o degradar su rendimiento.

El autor Gaitán (2017), en la investigación "Líneas de Productos Software: Generando Código a Partir de Modelos y Patrones" tuvo como objetivo la producción de artefactos de calidad minimizando costos y mejorando la eficiencia en los procesos a través de objetos altamente flexibles y reutilizables. Por ese motivo, desarrolló una propuesta de diseño de un prototipo para generación de código automatizado a partir de modelos MDA (Arquitectura Dirigida por Modelos) y la implementación de Patrones de diseño como MVC para entornos Web. Por lo que llegó a la conclusión que, en la ingeniería de software, el Proveedor de Servicios corresponde a una implementación eficaz de la Reutilización donde los productos software guiados por artefactos altamente flexibles, hacen artefactos mucho más eficientes y con un alto grado de calidad, donde prima la calidad del producto, de ahí que las líneas de producto se deriven de las Factorías de Software.

El autor Sayago (2018), en la investigación "Generador de Código Utilizando el Paradigma de Líneas de Producto Software" tuvo como objetivo dar a conocer las ventajas y utilidad potencial de la línea de productos software construido por analogía y la mejora en el desarrollo de una aplicación web Java EE. Para esto se elaboró un

generador de código utilizando el paradigma de líneas de producto software junto con otras metodologías de desarrollo por analogía. Llegando a la conclusión que el generador de código elaborado cumple con los requerimientos iniciales para los que fue desarrollado, como son automatizar y generar el código para una parte de una aplicación Java EE, que es la capa de negocio y sus respectivos DAO's y EJB's.

Los autores Collejas, Alarcón y Álvarez (2017), en la investigación "Modelos de calidad del software, un estado del arte" tuvieron como objetivo describir las características y estructura de los modelos de software que existen actualmente, a través de revisión de información; dando a conocer modelos que se consideran pioneros o base del desarrollo de otros recientes y poder aplicarlos a las empresas, para certificar y garantizar la calidad de sus productos y procesos. Por lo que concluyeron, que los modelos de calidad clásicos sirven como base de los más recientes, y han permitido que los modelos actuales se consoliden como los más completos con base en la evolución del software, para así optimizar los procesos de las organizaciones y garantizar que se cumple con criterios o estándares que respaldan la calidad de la gestión de procesos del negocio.

El autor Carretero (2013), en la investigación "Entorno de Pruebas de Generadores de Código Automático" tuvo como objetivo la generación automática de UnitTest y Documentación del código en forma gráfica. Para ello, se optó por la creación de un entorno de validación que fuera genérico para realizar pruebas sobre cualquier tipo de generador. Por lo que concluye que con la aparición del paradigma del DSDM surge la generación de código automática a partir de modelos, que trae consigo una nueva necesidad. Los procesos automatizados para la generación automática requieren de nuevas técnicas que validen y garanticen la correcta que el resultado de la generación es válido y correcto desde el punto de vista funcional.

Los autores Irrazábal, Greiner y Dapozo (2015), en la investigación "La refactorización de software basada en valor: Revisión sistemática de la literatura" tuvieron como objetivo mostrar una revisión sistemática de la literatura sobre la refactorización basada en valor y establecer que las funcionalidades de un sistema tienen diferente grado de importancia, y que algunas de ellas aportan más "valor" que otras. Para ello se identificaron 15 artículos primarios de un total de 2300, dando como resultado opiniones opuestas sobre cómo afecta las refactorizaciones a la mantenibilidad. Como conclusión de la revisión se puede afirmar que ningún artículo habla directamente del valor que puede aportar una refactorización al software sobre el que va a ser aplicada. Sin embargo, la mayoría de los estudios sugieren que las refactorizaciones semiautomáticas o asistidas son las más adecuadas para llevar a cabo la refactorización.

Los autores Ancán, Cares y Cravero (2018), en la investigación "Código con mal olor: un mapeo sistemático" tuvieron como objetivo develar el estado actual de los estudios relacionados con los malos olores en el código fuente, considerando principalmente su detección. Los resultados de la revisión indican un predominio de estudios que detectaban malos olores sobre código escrito en Java. Se pudo constatar que la literatura analizada carece de estudios que reporten métodos, herramientas y estrategias de detección en las categorías" abusadores de la orientación a objetos e "inhibidores", mientras que la mayor concentración de artículos está en las categorías de olor denominada "hipertrofias" y "prescindibles", cuyos principales métodos de detección corresponden a métricas y análisis de logs.

Los autores Méndez, Garrido, Overbey, Tinetti y Johnson (2010), en la investigación "Refactorización en Código Fortran Heredado" tuvieron como objetivo mostrar los obstáculos hallados por los programadores en las tareas de mantenimiento

de software heredado escrito en Fortran y dar a conocer la solución a este problema proponiendo la refactorización como técnica para ser aplicada en dichos procesos, intentando, además, desmitificar los prejuicios que Fortran ha adquirido dentro del ámbito de la producción de software. Como resultado se pudo obtener que el espectro de aplicaciones que aún sigue ejecutándose y ha sido escrito en este lenguaje es significativamente importante. Si bien se han utilizado herramientas automatizadas con el fin de mejorar los procesos de mantenimiento, implantación o migración de sistemas heredados escritos en Fortran, estas no han tenido resultado en la contienda con este tipo de software.

El autor Olmedilla (2005), en la investigación "Revisión Sistemática de Métricas de Diseño Orientado a Objetos" tuvo como objetivo revisar la literatura para evaluar el estado actual de la cuestión entorno a las métricas OO de diseño y discernir cuáles son las métricas que se pueden aplicar exclusivamente al diseño. Por otro lado, se pretende saber si las propiedades OO antes mencionadas siguen siendo el objeto principal y directo de medición, o bien si se miden de manera indirecta a través de nuevos elementos, como los patrones, o si por el contrario ha cambiado radicalmente la forma de medir la calidad de un diseño OO. Finalmente concluyó que la mantenibilidad es el indicador de calidad más utilizado, lo cual es lógico dado que la Orientación a Objetos se considera un paradigma que favorece la flexibilidad y la reutilización. Otros indicadores de calidad como la eficiencia o la portabilidad no están presentes en los modelos completos, lo que puede deberse a que los estudios están sesgados por no considerarse diferentes dominios de aplicación. Sólo cuatro estudios establecen un modelo completo de evaluación de calidad del diseño de manera cuantitativa y de ello sólo dos se pueden aplicar exclusivamente a entidades de diseño.

Los autores Ilyas, Hummayun y Nawaz (2013), en la investigacion “A Comparative Study on Code Smell Detection Tools” tuvieron como objetivo realizar un estudio comparativo con respecto a dos instrumentos de detección de malos olores. Para ello, se analizó dos herramientas, JDeodorant e inCode, de detección de olores de código y se presentó una comparación. Por lo que se llegó a la conclusión que la diferencia en los resultados radica en el uso de diferentes enfoques para la detección de olores. Ambas herramientas tienen sus propias eficiencias y deficiencias como la falta de soporte de herramientas en el contexto de la detección y refactorización del olor, las herramientas disponibles no son lo suficientemente maduras. Además del desarrollo de nuevas herramientas, las herramientas actuales también necesitan ser revisadas.

Los autores Greiner, Demchum, Dapozo y Estayno (2010), en la investigación “Una propuesta de solución para automatizar la medición de aplicaciones orientadas a objeto” tuvo como objetivo describir una propuesta de solución orientada a contribuir a la calidad del producto software automatizando el proceso de medición de aplicaciones orientadas a objetos. De esta manera se pudo concluir que los modelos de calidad consideran la medida de la calidad del software como una exigencia para la obtención de productos y servicios que satisfagan las necesidades y expectativas de los usuarios. Por lo tanto, esta solución contribuye a facilitar la medición del software, dado que propicia un marco de trabajo que permite a los desarrolladores detectar situaciones que puedan afectar la comprensión y mantenibilidad del software, como así también, generar un registro histórico de los valores de las métricas que permita realizar predicciones y una adecuación del proceso de medición al contexto organizacional específico.

Los autores Vásquez, Moreno y García (2001), en la investigación "Métricas orientadas a objetos" tuvieron el objetivo ofrecer una panorámica del estado actual de la medición de productos software y en particular en orientación a objetos, enfoque que ha alcanzado en la actualidad altas cotas de popularidad debido a los beneficios que promete: rápido desarrollo, reutilización, gestión de la complejidad, etc. Para ello se realizó una breve descripción del enfoque orientado a objetos, de igual manera se detalló las características principales del software orientado a objetos, se presenta una taxonomía de métricas OO en tres dimensiones: una la propiedad que se quiere medir (cohesión, acoplamiento, herencia, etc.), otra el nivel a que se quiere medir (variable, método, clase, Métricas orientadas a objetos DPTOIA-IT-2001-002 2 sistema...) y otra la etapa del ciclo de vida en que se mide (análisis, diseño, implementación, prueba...). Se llegó a la conclusión que habiendo repasado las métricas consideradas más representativas y Las de otros autores, desde una perspectiva basada en las características específicas del enfoque orientado a objetos, más que en el análisis particular del conjunto de métricas propuesto por un solo autor estas llevan una mayor dificultad en la comprensión de métricas, pues algunas miden las mismas propiedades, pero desde diferentes perspectivas.

Por otro lado, el presente proyecto se justifica por lo siguiente:

Existen programas generadores de código, pero muchos se enfocan solo en cumplir en lograr la persistencia de datos, es por esta razón que existe la importancia del siguiente aporte al campo del desarrollo de software.

Es conveniente el uso de tecnologías de software para ayudar al desarrollador en el proceso de programación, yendo más allá de las facilidades que otorgan los IDE's y utilizar herramientas que automaticen la codificación, esto constituye una justificación aplicativa o práctica.

Además, el producto de software de este proyecto, no solo ayuda a mejorar los tiempos de desarrollo, sino que aplica y fomenta las buenas prácticas de convenciones de nombres de variables y estándares de calidad de código, brindándole la importancia que estas merecen, estableciendo así una justificación valorativa.

Finalmente, cabe resaltar que a la enseñanza de calidad de código en su rama de mantenibilidad no se le ha dado la importancia necesaria, siendo los cursos asociados a ella solo teóricos y careciendo de horas de laboratorio, donde el docente debe hacer el esfuerzo por explicar en pizarra los paradigmas de codificación.

Adicionalmente, el proyecto de investigación presentó las siguientes limitaciones:

Falta información bibliográfica sobre generadores de código, se superó revisando tesis relacionadas al tema y con investigación propia sobre los algoritmos para acceder a las propiedades de una Base de Datos relacional.

La mayoría de los sistemas de verificación de Calidad de Código son de pago y/o no se pueden modificar, se superó este inconveniente buscando en GitHub software libre y de código abierto para su posterior modificación y adaptación.

Conjuntamente, en este trabajo de investigación recogemos conceptos como:

Crud. El concepto CRUD está relacionado con la gestión de datos digitales y agrupa las funciones requeridas por un usuario que tiene la intención de crear y gestionar datos. Además, se debe tener en cuenta que dicha gestión está basada en operaciones adaptadas a los requerimientos del sistema y de usuario. Es así, que, para los expertos en el tema, las operaciones son herramientas de acceso necesarias para verificar, por ejemplo, los problemas de la base de datos; mientras que para los

usuarios CRUD significa: crear una cuenta (create) y utilizarla (read), actualizarla (update) o borrarla (delete) en cualquier momento (1 & 1 IONOS, 2017).

Las funciones CRUD son las que distribuyen la carga entre los diferentes puntos del sistema de información, tanto localmente (p.ej. disco local), como remotamente (base de datos, LDAP) (Medina, 2014).

Asimismo, estas funciones también se consideran para describir el ciclo de vida de los datos. Diferentes usuarios pueden tener diversos ciclos de CRUD basados en los requisitos del sistema. Un cliente, por ejemplo, podría tener la capacidad de crear una cuenta, recuperarla al regresar a un sitio web, actualizar la información de facturación o eliminarla si es necesario. Un gerente de operaciones, por el contrario, puede crear registros de productos, llamarlos según sea necesario, modificar los detalles del empaque o de las materias primas o eliminarlos si el producto fue descontinuado (Rouse, 2008).

Metodología XP. Se basa en una alimentación continua entre el cliente y el equipo de desarrollo; una comunicación fluida entre todos los participantes y la simplicidad en las soluciones implementadas y actitud para enfrentar los cambios. Además, describe minuciosamente las prácticas de desarrollo que se van a utilizar, por ejemplo: lenguaje de programación, refactorización, pruebas unitarias, etc. (Laínez, 2015). Existen ciertas prácticas de desarrollo, así tenemos:

- **Programación en parejas:** es preferible que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto, debido a que la mayor calidad del código escrito es más importante que la posible pérdida de productividad inmediata.
- **Refactorización del código**, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad, pero sin modificar su

comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.

- **Pruebas unitarias continuas**, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación. Véase, por ejemplo, las herramientas de prueba JUnit orientada a Java, DUnit orientada a Delphi, NUnit para la plataforma.NET o PHPUnit para PHP. Estas tres últimas inspiradas en JUnit, la cual, a su vez, se insipiró en SUnit, el primer framework orientado a realizar tests, realizado para el lenguaje de programación Smalltalk. (Linarez, Querales, Graterol, Figueroa, Noguera y Ponte, 2012).

La metodología XP se caracteriza por:

- Metodología liviana de desarrollo de software.
- Conjunto de prácticas y reglas empleadas para desarrollar software.
- Basada en diferentes ideas acerca de cómo enfrentar ambientes muy cambiantes.
- En vez de planificar, analizar y diseñar para el futuro distante, hace todo esto un poco cada vez, a través de todo el proceso de desarrollo.
- Metodología basada en prueba y error.
- Fundamentada en Valores y Prácticas (Ruiz, 2017).

De la misma manera, existen otras características importantes, como:

- Se considera al equipo de proyecto como el principal factor de éxito del proyecto.
- Software que funciona por encima de una buena documentación.
- Interacción constante entre el cliente y el equipo de desarrollo.
- Planificación flexible y abierta.

- Rápida respuesta a cambios (Calvo, 2018).

Valores XP. Cada uno de estos valores mostrados a continuación se usa como un motor para actividades, acciones y tareas específicas de XP como:

- **Simplicidad:** XP propone el principio de hacer la cosa más simple que pueda funcionar, en relación al proceso y la codificación. Es mejor hacer hoy algo simple, que hacerlo complicado y probablemente nunca usarlo mañana.
- **Comunicación:** Algunos problemas en los proyectos tienen origen en que alguien no dijo algo importante en algún momento. XP hace casi imposible la falta de comunicación.
- **Realimentación:** Retroalimentación concreta y frecuente del cliente, del equipo y de los usuarios finales da una mayor oportunidad de dirigir el esfuerzo eficientemente (Cevallos, 2015).

Entregables. Podemos encontrar definiciones como:

- **Plan de entregas ("Release Plan")** el cronograma de entregas muestra qué historias de usuario serán agrupadas para conformar una entrega, y el orden de las mismas. Este cronograma será el resultado de una reunión entre todos los actores del proyecto: cliente, desarrolladores, gerentes, etc. (Molina, Zea, Redrován, Loja, Valarezo y Honores, 2018).
- **Historias de usuario:** Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software, es decir se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. Su función principal es identificar problemas percibidos, proponer soluciones y

estimar el esfuerzo que requieren implementar las ideas propuestas (Kendall y Kendall, 2005).

Además, la imposición de este concepto permite añadir una visión más amplia al proceso de desarrollo (Sommerville, 2005).

Las siguientes son algunas de las ventajas de trabajar con historias de usuario.

- Son deseos o necesidades muy concretas que se centran en partes definidas del proyecto.
- Contienen información de una fuente externa que ve nuestro producto sin prejuicios, el usuario potencial o real.
- Fomentan el trabajo en grupo en busca de soluciones.
- Permiten estimar el esfuerzo que va a requerir desarrollar una idea.

Vamos a conocer las ventajas y desventajas que posee la metodología XP frente a otro tipo de metodologías. Así tenemos:

Ventajas

- Se consiguen productos usables con mayor rapidez.
- El proceso de integración es continuo, por lo que el esfuerzo final para la integración es nulo. Se consigue integrar todo el trabajo con mucha mayor facilidad.
- Se atienden las necesidades del usuario con mayor exactitud. Esto se consigue gracias a las continuas versiones que se ofrecen al usuario.
- Se consiguen productos más fiables y robustos contra los fallos gracias al diseño de los test de forma previa a la codificación.
- Obtenemos código más simple y más fácil de entender, reduciendo el número de errores.

- Gracias al "refactoring" es más fácil el modificar los requerimientos del usuario.
- Conseguimos tener un equipo de desarrollo más contento y motivado. Las razones son, por un lado, el que la XP no permite excesos de trabajo (se debe trabajar 40 horas a la semana), y por otro la comunicación entre los miembros del equipo que consigue una mayor integración entre ellos.

Desventajas

- Resulta muy complicado planear el proyecto y establecer el costo y la duración del mismo.
- No se puede aplicar a proyectos de gran escala, que requieran mucho personal, a menos que se las subdivida en proyectos más pequeños.
- Es más complicado medir los avances del proyecto, pues es muy complicado el uso de una medida estándar.
- Altas comisiones en caso de fallar (Fernández, 2010).

Cuadro comparativo de Metodologías. A continuación, se muestra un cuadro que compara las distintas metodologías:

Tabla 1
Comparativo de Metodologías

	Metodología		
Criterio Comparación	**RUP**	**XP**	**SCRUM**
Tipo de Framework	Análisis, diseño, implementación y documentación de sistemas orientados a objetos.	Basado en la adaptabilidad, mayor flexibilidad, dinámico y funcional.	Gestión y desarrollo de software, basado en un proceso iterativo e incremental.

Tipo de Revisión	En cada fase se realiza una o más iteraciones, perfeccionando así los objetivos. Si no se termina una fase no se continúa con la siguiente.	Se debe integrar como mínimo una vez al día y realizar las pruebas sobre la totalidad del proceso.	Breve revisión diaria, donde se describen 3 cuestiones: Trabajo realizado el día anterior. Trabajo previsto a realizar. Cosas que puede realizar o impedimentos.
Objetivos	Orientado a objetos que establece las bases, plantillas y ejemplos para todos los aspectos y fases de desarrollo de software.	Basada en dar prioridad a trabajos con resultado directo. Satisfacción del cliente. Trabajo en grupo. Actuar sobre variables: Coste, Tiempo, Calidad y Alcance.	Indicado para proyectos en entornos complejos. Obtener resultados prontos. Requisitos cambiantes. Innovación y competitividades fundamentales.
Tipos de Desarrollo	Proceso iterativo incremental por fases: Inicio. Elaboración. Construcción. Transición.	Liviana y adaptable. Desarrollo por fases: Planificación del proyecto. . Diseño. . Codificación. . Pruebas.	Desarrollo simple, que requiere trabajo duro. Control de forma empírica y adaptable a la evolución del proyecto.
Facilidad de Uso	Dirigido por Casos de Uso. Establecimiento temprano de una buena arquitectura. Iterativo e incremental. Incremental, el trabajo se divide en mini proyectos.	Orientada para Pequeños o medianos equipos. No apto para mucho personal. Requisitos con probabilidad de cambiar.	Modelo Adaptable. Construcción incremental basada en iteraciones. No existe trabajo con diseños o abstracciones.

Fuente: Elaboración propia

Mantenibilidad de Software. Al hablar de mantenibilidad, se puede decir que es la capacidad del software para ser modificado (correcciones, mejoras, adaptaciones, etc.) (Granados, 2015).

En la actualidad las empresas han estado demandando sistemas de software que vayan de acuerdo a la mayoría de los requerimientos de los usuarios y a los estándares de calidad, para lo cual ha incrementado el tamaño de los sistemas de software y su complejidad. Este hecho logra que los sistemas se vuelvan difíciles de

mantener y por ende incrementar costos. En este sentido, el glosario de términos del estándar de la IEEE de ingeniería de software define mantenibilidad como la facilidad con la cual un sistema de software o componente puede ser modificado para corregir fallas; mejorar su desempeño u otros atributos; o adaptarse a cambios del ambiente.

Asimismo, la mantenibilidad es un factor altamente significativo en el éxito económico del producto de software. Es un atributo importante de calidad, pero es difícil de estimar, ya que involucra predicciones acerca de cambios futuros. Por lo que, si el modelo de predicciones de mantenibilidad es exacto, entonces el diseño de correcciones podría ser adoptado y ayudar a reducir esfuerzos futuros de mantenibilidad. Estos lineamientos se deben seguir en todas las etapas de desarrollo de software (ingeniería de requerimientos, diseño, programación, pruebas, etc.). Entonces podemos decir que el objetivo del ingeniero de software es producir programas funcionales (que hagan lo que se supone deben hacer), correctos (sin defectos) y mantenibles (susceptibles de evolucionar ante los cambios manteniendo las dos cualidades anteriores) y que cumplan con los más altos estándares de calidad en programación (Pérez, Martínez, Nava, Núñez, Vásquez y Flores, 2015).

Métricas de Software. El glosario de términos del estándar de la IEEE de ingeniería de software las define como una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado. Es esencial si es que se desea realmente conseguir la calidad en software. Las métricas son la maduración de una disciplina, que van a ayudar a la evaluación de los modelos de análisis y de diseño, en donde proporcionarán una indicación de la complejidad de diseños procedimentales y de código fuente, y también ayudaran en el diseño de pruebas más efectivas; es por eso que propone un proceso de medición, el cual se puede caracterizar por cinco actividades:

- **Formulación:** La obtención de medidas y métricas del software apropiadas para la representación de software en cuestión.
- **Colección:** El mecanismo empleado para acumular datos necesarios para obtener las métricas formuladas.
- **Análisis:** El cálculo de las métricas y la aplicación de herramientas matemáticas.
- **Interpretación:** La evaluación de los resultados de las métricas en un esfuerzo por conseguir una visión interna de la calidad de la representación.

- **Realimentación:** Recomendaciones obtenidas de la interpretación de métricas técnicas trasmitidas al equipo de software (Rodríguez y Martínez, 2006).

Se han propuesto cientos de métricas para el software, pero no todas proporcionan un soporte práctico para el desarrollador de software. Algunas demandan mediciones que son demasiado complejas, otras son tan esotéricas que pocos profesionales tienen la esperanza de entenderlas y otras violan las nociones básicas intuitivas de lo que realmente es el software de alta calidad. Existen una serie de características que deberían acompañar a las métricas efectivas del software. Dichas características son:

- Simples y fáciles de calcular.
- Empírica e intuitivamente persuasivas.
- Consistentes y objetivas.
- Consistentes en el empleo de unidades y tamaño.
- Independientes del lenguaje de programación.
- Eficaz mecanismo para la realimentación de calidad (Sicilia, 2009).

Por consiguiente, podemos hacer la siguiente clasificación

- **Métrica de Complejidad Ciclomática de McCabe:** la complejidad ciclomática se basa en el recuento del número de caminos lógicos individuales contenidos en un programa. Para calcular la complejidad del software, Thomas McCabe utilizó la teoría y flujo de grafos. Para hallar la complejidad ciclomática, el programa se representa como un grafo, y cada instrucción que contiene, un nodo del grafo. Las posibles vías de ejecución a partir de una instrucción (nodo) se representan en el grafo como aristas (Sicilia, 2009).
  También se afirma que McCabe demuestra experimentalmente que cuando la complejidad ciclomática de un módulo excede de 10 ($V(G)>10$), éste llega a ser demasiado complejo para poderlo probar y mantener (Amo, Martínez y Segovia, 2005).
- **Métrica de Acoplamiento entre Objetos (CBO) de Chidamber y Kemerer:** el acoplamiento entre objetos de una clase es el número de clases

a las cuales una clase está ligada, sin tener con ella relaciones de herencia. Hay dependencia entre dos clases cuando una de ellas usa métodos o variables de la otra clase. Es consistente con las tradicionales definiciones de acoplamiento: medida del grado de interdependencia entre módulos. Cuanto más independiente es un objeto, más fácil es reutilizarlo en otra aplicación. Al reducir el acoplamiento se reduce la complejidad, se mejora la modularidad y se promueve el encapsulamiento. Una medida del acoplamiento es útil para determinar la complejidad de las pruebas necesarias de distintas partes de un diseño. Cuanto mayor sea el acoplamiento entre objetos más rigurosas han de ser las pruebas (Chidamber y Kemerer, 1994)

- **Métrica de Índice de la Mantenibilidad (IM):** El índice de mantenibilidad (IM) es un modelo de mantenibilidad de software que fue propuesto por Omán y Hagemeister en la Universidad de Idaho (1991), este modelo consiste en un número de métricas calculadas fácilmente, y que es capaz de predecir fácil y rápidamente la mantenibilidad de un producto de software. El Índice de mantenibilidad está dado como una ecuación polinómica compuesta por variables predictoras. A través de una serie de estudios se ha demostrado que hay una fuerte correlación entre la mantenibilidad de software y las variables predictoras como Complejidad Ciclomática de McCabe, Volumen Halstead, Número de Líneas de código, y el Número de comentarios de código (Sánchez, 2016).

Java. Hablar de Java es referirse a uno de los lenguajes de programación que tiene mayor importancia sobre los demás. Java está en todas partes, desde portátiles hasta centros de datos, desde consolas para juegos hasta súper computadoras, desde teléfonos móviles hasta Internet, incluso existen muchas aplicaciones y sitios web que no funcionan a menos que tengan Java instalado (Sznajdleder, 2013). Aunque ha sido fuertemente ligado a Internet, es importante recordar que Java es un lenguaje de programación de uso general. Las innovaciones y desarrollo de los lenguajes de programación ocurren por dos razones principales:

- Para adaptarse a los cambios en ambientes y usos.
- Para implementar mejoras en el arte de la programación.

Por otro lado, se comenta que Java está siendo utilizado en muchos ámbitos y en variedad de tecnologías, desde el chip de una tarjeta de crédito hasta un servidor de la más alta gama. Por ejemplo, la cantidad de memoria disponible en el chip de una tarjeta de crédito y la de un servidor es muy distinta, por lo que habrá que tenerlo en cuenta a la hora de desarrollar las aplicaciones" (Ordax y Ocaña, 2012).

Java es mucho más que un simple lenguaje de programación y para conocerlo aún más, se debe tener en cuenta sus principales características:

- **Orientado a objetos:** se refiere a que el método de programación es flexible, pues facilita todo el ciclo de vida de un software, empezando por el análisis y diseño, seguido de la implementación hasta el mantenimiento.
- **Distribuido:** Java dispone de una gran cantidad de clases que logran la comunicación entre programas ejecutados en ordenadores remotos conectados en red evitando la aparición de otros programas.
- **Concurrente:** permite el desarrollo de programas correlativos para conseguir un mejor aprovechamiento del procesador cuando es necesario realizar varias tareas al mismo tiempo.
- **De Alto Rendimiento:** sobre todo con la aparición de hardware especializado y mejor software.
- Requiere del aprendizaje de una gran cantidad de técnicas con la finalidad de que el lector pueda desarrollar aplicaciones flexibles (Garrido, 2015).

Spring Framework. Spring es un framework que da soporte al desarrollo de aplicaciones empresariales en Java. Proporciona una serie de características, entre las que tenemos que destacar la inyección de dependencias, la gestión de transacciones, el soporte para pruebas automatizadas y el soporte orientado a aspectos de programación. Se considera un software libre, y se puede utilizar en contenedores web, dispensando servidores de aplicaciones JEE como Glassfish y JBoss, así como para aplicaciones de escritorio (Méndez, 2016).

Algunos de los beneficios de implementar Spring Framework, son:

- Spring está organizado de forma modular. A pesar de la cantidad de paquetes y clases que tiene, solo debemos ocuparnos de aquellos que necesitemos para nuestro desarrollo e ignorar el resto.

- Utiliza algunas de las tecnologías existentes, como varios frameworks ORM, JEE, temporizadores Quartz y JDK, frameworks de registro y otras tecnologías de visualización.
- Probar una aplicación escrita con Spring es un proceso simple, porque el código dependiente del entorno se traslada a este framework. Además, mediante el uso de JavaBeanstyle, se vuelve más fácil utilizar la inyección de dependencia para hacer pruebas, para ello podemos hacer uso de datos dummies o mocks, para ver las respuestas.
- El framework web de Spring es un framework MVC web bien diseñado, que proporciona una excelente alternativa a los frameworks web como Struts u otros frameworks web sobre diseñados o menos populares.
- Spring proporciona una API para traducir excepciones específicas de la tecnología (como por ejemplo las generadas por JDBC, Hibernate o JDO) en excepciones consistentes y no verificadas.
- Los contenedores de IoC (Inversion of Control) tienden a ser livianos, especialmente cuando se comparan con los Enterprise JavaBeans (EJB). Esto es ideal para desarrollar y desplegar aplicaciones en máquinas con memoria y recursos limitados.
- Spring proporciona una interfaz de gestión de transacciones coherente que puede reducirse a una transacción local (utilizando una única base de datos) y ampliarse a transacciones globales.
- Permite separar el registro, la auditoría, las transacciones declarativas, la seguridad, el almacenamiento en caché, de la lógica comercial a través de la AOP (Programación Orientada a Aspectos).
- Cuenta con plantillas para diversas tecnologías entre la cuales podemos destacar las siguientes: JDBC, Hibernate y JPA, de forma tal que no hay necesidad de escribir un código extenso, ya que con estas plantillas simplifica el trabajo en cuanto a los pasos básicos a implementar de estas tecnologías (Muradas, 2018).

Spring Boot y Maven. Spring Boot es una de las tecnologías dentro del mundo de Spring de las que más se utiliza. Para entender el concepto primero se debe entender sobre cómo se construyen aplicaciones con Spring Framework (Álvarez, 2016)

Fundamentalmente existen tres pasos a realizar. El primero es crear un proyecto Maven y descargar las dependencias necesarias. En segundo lugar, desarrollamos la aplicación y en tercer lugar la desplegamos en un servidor. Únicamente el paso dos es una tarea de desarrollo. Los otros pasos están más orientados a infraestructura.

SpringBoot nace con la intención de simplificar los pasos 1 y 3 y que el programador se pueda centrar en el desarrollo de su aplicación. ¿Cómo funciona? El enfoque es sencillo y lo entenderemos realizando un ejemplo. Para ello nos vamos a conectarnos al asistente de Boot.

El asistente es intuitivo, se elige el package al que se quiere que pertenezcan las clases, se selecciona el nombre del proyecto y por último las dependencias. Eso sí ya no se trata de elegir jar por jar sino que tipo de aplicación se desea.

Esta clase es la encargada de arrancar la aplicación de Spring a diferencia de un enfoque clásico no hace falta desplegarla en un servidor web ya que Spring Boot provee de uno.

La ejecución de la aplicación es como a nivel de consola, esto abrirá un servidor web y accederemos a la URL, por defecto la 8080.

JPA (Java Persistence Api). JPA es la propuesta estándar que ofrece Java para implementar un Framework Object Relational Mapping (ORM), que permite interactuar con la base de datos por medio de objetos, de esta forma, JPA es el encargado de convertir los objetos Java en instrucciones para el Manejador de Base de Datos (MDB). Cuando empezamos a trabajamos con bases de datos en Java lo primero que nos enseñan es a utilizar el API de JDBC el cual nos permite realizar consultas directas a la base de datos a través de consultas SQL nativas. JDBC por mucho tiempo fue la única forma de interactuar con las bases de datos, pero representaba un gran

problema y es que Java es un lenguaje orientado a objetos y se tenía que convertir los atributos de las clases en una consulta SQL como SELECT, INSERT, UPDATE, DELTE, etc. lo que ocasionaba un gran esfuerzo de trabajo y un provocaba muchos errores en tiempo de ejecución, debido principalmente a que las consultas SQL se tenían que generar frecuentemente al vuelo (Blancarte, 2018).

Hibernate. Es una herramienta de mapeo objeto-relacional (ORM) para la plataforma Java (y disponible también para .Net con el nombre de NHibernate) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones (Ríos, 2005).

Servicios web tipo Rest. Representational State Transfer es un conjunto de técnicas orientadas a crear servicios web en los que se renuncia a la posibilidad de specificar la interfaz de los servicios de forma abstracta a cambio de contar con una convención que permite manejar la información mediante una serie de operaciones estándar. La idea detrás de REST es el desarrollo de servicios orientados a la manipulación de recursos. En un servicio REST típico, tenemos una URL por cada recurso (documento, entidad, etc.) que gestionamos y que realiza una tarea diferente sobre dicho recurso en función del método HTTP que utilicemos (Brito, 2009).

JSON. Java Script Object Notation (JSON) es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript. Es comúnmente utilizado para transmitir datos en aplicaciones web (por ejemplo: enviar algunos datos desde el servidor al cliente, así estos datos pueden ser mostrados en páginas web, o vice versa). Asimismo, JSON está constituido por dos estructuras:

- **Una colección de pares de nombre/valor.** En varios lenguajes estos son conocidos como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- **Una lista ordenada de valores.** En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

En JSON, se presentan de estas formas:

Un objeto es un conjunto desordenado de pares nombre/valor. Un objeto comienza con {(llave de apertura) y termine con} (llave de cierre). Cada nombre es seguido por: (dos puntos) y los pares nombre/valor están separados por, (coma).

Un arreglo es una colección de valores. Un arreglo comienza con [(corchete izquierdo) y termina con ']' (corchete derecho). Los valores se separan por (coma).

- Un valor puede ser una cadena de caracteres con comillas dobles, o un número, o true o false o NULL, o un objeto o un arreglo. Estas estructuras pueden anidarse.
- Una cadena de caracteres es una colección de cero o más caracteres Unicode, encerrados entre comillas dobles, usando barras divisorias invertidas como escape. Un carácter está representado por una cadena de caracteres de un único carácter. Una cadena de caracteres es parecida a una cadena de caracteres C o Java.
- Un número es similar a un número C o Java, excepto que no se usan los formatos octales y hexadecimales (Sepúlveda, 2017).

PostgreSQL. Es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiarles a otras bases de datos comerciales.

Por otro lado, también se puede decir que PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando.

La capacidad de SQL es tal como se describe a continuación:

- Llaves primarias (primary keys) y foráneas (foreign keys).
- Check, Unique y Not null constraints.
- Restricciones de unicidad postergables (deferrable constraints)
- Columnas auto-incrementales.
- Índices compuestos, únicos, parciales y funcionales en cualquiera de los métodos de almacenamiento disponibles, B-tree, R-tree, hash ó GiST.
- Sub-selects.
- Consultas recursivas.
- Funciones 'Windows'.
- Joins.
- Vistas (views).
- Disparadores (triggers) comunes, por columna, condicionales.
- Reglas (Rules).
- Herencia de tablas (Inheritance).
- Eventos LISTEN/NOTIFY (Zea, Molina y Redrován, 2017).

Arquitectura Modelo Vista Controlador. El Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo:

- **El Modelo** que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
- **La Vista**, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
- **El Controlador**, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno (Eslava, 2013).

Spring Data JPA DataTable. Este proyecto es una extensión del proyecto Spring Data JPA para facilitar su uso con jQuery plugin DataTables con el procesamiento del lado del servidor habilitado. Esto le permitirá manejar las solicitudes de Ajax enviadas por DataTables para cada sorteo de la información en la página (es decir, cuando página, ordena, busca, etc.) desde Spring @RestController (Arrachequesne, 2018).

Herramientas para medir el software.

- **Tateti Software:** Tateti es un software de código abierto desarrollado en java por Nina Satragno, una desarrolladora argentina que ha trabajado en Google. Tateti Software permite medir un código fuente desarrollado en java con las métricas de Halstead, las cuales permiten obtener el tamaño del vocabulario, la longitud, el volumen, el nivel de dificultad y el esfuerzo de un método, clase, paquete o todo el proyecto.
- **CK4J Software:** Es una herramienta de software que permite medir un compilado en .jar o .war de java con las métricas de Chidamber y Kemerer lo que facilita identificar LCOM: falta de cohesión de métodos, NOC: Número de Hijos, WMX: Métodos ponderados por clase, CBO: Acoplamiento entre objetos.

Definición de términos básicos

Software Generador de Código de Funcionalidades CRUD: es un programa informático que permite, bajo ciertos parámetros, generar código tipo CRUD para el lenguaje de programación para el que fue creado (Java, .NET, PHP, etc). CRUD es el acrónimo de Create, Read, Update, Delete en inglés, que traducido al español sería Crear, Leer, Actualizar y Eliminar y es la base de todo software de aplicación empresarial como sistemas de ventas, software de recepción de pedidos, software para hoteles, colegios, entre otros.

Mantenibilidad de Software: El (Institute of Electrical and Electronics Engineers, 1990) define mantenibilidad como: "La facilidad con la que un sistema o componente software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno", a pesar de que la Mantenibilidad puede llegar a presentarse como subjetiva y en función al programador que realice el mantenimiento, esta se puede medir bajo ciertas reglas que ayuden a cuantificar su nivel de Mantenibilidad.

1.2. Formulación del problema

¿De qué manera la aplicación de un software generador de funcionalidades tipo CRUD influye en la mantenibilidad de los sistemas de información empresariales?

1.3. Objetivos

1.3.1. Objetivo general

Determinar la influencia en la mantenibilidad de código de los sistemas de información empresariales, mediante la aplicación de un software generador de código de funcionalidades tipo CRUD.

1.3.2. Objetivos específicos

- Definir la influencia de un software generador de código en el nivel de facilidad para aplicar pruebas unitarias al código generado.
- Determinar la influencia de un software generador de código en el nivel de facilidad de modificación o cambios en el código generado.

1.4. Hipótesis

La aplicación de un Software generador de código de funcionalidades tipo CRUD influye positivamente en la mantenibilidad de los Sistemas de Información Empresariales, facilitando las pruebas unitarias y los cambios.

CAPÍTULO II. METODOLOGÍA

2.1. Tipo de investigación

La investigación es aplicada y preexperimental.

2.2. Población y muestra

Para la población se tomará en cuenta los módulos CRUD de un software para Facturación Electrónica y un software para Repartos.

Para la muestra se consideran dos sistemas, uno de Facturación Electrónica y otro de Repartos Courier.

2.3. Técnicas e instrumentos de recolección de y análisis de datos

Tabla 2
Técnicas de Recolección de Datos

Variables	Indicador a medir	Instrumentos
Mantenibilidad de Software	Complejidad Ciclomática	Análisis de Código
	Acoplamiento entre Objetos	
	Densidad de Comentarios en el código	
	Volumen del programa	

Fuente: Elaboración propia

2.4. Procedimiento

Para la presente investigación se tomaron las muestras de CRUD de dos sistemas empresariales, uno de Facturación Electrónica y el otro de Repartos en los cuales se tomaron muestras de fragmentos de código, básicamente los segmentos donde se implementaron las funcionalidades de tipo CRUD.

Para medir la complejidad ciclomática se procedió a analizar el código mediante el uso de la herramienta de Software Tateti, el cual se encarga de analizar el

código de una clase línea por línea, posteriormente aplica la métrica de complejidad ciclomática descrita anteriormente y muestra el resultado de la medición de la complejidad ciclomática a nivel de toda la clase.

Para medir el acoplamiento entre objetos se procedió a analizar el código mediante el uso de la herramienta de software CK4J, en este caso el software lee el empaquetado (.jar o .war) de la aplicación y analiza las clases internas para determinar el nivel de acoplamiento entre clases según la métrica de Chidamber y Kemerer explicada anteriormente.

Para medir la densidad de comentarios en el código se procedió a analizar el código mediante el uso de la herramienta de Software Tateti, en este caso el software se encarga de analizar el código, línea por línea, posteriormente hace un recuento de la cantidad de líneas de código y la cantidad de líneas de código comentadas de los métodos, clases o package seleccionados.

Para medir el volumen del programa se procedió a analizar el código mediante el uso de la herramienta de Software Tateti, en este caso el software se encarga de analizar el código, línea por línea, hace un recuento de operadores y operandos de la clase a evaluar y utilizando la métrica de Halstead calcula el Volumen de un programa.

A continuación, la tabla muestra el resultado de una medición a todas las Clases que intervienen en el CRUD de Tipos de Afectación de un sistema de Facturación Electrónica y un Sistema de Repartos en sus respectivas capas (Entidad, Acceso a Datos, Negocio y Controller).

Tabla 3
Recolección de datos del Módulo de Gestión de Entregas

Clase / Métrica	CC	CBO	d	Vol.	LoC
Entidad	1.00	10.00	43.67	683.70	261.00
Acceso a Datos	26.00	22.00	23.24	3168.38	314.00
Negocio	18.00	12.00	33.59	846.49	259.00
Controller	14.00	22.00	12.14	2113.50	173.00
Promedio	14.75	16.50	28.16	1,703.02	251.75
Índice de Mantenibilidad					46.55

CC: Complejidad Ciclomática.
CBO: Acoplamiento entre Objetos.
d: Densidad de Comentarios en el Código.
Vol.: Volumen del Programa.
LoC: Línea de Código.
Fuente: Elaboración Propia

Tabla 4
Recolección de datos del Módulo de Gestión de Tipo de Afectación IGV.

Clase / Métrica	CC	CBO	d	Vol.	LoC
Entidad	1.00	5.00	15.71	97.67	70.0
Acceso a Datos	26.00	14.00	2.91	2489.11	206.00
Negocio	6.00	10.00	13.92	148.75	79.00
Controller	16.00	16.00	8.18	2026.26	159.00
Promedio	12.25	11.25	10.18	1,190.45	200.32
Índice de Mantenibilidad					49.8051

CC: Complejidad Ciclomática.
CBO: Acoplamiento entre Objetos.
d: Densidad de Comentarios en el Código.
Vol.: Volumen del Programa.
LoC: Línea de Código.
Fuente: Elaboración Propia

A continuación, el consolidado de datos recolectados del sistema después de aplicar el software generador de código con una nueva arquitectura.

Tabla 5

Recolección de Datos del Módulo de Gestión de Entregas

Clase / Métrica	CC	CBO	d	Vol.	LoC
Entidad	1.00	8.00	49.68	499.21	177.00
Acceso a Datos	1.00	6.00	12.00	11.09	59.00
Service	1.00	1.00	31.58	0.00	27.00
Repository	1.00	1.00	60.00	0.00	13.00
Controller	6.00	18.00	7.22	214.05	97.00
Promedio	2.00	6.80	32.10	144.87	74.60
Índice de Mantenibilidad					82.43

CC: Complejidad Ciclomática.
CBO: Acoplamiento entre Objetos.
d: Densidad de Comentarios en el Código.
Vol.: Volumen del Programa.
LoC: Línea de Código.
Fuente: Elaboración Propia

Tabla 6

Recolección de Datos del Módulo de Gestión de Tipo de Afectación IGV

Clase / Métrica	CC	CBO	d	Vol.	LoC
Entidad	1.00	8.00	48.72	202.12	87.00
Acceso a Datos	1.00	6.00	12.0	11.09	50.00
Service	1.00	1.00	31.58	0.000	27.00
Repository	1.00	1.00	60.00	0.00	10.00
Controller	6.00	14.00	7.53	221.65	93.00
Promedio	2.00	6.00	31.96	86.97	53.40
Índice de Mantenibilidad					90.49

CC: Complejidad Ciclomática.
CBO: Acoplamiento entre Objetos.
d: Densidad de Comentarios en el Código.
Vol.: Volumen del Programa.
LoC: Línea de Código.
Fuente: Elaboración Propia

Luego se hizo el cálculo de los indicadores para obtener los valores de cada dimensión de la siguiente manera:

Facilidad para realizar Pruebas Unitarias:

$$FPU = \mathrm{CC} + \mathrm{CBO}$$

Ecuación 1. Facilidad para hacer Pruebas Unitarias.
donde:

$$\mathrm{CC} + \mathrm{CBO} \leq 24 \qquad \forall \begin{cases} CC & \leq 10 \\ CBO & \leq 14 \end{cases}$$

Facilidad para hacer Cambios:

$$FC = Vol * d$$

Ecuación 2. Facilidad para hacer Cambios.

donde:

$$\mathrm{FC} \leq 7440 \quad \forall \begin{cases} Vol & \leq 744 \\ d & > 10 \end{cases}$$

Medición de la Variable Dependiente:

Cálculo del Índice de Mantenibilidad

Para calcular el índice de la mantenibilidad de código se emplea la siguiente fórmula:

$$\mathrm{IM} = 171 - 5.2 * \ln(\mathrm{avgVol}) - 0.23 * \mathrm{avgCC} - 16.2 * \ln(\mathrm{avgLoC}) + 50 * sin\,(\sqrt{2.4 * d})$$

Ecuación 3. Índice de Mantenibilidad.

Donde:

avgVol: Promedio del Volumen del programa por módulo.
avgCC: Promedio de la Complejidad Ciclomática por módulo.
avgLoC: Promedio de Líneas de Código por módulo.
d: Densidad de Comentarios en el Código.

Después de haber obtenido los valores de cada clase por módulo, se procedió a hacer el contraste de los resultados para averiguar si los índices de mantenibilidad han sido afectados positiva o negativamente.

CAPÍTULO III. RESULTADOS

Prueba de Hipótesis:

La contrastación de Hipótesis se ha realizado de acuerdo al método propuesto Pre Test y Post Test, para poder aceptar o rechazar la Hipótesis. Así mismo, para la realización de este diseño se identificaron indicadores cuantitativos los cuales se describen a continuación:

Tabla 7
Variables de Hipótesis

Variable	Indicador	Tipo de indicador
Dependiente	Complejidad Ciclomática	Cuantitativo
	Acoplamiento entre objetos	Cuantitativo

Fuente: Elaboración propia

Prueba de Hipótesis para el indicador 1: Cuantitativo.

Para hallar la complejidad ciclomática se realizó el análisis del código generado a través de la herramienta de Software **"Tateti"** que permite medir el recuento del número de caminos lógicos individuales contenidos en un programa, en la siguiente tabla se muestra el promedio del cálculo de la Complejidad Ciclomática una clase en particular para cada programa.

Tabla 8
Comparación de Resultados de Complejidad Ciclomática

	Pre-Test	Post-Test	Diferencia
Facturación Electrónica	12.25	2.00	10.25
Sistema de Repartos	14.75	2.00	12.75
Sumatoria	27.00	4.00	23.00
Promedio	13.5	2.00	11.50

Fuente: Elaboración propia

Prueba de Hipótesis para el indicador 2: Cuantitativo.

Cálculo para hallar el acoplamiento entre objetos del código sin aplicar el generador de código.

Para hallar el acoplamiento entre objetos se realizó el análisis del código generado a través de la herramienta de software **"CK4J"** para hacer un recuento del número de clases que están acopladas a otra clase en particular, los resultados se pueden apreciar en el anexo N de la matriz de instrumentos.

Tabla 9
Comparación de Resultados de Acoplamiento entre Objetos

	Pre-Test	Post-Test	Diferencia
Facturación Electrónica	11.25	6.00	17.25
Sistema de Repartos	28.16	6.80	23.3
Sumatoria	27.75	12.80	40.55
Promedio	13.88	6	7.88

Fuente: Elaboración propia

Prueba de Hipótesis para el indicador 3: Cuantitativo.

Se realizó el análisis del código generado a través de la herramienta de Software **"Tateti"** que permite medir la cantidad de comentarios en relación a la cantidad de

Tabla 10
Comparación de Resultados de la Densidad de Comentarios

	Pre-Test	Post-Test	Diferencia
Facturación Electrónica	10.18	31.96	21.78
Sistema de Repartos	28.16	32.10	3.94
Sumatoria	38.34	64.06	25.72
Promedio	19.17	32.03	12.86

Fuente: Elaboración propia

Prueba de Hipótesis para el indicador 4: Cuantitativo.

Se realizó el análisis del código generado a través de la herramienta de Software **"Tateti"** que permite medir el volumen de un programa, en la siguiente tabla se muestran los resultados obtenidos.

Tabla 11
Comparación de Resultados del Volumen del Programa

	Pre-Test	Post-Test	Diferencia
Facturación Electrónica	1,190.45	86.97	1,103.48
Sistema de Repartos	1,703.02	144.87	1,558.15
Sumatoria	2,893.47	231.84	2,661.63
Promedio	1,446.74	115.92	1,330.82

Fuente: Elaboración propia

Comparación global de resultados:

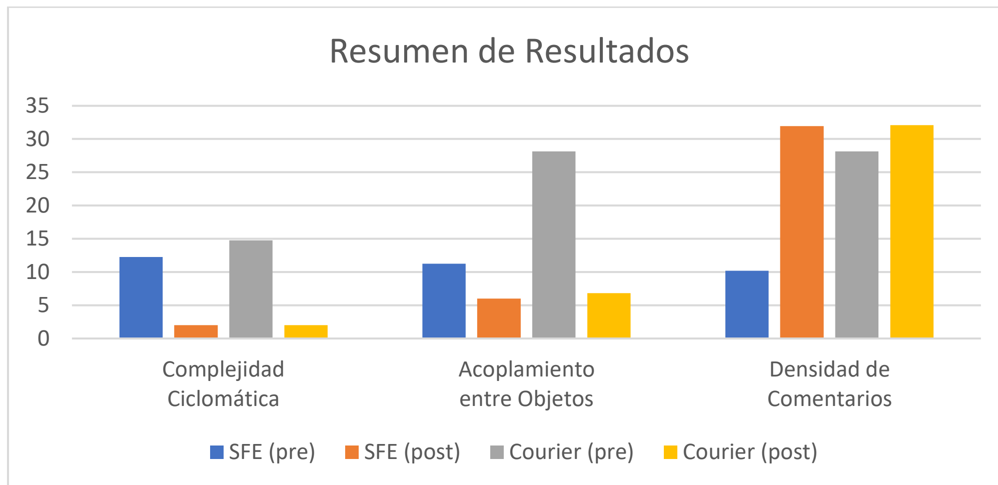


Figura 1. Comparación de Resultados de Indicadores (I Parte). Fuente: Elaboración propia

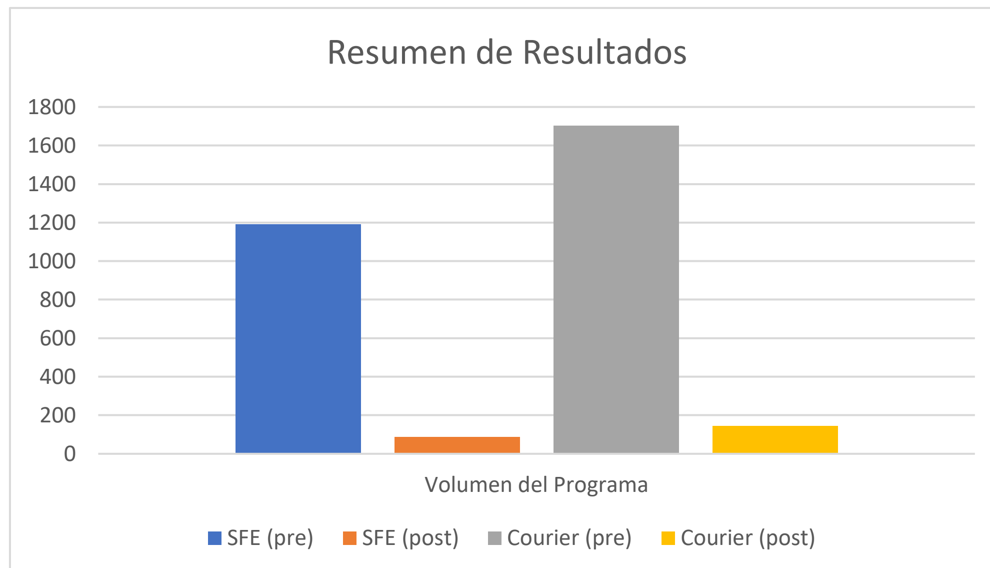


Figura 2. Comparación de Resultados de Indicadores (II Parte). Fuente: Elaboración propia

Tabla 12

Resultados de Facilidad para hacer Pruebas Unitarias.

	Fórmula	Total
Facturación Electrónica (Pre)	$FPU = \text{CC} + \text{CBO}$	23.5
Sistema de Repartos (Pre)		42.91
Facturación Electrónica (Post)		8.00
Sistema de Repartos (Post)		8.80

Fuente: Elaboración propia

Comparación global de resultados de Facilidad para hacer Pruebas Unitarias:

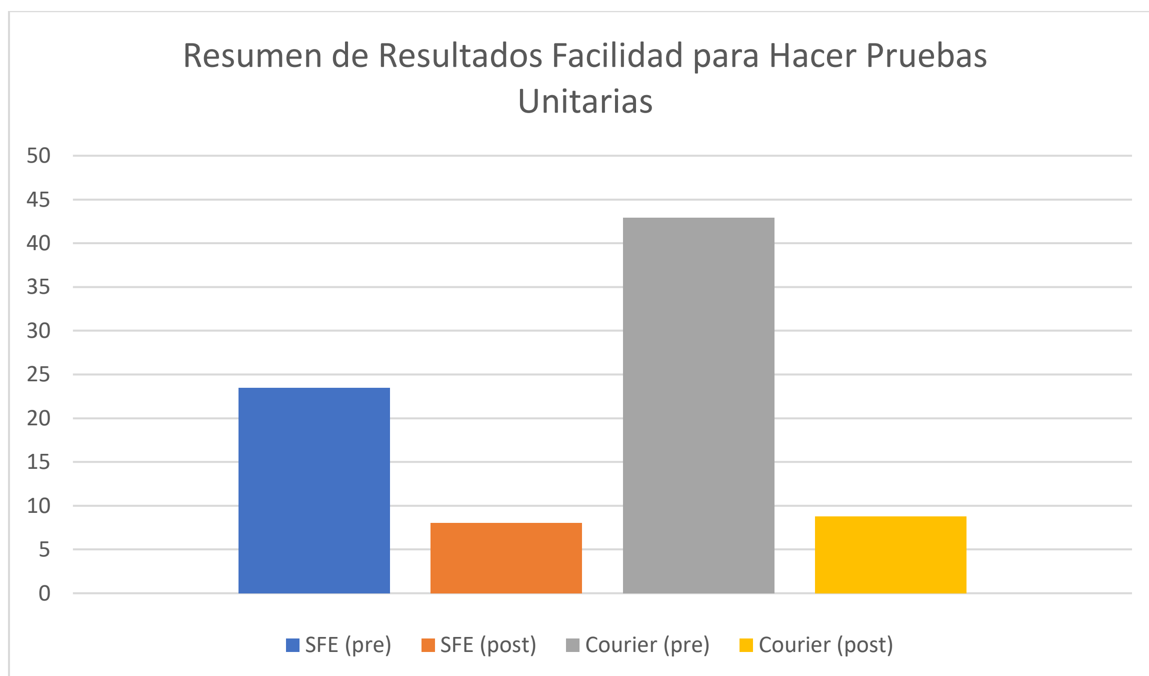


Figura 3. Comparación de Resultados de Índice de Mantenibilidad. Fuente: Elaboración propia

Tabla 13

Resultados de Facilidad para hacer Cambios.

	Fórmula	Total
Facturación Electrónica (Pre)	$FC = Vol * d$	12,118.78
Sistema de Repartos (Pre)		47,957.04
Facturación Electrónica (Post)		2,779.56
Sistema de Repartos (Post)		4,650.33

Fuente: Elaboración propia

Comparación global de resultados de Facilidad para hacer Cambios:

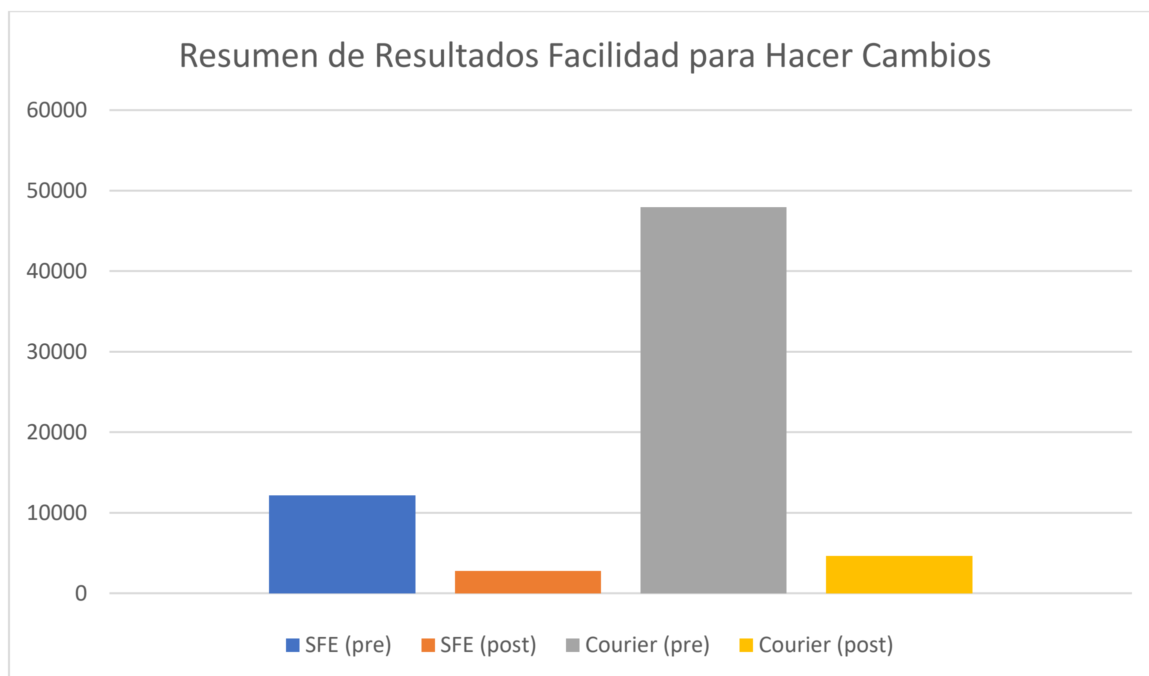


Figura 4. Comparación de Resultados de Índice de Mantenibilidad. Fuente: Elaboración propia

Tabla 14
Resultados de Índice de Mantenibilidad

	Fórmula	Total
Facturación Electrónica (Pre)	$IM = 171 - 5.2 * \ln(avgVol) - 0.23 * avgCC - 16.2 * \ln(avgLoC) + 50 * sin(\sqrt{2.4 * d})$	49.81
Sistema de Repartos (Pre)		46.55
Facturación Electrónica (Post)		90.49
Sistema de Repartos (Post)		82.43

Fuente: Elaboración propia

Comparación global de resultados de Índice de Mantenibilidad:

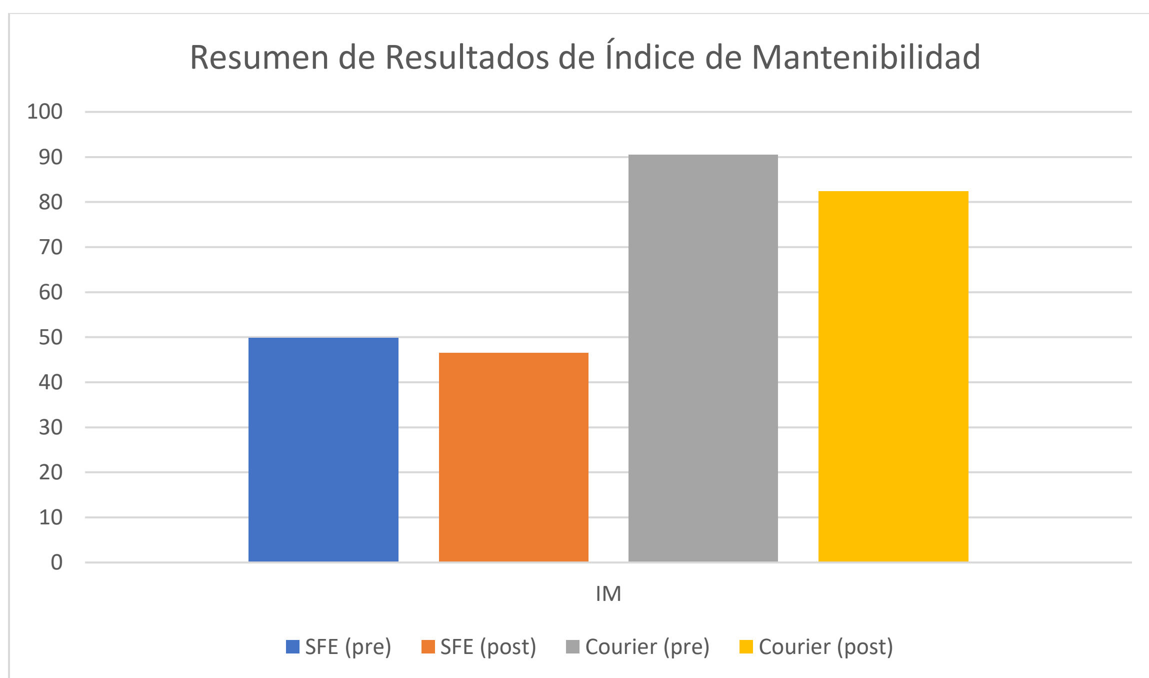


Figura 5. Comparación de Resultados de Índice de Mantenibilidad. Fuente: Elaboración propia

CAPÍTULO IV. DISCUSIÓN Y CONCLUSIONES

4.1 Discusión

Con los resultados obtenidos en el capítulo anterior, se valida la Hipótesis, que la aplicación de un Software generador de código de funcionalidades tipo CRUD influye positivamente en la mantenibilidad facilitando las pruebas unitarias y modificaciones de los Sistemas de Información Empresariales.

En la tabla 13 encontramos que los valores de la VD de Mantenibilidad son de 49.81 y 46.55 para los Sistemas de Facturación Electrónica y Courier, mientras que, para los sistemas creados con el generador de código, los valores de Índice de Mantenibilidad son de 90.49 y 82.43 respectivamente, por lo tanto, se encuentra en un rango aceptable tal como lo menciona Don Coleman (1990) y coincide con el trabajo realizado por Millalén A. (2014) en su tesis titulada "Mejoramiento de la Mantenibilidad de Software en una Empresa de Servicios Electrónicos", donde a través de la refactorización de código logra aumentar el índice de Mantenibilidad de código.

En la tabla 9 encontramos en el pretest que para los sistemas de facturación electrónica y de Repartos la complejidad ciclomática en promedio es de 12.25 y 14.75 mientras que, para los nuevos sistemas creados con el generador, la complejidad ciclomática es de 2 y 2 respectivamente, reduciendo la complejidad en 10 unidades y ocupando el rango del 1 al 10, por lo tanto, tal como lo menciona (Thomas McCabe, 1976) es una evidencia de complejidad simple y sin mucho riesgo. (ver anexo 4).

En la tabla 10 encontramos en el pretest que para los sistemas de facturación electrónica y Repartos el acoplamiento entre clases es de es de 11.25 y 28.16, mientras que, para los nuevos sistemas creados con el generador, el acoplamiento entre clases

promedio es de 6 y 6.8 respectivamente, por lo tanto, en un rango aceptable tal como lo menciona Chidamber y Kemerer (1994). (ver anexo 4)

En la tabla 11 encontramos en el pretest que el promedio de densidad de comentarios en el código de los sistemas de facturación electrónica y en el sistema Repartos es de 10.18% y 28.16% respectivamente, mientras que, para los nuevos sistemas creados con el generador, la densidad de comentarios es de 31.96% y 32.10%, esto evidencia que el generador de código permite aumentar la densidad de comentarios, aunque no existe un aumento significativo, le permite cumplir con el estándar permitido. (ver anexo 4)

En la tabla 12 encontramos en el pretest que el volumen del programa en el código de los sistemas de facturación electrónica y en el sistema de Repartos es de 1190.45 y 1703.02 mientras que, para los nuevos sistemas creados con el generador, el volumen del programa es de 86.97 y 144.87. Por lo tanto, se ubican dentro de los rangos aceptables dentro de los intervalos permitidos propuestos por Chidamber y Kemerer. (ver anexo 4)

Ambos resultados de las mediciones del software se interpretan de una manera positiva gracias a los datos arrojados por la medición del Índice de Mantenibilidad que resume todos los indicadores en una sola ecuación y en la que podemos demostrar matemáticamente que la mantenibilidad aumenta al utilizar generadores de código que cumplen con patrones de diseño y principios de codificación.

Por estas razones, es que la presente investigación pretende dar a conocer cuáles son las ventajas de utilizar generadores de código y cuáles serían los beneficios en la mantenibilidad del código y que es importante que se analice si verdaderamente

conviene iniciar un proyecto desde cero o utilizar herramientas de generación de código estandarizadas.

4.2 Conclusiones

Al finalizar la presente Tesis, se llegaron a las siguientes conclusiones:

El software generador de código tipo CRUD tiene una influencia positiva sobre la mantenibilidad de software.

Se demuestra la influencia positiva del generador de código en la dimensión de Facilidad para hacer pruebas unitarias en la mantenibilidad de software según los resultados de la medida de la complejidad ciclomática y el acoplamiento entre objetos.

Se demuestra la influencia positiva del generador de código en la dimensión de Facilidad para hacer modificaciones o cambios en la mantenibilidad de software según los resultados de la medida de la densidad de comentarios en el código y el volumen del programa.

Se demuestra que la utilización de una herramienta de generación de código que respete los principios de código mejora la mantenibilidad del programa.

4.3 Recomendaciones

Se recomienda abarcar temas de mantenibilidad de la ISO 25010 que no se tomaron en cuenta en esta investigación como son la Modularidad, Analizabilidad y Reusabilidad.

Se recomienda una capacitación incisiva a los programadores sobre temas de calidad de código, pues un generador de código realiza la creación de la estructura de los sistemas, pero depende del desarrollador seguir y continuar con las buenas prácticas propuestas.

Finalmente, se recomienda el uso de software de medición de calidad de código que sigan métricas propuestas por expertos.

REFERENCIAS

Ancán, O., Cares C. & Cravero, A. (octubre 2018). Código con mal olor: un mapeo sistemático. *Revista Cubana de Ciencias Informáticas, 12(4).* Obtenido de http://scielo.sld.cu/pdf/rcci/v12n4/rcci13418.pdf

Callejas, M., Alarcón, A. C., & Álvarez, A. M. (junio de 2017). Modelos de calidad del software, un estado del arte. *Facultad de Ingeniería, 13(1).* Obtenido de http://www.scielo.org.co/pdf/entra/v13n1/1900-3803-entra-13-01-00236.pdf

Carretero, J. (mayo de 2013). Entorno de Pruebas de Generadores de Código Automático. *Facultad de Ingeniería.* Obtenido de http://ir.ii.uam.es/~fdiez/TFGs/gestion/leidos/2013/20130610JoseCarreteroArias.pdf

Cecilio, A. (2 de marzo de 2016). ¿Qué es Spring Boot?. En Blog: Arquitectura Java. Recuperada de https://www.arquitecturajava.com/que-es-spring-boot/

Chidamber, Kemerer (1994). A metric suite for object oriented design. En Revista IEEE transactions on software engineering, 20 (6) pp.467-493.

Damien, A. (s.f.). Arquitectura contemporánea. GitHub. Recuperado de https://github.com/darrachequesne/spring-data-jpa-datatables

Daniel, R., Raúl N., José Rubén, L. & Alicia D. (2017). Curso de Ingeniería de Software: 2ª Edición. Plaza América, Vigo, España: IT Campus Academy. Recuperado de https://books.google.com.pe/books?id=G2Q4DgAAQBAJ&pg=PA208&dq=metodolog%C3%ADa+XP&hl=es419&sa=X&ved=0ahUKEwjq1PG1o_jeAhXlpVkKHW9KCfUQ6AEIPzAE#v=onepage&q=metodolog%C3%ADa%20XP

Diego, C. (2018). Metodología XP Programación Extrema (Metodología ágil). Recuperado de http://www.diegocalvo.es/metodologia-xp-programacion-extrema-metodologia-agil/

Elena, R. (2017). Nuevas tendencias en los sistemas de información. Madrid, España: Centro de Estudios Ramon Areces SA. Recuperado de https://books.google.com.pe/books?id=6ZVADwAAQBAJ&pg=PA280&dq=metodolog%C3%ADa+XP&hl=es419&sa=X&ved=0ahUKEwjdh4eAhPreAhVL7FMKHX51DTEQ6AEIRzAF#v=onepage&q=metodolog%C3%ADa%20XP&f=false

Emil, F. (s.f.). CK4J. GitHub. Recuperado de https://github.com/Pyknic/CK4J

EncuRed (s.f). Programación Extrema (XP). Recuperado de https://www.ecured.cu/Programaci%C3%B3n_Extrema_(XP)

Erazo, J., Flores, A. & Pino, F. J. (junio de 2016). Análisis y clasificación de atributos de mantenibilidad del software: una revisión comparativa desde el estado del arte. *Entre Ciencia e Ingeniería, 10(19).* Obtenido de http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1909-83672016000100006

Erazo, J., Flores, A. & Pino, F. J. (julio de 2016). Generando productos software mantenibles desde el proceso de desarrollo: El modelo de referencia MANTuS. *Ingeniare. Rev. chil. ing., 24(3).* Obtenido de https://scielo.conicyt.cl/scielo.php?script=sci_arttext&pid=S0718-33052016000300007

Fernando, A. (2005). Introducción a la ingeniería del software. Madrid, España: Delta publicaciones. Recuperado de https://books.google.com.pe/books?id=FHTnCgAAQBAJ&pg=PT81&dq=mantenibilidad+de+software&hl=es419&sa=X&ved=0ahUKEwj87cfQ0vreAhVqpVkKHTYgC_gQ6AEIMzAC#v=onepage&q&f=false

Fernando, A. (2013). Java a fondo: - estudio del lenguaje y desarrollo de aplicaciones. Buenos Aires, Argentina: Alfaomega Grupo Editor. Recuperado de https://books.google.com.pe/books?id=WcL2DQAAQBAJ&pg=PT114&dq=que+es+java

&hl=es419&sa=X&ved=0ahUKEwiKx7Pek_jeAhVFp1kKHZWTCr4Q6AEIMzAC#v=onepage&q&f=false

Gabriel, M. (2016). Aprende a Desarrollar con Spring Framework: 2ª Edición. Madrid, España: IT Campus Academy. Recuperado de https://books.google.com.pe/books?id=4BbfDQAAQBAJ&printsec=frontcover&dq=SPRING+FRAMEWORK&hl=es419&sa=X&ved=0ahUKEwjrtsKQw_zeAhVDlVkKHYz8CjQQ6AEINTAB#v=onepage&q&f=false

Gaitán, C. A. (junio 2017)._Líneas de Productos Software: Generando Código a Partir de Modelos y Patrones. *Scientia et Technica, 22(2).* Obtenido de http://www.redalyc.org/pdf/849/84953103009.pdf

Greiner, C., Demchum, D., Dapozo, G. & Estayno, M. (junio 2010). Una propuesta de solución para automatizar la medición de aplicaciones orientadas a objeto. *Facultad de Ciencias Exactas y Naturales y Agrimensura.* Obtenido de http://sedici.unlp.edu.ar/bitstream/handle/10915/19301/066.pdf?sequence=1

Hamid, A., Ilyas, M., Hummayun, M. & Nawaz, A. (marzo de 2013). A Comparative Study on Code Smell Detection Tools. *International Journal of Advanced Science and Technology, (60).* Obtenido de https://pdfs.semanticscholar.org/4229/3bfa78f90c6c377afd1a7cd83d9b62245da7.pdf?fbclid=IwAR33_97ie28ZP1PmdILYormESk1GQE-1vLrJLBzVSiOWE4nq9h6P3xHXrxg

Ian, S. (2005). Ingeniería del software. Madrid, España: Pearson Educación. Recuperado de https://books.google.com.pe/books?id=gQWd49zSut4C&printsec=frontcover&hl=es&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

Irrazábal, E., Greiner, C. & Dapozo, G. (julio de 2015). La refactorización de software basada en valor: Revisión sistemática de la literatura. *Simposio Argentino de Ingeniería de Software.* Obtenido de http://44jaiio.sadio.org.ar/sites/default/files/asse130-144.pdf

Jan, S. (2016). Índice de Mantenibilidad de Software y JavaScript. Recuperado de https://frontendlabs.io/3121--indice-de-mantenibilidad-software-javascript

Javier, E. (2013). El nuevo PHP. Conceptos Avanzados. Madrid, España: 3Ciencias. Recuperado de https://books.google.com.pe/books?id=NSj3AQAAQBAJ&pg=PA109&dq=Arquitectura+Modelo+Vista+Controlador+(MVC).&hl=es419&sa=X&ved=0ahUKEwiB6tygoP3eAhUFq1kKHcfrDmkQ6AEILDAB#v=onepage&q=Arquitectura%20Modelo%20Vista%20Controlador%20(MVC).&f=false

Javier, M. (2014). Pruebas de Rendimiento TIC. Laderas del Campillo, España: Lulu.com. Recuperado de https://books.google.com.pe/books?id=kTvbBgAAQBAJ&printsec=frontcover#v=onepage&q&f=false

Javier T, Isabel R & José, D. (2007). Técnicas cuantitativas para la gestión en la ingeniería del software. La Coruña, España: Netbiblo. Recuperado de https://books.google.com.pe/books?id=PZQoZ9KTNaEC&pg=PA336&dq=M%C3%A9trica+de+Acoplamiento+entre+Objetos+(CUBO)+de+Chidamber+y+Kemerer&hl=es419&sa=X&ved=0ahUKEwiSoXP5freAhWStlMKHdLrACYQ6AEIKDAA#v=onepage&q&f=false

Jimmy, M., Mariuxi, Z., Fausto R., Nancy L., Milton V. & Joofre H. (2018). SNAIL, Una metodología híbrida para el desarrollo de aplicaciones web. Madrid, España: Pearson Educación. Recuperado de

https://books.google.com.pe/books?id=_KlcDwAAQBAJ&pg=PA96&dq=Plan+de+entregas+(%E2%80%9CRelease+Plan%E2%80%9D):&hl=es419&sa=X&ved=0ahUKEwiGyWcyPreAhVHvVMKHa-4CxwQ6AEIKDAA#v=onepage&q&f=false

José O. & Pilar A. (2012). Programación web en Java. Madrid, España: Ministerio de Educación. Recuperado de https://books.google.com.pe/books?id=ry8bAgAAQBAJ&printsec=frontcover#v=onepage&q&f=false

José Rubén, L. (2015). Desarrollo de Software ÁGIL: Extreme Programming y Scrum. Plaza América, Vigo, España: IT Campus Academy. Recuperado de https://books.google.com.pe/books?id=M4fJCgAAQBAJ&printsec=frontcover&dq=metodolog%C3%ADa+XP&hl=es419&sa=X&ved=0ahUKEwjdh4eAhPreAhVL7FMKHX51DTEQ6AEIOTAD#v=onepage&q&f=false

Karla, C. (2015). Metodología de Desarrollo Ágil: XP y Scrum. Recuperado de https://ingsotfwarekarlacevallos.wordpress.com/2015/05/08/metodologia-de-desarrollo-agil-xp-y-scrum/

Kenneth, E. (2005). Analisis y Diseno de Sistemas - 6b: Edicion (Spanish Edition). Madrid, España: Pearson Educación. Recuperado de https://books.google.com.pe/books?id=5rZA0FggusC&pg=PT199&dq=QUE+SON+LAS+HISTORIAS+DE+USUARIO+XP&hl=es419&sa=X&ved=0ahUKEwiixqGamPreAhUr1lkKHW9cAsgQ6AEIKDAA#v=onepage&q=QUE%20SON%20LAS%20HISTORIAS%20DE%20USUARIO%20XP&f=false

López, A. M., Cabrera, C. & Valencia, L. E. (septiembre de 2008). Introducción a la calidad de software. *Scientia et Technica.* Obtenido de https://dialnet.unirioja.es/descarga/articulo/4745899.pdf

Malavolta, A. (julio 2016). Análisis de detección de Code Smells para el lenguaje JavaScript. *Facultad de Ciencias Exactas.* Obtenido de https://www.ridaa.unicen.edu.ar/xmlui/bitstream/handle/123456789/1724/Trabajo%20Final%20Malavolta.pdf?sequence=1&isAllowed=y

Mariuxi Z., Jimmy R. & Fausto R. (2017). Administración de Bases de Datos con Postgresql. Alicante, España: 3Ciencias. Recuperado de https://books.google.com.pe/books?id=5mkDgAAQBAJ&pg=PA12&dq=PostgreSQL&hl=es419&sa=X&ved=0ahUKEwjdxb_rnv3eAhVxuVkKHdymABUQ6AEIYDAH#v=onepage&q&f=false

Méndez, M., Garrido, A., Overbey, J., Tinetti, F. G. & Johnson, R. (enero de 2010). Refactorización en Código Fortran Heredado. *Congreso Argentino de ciencias de la computación*. Obtenido de https://www.researchgate.net/publication/261724947_Refactorizacion_en_codigo_Fortran_heredado

Nacho, B. (2009). Manual de desarrollo web con Grails. Madrid, España: Imaginaworks Software Facto. Recuperado de https://books.google.com.pe/books?id=N2AvBwAAQBAJ&printsec=frontcover#v=onepage&q&f=false

Nina, S. (s.f.). Tateti Software. GitHub. Recuperado de https://github.com/nsatragno/tateti

Nuria, R., William, M. (2006). Planificación Y Evaluación de Proyectos Informáticos. San José, Costa Rica: EUNED. Recuperado de https://books.google.com.pe/books?id=UK5Ys_kBlwYC&pg=PA126&dq=que+es+una+metrica+de+software&hl=es419&sa=X&ved=0ahUKEwiq4b_52_reAhWM71MKHaiQBSMQ6AEIOjAD#v=onepage&q&f=false

Olmedilla, J. J. (septiembre de 2005). Revisión Sistemática de Métricas de Diseño Orientado a Objetos. *Facultad de Informática.* Obtenido de http://www.dlsiis.fi.upm.es/docto_lsiis/Trabajos20042005/Olmedilla.pdf

Open Stacks (s.f.). Métricas del Mantenimiento de Software. Recuperado de https://cnx.org/contents/oEhMfFuG@9.1:Yi7zVzOP@2/Definici%C3%B3n-de Mantenibilidad

Pablo, G. (2015). Comenzando a programar con JAVA. Elche, España: Universidad Miguel Hernández. Recuperado de https://books.google.com.pe/books?id=4v8QCgAAQBAJ&pg=PA107&dq=que+es+java&hl=es419&sa=X&ved=0ahUKEwiKx7Pek_jeAhVFp1kKHZWTCr4Q6AEIOTAD#v=onepage&q&f=false

Pérez, H. G., Martínez, F. E., Nava, S. E., Núñez, A. S., Vásquez, M. & Flores J. A. (2015). Analizando la Mantenibilidad de Software Desarrollado Durante la Formación Universitaria. *Revista Latinoamericana de Ingeniería de Software, 3(6).* Obtenido de https://www.researchgate.net/publication/295839982_Analizando_la_Mantenibilidad_de_Software_Desarrollado_Durante_la_Formacion_Universitaria

Rafael, G. (2015). Despliegue y puesta en funcionamiento de componentes software. Antequera, España: Upc Edicions Upc. Recuperado de https://books.google.com.pe/books?id=FHTnCgAAQBAJ&pg=PT81&dq=mantenibilidad+de+software&hl=es419&sa=X&ved=0ahUKEwj87cfQ0vreAhVqpVkKHTYgC_gQ6AEIMzAC#v=onepage&q&f=false

Sagrado, J., Águila, I. M., Bosch, A. & Chicano, F. (agosto de 2017). Impacto de las métricas CK en la refactorización. *Departamento de Lenguajes y CC. de la Computación.* Obtenido de

https://www.researchgate.net/publication/318814790_Impacto_de_las_metricas_CK_en_la_refactorizacion

Sayago, J. P. (julio de 2018). Generador de Código Utilizando el Paradigma de Líneas de Producto Software. *Revista Científica Hallazgos21, 3(2).* Obtenido de https://www.researchgate.net/publication/327021042_Generador_de_Codigo_Utilizando_el_Paradigma_de_Lineas_de_Producto_Software

Sergio, R. (2015). JSF 2 + Hibernate 4 + Spring 4: PrimeFaces 5 with JAX-WS y EJB'S. Madrid, España: Sergio Ríos. Recuperado de https://books.google.com.pe/books?id=N2AvBwAAQBAJ&printsec=frontcover#v=onepage&q&f=false

Tech Target (2008). CRUD: cycle (Create, Read, Update and Delete Cycle). Recuperado de https://searchdatamanagement.techtarget.com/definition/CRUD-cycle

Truica, C. O., Radulescu, F., Boicea A., & Bucur, I. (agosto de 2015). Performance evaluation for CRUD operations in asynchronously replicated document oriented database. *Computer society.* Obtenido de https://www.academia.edu/14817696/Performance_evaluation_for_CRUD_operations_in_asynchronously_replicated_document_oriented_database

Vásquez, P. J., Moreno, M. N., & García, F. J. (noviembre de 2001) Métricas orientadas a objetos. *Facultad de Informática y Automática.* Obtenido de https://gredos.usal.es/bitstream/handle/10366/21758/DPTOIA-IT-2001-002.pdf?sequence=1&isAllowed=y

Vicenc, F. (2010). Desarrollo de sistemas de información: una metodología basada en el modelado. Catalunya, España: Upc Edicions Upc. Recuperado de https://books.google.com.pe/books?id=Sqm7jNZS_L0C&pg=PA102&dq=ventajas+y+des

ventajas+de+la+metodolog%C3%ADa+xp&hl=es419&sa=X&ved=0ahUKEwi88ajwzPre

AhWmt1kKHa_7AhsQ6AEILjAB#v=onepage&q&f=false

Anexo nro. 1: Matriz de Consistencia

GENERADOR DE CÓDIGO DE FUNCIONALIDADES TIPO CRUD PARA FAVORECER LA MANTENIBILIDAD DE SOFTWARE APLICADO A SISTEMAS DE INFORMACIÓN EMPRESARIALES.				
PROBLEMA	**HIPÓTESIS**	**OBJETIVO GENERAL**	**VARIABLE INDEPENDIENTE**	**METODOLOGÍA**
¿De qué manera la aplicación de un software generador de funcionalidades tipo CRUD influye en la mantenibilidad de los sistemas de información empresariales?	La aplicación de un Software generador de código de funcionalidades tipo CRUD influirá positivamente en la mantenibilidad facilitando las pruebas unitarias y los cambios en Sistemas de Información Empresariales.	Definir la influencia en la mantenibilidad de código en sistemas de información empresariales, mediante la aplicación de un software generador de código de funcionalidades tipo CRUD.	Software Generador de Código de Funcionalidades tipo CRUD	**Diseño**

				Pre-Experimental *G O1 X O2* Donde: G = Muestra X= *Software Generador de Código de funcionalidades tipo CRUD* O1: Medición pre-experimental de *Mantenibilidad de Software en Sistemas de información Empresariales.* O2: Medición post-experimental de *Mantenibilidad de Software en Sistemas de información Empresariales*
				Población

				CRUD's para software de Facturación Electrónica. *CRUD's para software de Empresa Courier.*
		OBJETIVOS ESPECIFICOS	**VARIABLE DEPENDIENTE**	**Muestra**
		Determinar el nivel de facilidad para aplicar pruebas unitarias al código generado. Determinar el nivel de facilidad de modificación o cambios al código generado.	Mantenibilidad de Software	*N1=2* Total: 2

Anexo nro. 2: Matriz de Operacionalización

VARIABLE DEPENDIENTE	DEFINICIÓN CONCEPTUAL	DEFINICIÓN OPERACIONAL	DIMENSIONES	INDICADORES	ESCALA de MEDICION
Mantenibilidad de Software	Es la "capacidad del software para ser modificado y probado con la finalidad de realizar correcciones, mejoras, adaptaciones, etc." (Granados, 2015, p. 50).	Capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas. Se subdivide en: -Capacidad para ser modificado. -Capacidad para ser probado. (ISO 25010)	Facilidad para hacer Pruebas Unitarias.	Métrica de Complejidad Ciclomática de McCabe.	ORDINAL
				Métrica de Acoplamiento entre Objetos (CBO) de Chidamber y Kemerer.	
			Facilidad de Modificación o Cambios.	Densidad de Comentarios en el código.	

				Métrica de Volumen de un programa de Halstead	
VARIABLE INDEPENDIENTE	**DEFINICIÓN CONCEPTUAL**	**DEFINICIÓN OPERACIONAL**	**DIMENSIONES**	**INDICADORES**	**ESCALA DE MEDICIÓN**
Software Generador de funcionalidades tipo CRUD.	Genera aplicaciones para gestionar las operaciones básicas CRUD en minutos, desde el lado del servidor y el cliente, obteniendo los metadatos de las tablas definidas en la Base de Datos, basándose en un conjunto de reglas.	Un software generador de código de funcionalidades tipo CRUD es medido por el nivel de reconocimiento estructural de la Base de Datos fuente (metadatos) de una base de datos relacional y por la capacidad de generar código tanto del lado del servidor (back-end) como del lado del cliente (front-end).	Reconocimiento de metadatos de la Base de Datos.	Reconoce esquemas.	NOMINAL
				Reconoce tablas.	
				Reconoce columnas por tabla.	
				Reconoce tipos de dato.	
			Generación de BackEnd	Genera la arquitectura de clases.	
				Genera el proyecto ejecutable.	
			Generación de front-End	Genera una composición de estilos CSS y JS.	
				Genera la arquitectura vistas CRUD.	

				Genera el proyecto ejecutable.	

Anexo nro. 3: Matriz de Instrumentos

SISTEMA DE FACTURACIÓN ELECTRÓNICA						
VARIABLE	**DIMENSIÓN**	**INDICADOR**	**Fórmula**	**OPCION DE RESPUESTA**		
				leyenda	valor	Total
MANTENIBILIDAD DE SOFTWARE	Facilidad para hacer Pruebas Unitarias.	1. Complejidad Ciclomática.	$v(G) = e - n + 2$	**e**: número de aristas.		
				n: número de nodos.		
		2. Acoplamiento entre Objetos.	$CBO = F\left(\sum_{j=0}^{N} f(x)\right)$	**N**: número de clases acopladas.		
	Facilidad de Modificación o Cambios.	3. Densidad de Comentarios en el código.	$d = \frac{\mathrm{LC} * 100}{\mathrm{LoC}}$	**LC**: Líneas de Código.		
				LoC: Líneas Comentadas.		
		4. Volumen del programa.	$\mathrm{V} = \mathrm{N} * \log2(\mathrm{n})$ donde: N= N1+N2 n=n1+n2	**N1**: número total de operadores.		
				N2: número total de operandos.		
				n1: número de operadores únicos.		
				n2: número de operandos.		

Anexo nro. 4: Análisis de Datos

Dimensión: Facilidad para hacer pruebas Unitarias – Métrica de Complejidad Ciclomática.

Propósito	Determinar la complejidad ciclomática de código
Fórmula	$$v(G) = e - n + 2$$ **e**: número de aristas. **n**: número de nodos.
Tipo de Medida	e = número entero mayor que cero. **n** = número entero mayor que cero.
Interpretación	1 al 10: Programa Simple, sin mucho riesgo 11 al 20: Más complejo, riesgo moderado 21 al 50: Complejo, Programa de alto riesgo. 50 a más: Programa no testeable, Muy alto riesgo.

Dimensión: Facilidad para hacer pruebas Unitarias – Métrica de Acoplamiento entre Objetos.

Propósito	Determinar el nivel de acoplamiento entre objetos (clases).
Fórmula	$$CBO = F\left(\sum_{j=0}^{N} f(x)\right)$$ **N**: número de clases acopladas.
Tipo de Medida	j = número entero mayor o igual que cero. N= número entero mayor o igual a cero.
Interpretación	CBO <=14: Acoplamiento aceptable. CBO >14: Alto Acoplamiento.

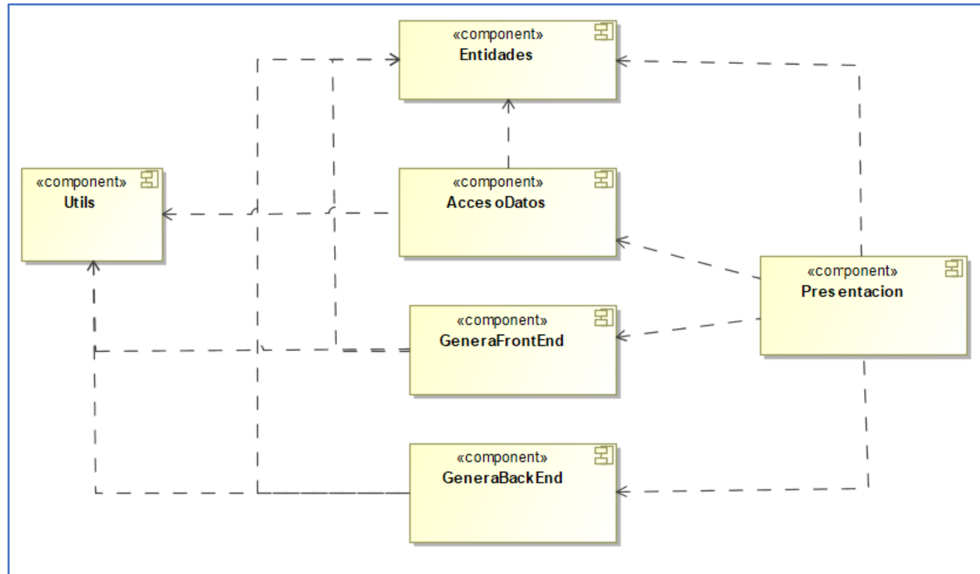
Dimensión: Facilidad de modificación o cambios – Métrica de Densidad de Comentarios en el código.

Propósito	Determinar el porcentaje de comentarios de código en proporción a las líneas de código de programación.
Fórmula	$$d = \frac{\mathrm{LC} * 100}{\mathrm{LoC}}$$ **LoC**: Líneas de Código. **LC**: Líneas Comentadas.
Tipo de Medida	LoC = número entero mayor que cero. LC = número entero mayor o igual que cero.
Interpretación	X<= 10: Densidad de Comentarios Baja. X> 10: Densidad de Comentarios moderada – Alta.

Dimensión: Facilidad de modificación o cambios – Métrica de Volumen de un programa.

Propósito	Determinar la dificultad de comprender un programa.
Fórmula	$$V = N * \log2(n)$$ N = N1+N2 n = n1+n2 **N1**: número total de operadores. **N2**: número total de operandos. **n1**: número de operadores únicos. **n2**: número de operandos.
Tipo de Medida	**N1**: número total de operadores. **N2**: número total de operandos. **n1**: número de operadores únicos. **n2**: número de operandos.
Interpretación	V <= 744: Mantenibilidad Aceptable. V > 744: Mantenibilidad compleja.

Anexo nro. 5: Diagrama de Componentes del Generador de Código



Anexo nro. 6: Vista del Generador de Código

Generador de Código v.2.1.1

CONFIGURACIÓN DE LA CONEXIÓN

TIPO BASE DATOS: POSTGRESQL
HOST: localhost
PUERTO: 5433
INTEGRATED SECURITY
USER ID: postgres
CLAVE: ************
Cargar Base de Datos

Condiguración de la Aplicación

BASE DATOS: sfeDemo
LENGUAJE PROG.: JAVA
EMPAQUETADO

Parámetros de Aplicación

Nombre: demoGenerador
Grupo: com.burbit.demo1
Package: com.burbit.demo1.backend
Autor: Julio Becerra Urbina
Artifact: Demo1
Versión: 0.0.1-SNAPSHOT
Descripción: ProyectoEjemplo
Generar Proyecto BackEnd

Generar código por Tabla

ESQUEMA: contabilidad
TABLA: comprobante
TIPO DE CLASE: Entity
Generar

```
package com.burbit.demo1.backend.contabilidad.entity;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Column;
import org.hibernate.annotations.OrderBy;
import java.math.BigDecimal;
import javax.persistence.FetchType;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.burbit.demo1.backend.fe.entity.TipoDocumento;
import com.burbit.demo1.backend.fe.entity.Moneda;
```

Generar Proyecto FrontEnd

Nombre: demoGeneradorFront

X	SCHEMA	TABLE NAME
☐	contabilidad	bitacora
☐	contabilidad	comprobante
☐	contabilidad	comprobante_item
☐	contabilidad	detraccion
☐	contabilidad	guia_remision_transportista
☐	contabilidad	nota_credito_item_nombre_servicio
☐	contabilidad	resumen_diario
☐	contabilidad	serial
☐	fe	moneda
☐	fe	tipo_afectacion_igv
☐	fe	tipo_documento
☐	fe	tipo_nota_credito
☐	fe	tipo_nota_debito
☐	fe	tipo_precio_venta_unitario
☐	fe	tipo_tributo
☐	fe	tipo_unidad_comercial

Generar Proyecto FrontEnd

Anexo nro. 7: Documento de Planificación del Proyecto Generador de Código

Introducción

En el presente documento se identificarán las necesidades de parte del Stakeholder, por consiguiente, se elaborará un plan de iteraciones con la finalidad de establecer tiempos de desarrollo y entrega.

Se propone con **SmartCrud,** ofrecer una herramienta que genere código fuente para el uso de los desarrolladores de software.

Historias de Usuario

Historia 1: Épica
Como desarrollador de Software quiero generar código CRUD para mis aplicaciones en java por lo menos en las capas principales como Entidades, Acceso a Datos y Negocio.

Historia 2
Como desarrollador de software quiero generar una vista web para mis CRUD's y que estas tengan validación en tiempo real y que estéticamente se vean muy bien.

Historia 3
Como desarrollador de Software quiero generar una capa de servicios REST para conectar el lenguaje de programación con las vistas Web.

Historia 4
Quiero que el código generado tenga calidad en su enfoque de mantenibilidad para poder realizar cambios sin dificultad.

Objetivos del Proyecto

1.1.1. **1.3.1. Objetivo general**

✓ Implementar un sistema informático generador de código que permita resolver la problemática de la carga de tiempo al momento de crear los CRUD's y estos sean fáciles de modificar.

1.1.2. **1.3.2. Objetivos específicos**

✓ Identificar la estructura completa de códigos CRUD.

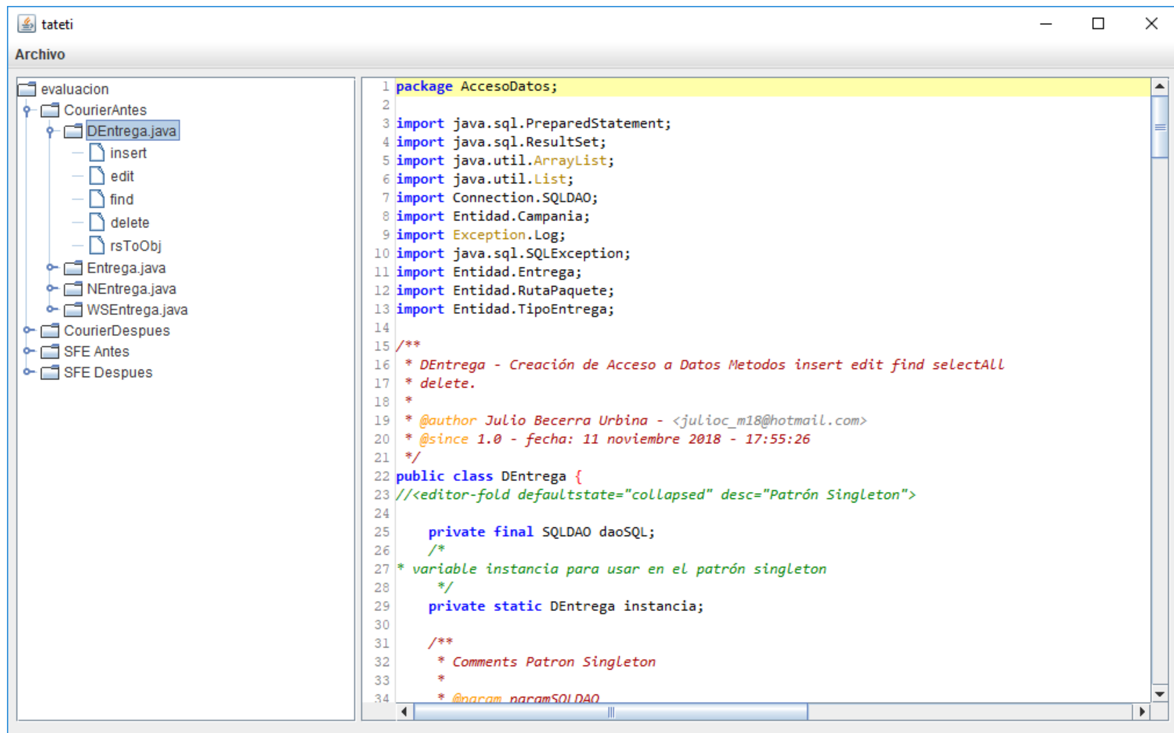
- ✓ Utilizar una base de datos relacional como fuente de información estructural para generar las reglas de generación de código como nombres de tablas, columnas, claves primarias, claves foráneas y otras restricciones.
- ✓ Optimizar el tiempo de respuesta de los CRUD's utilizando REST como puente entre la vista y el servidor.
- ✓ Integrar las librerías JQuery modernas para mejor performance y tiempo de respuesta de los CRUD's.
- ✓ Integrar estilos CSS para mejorar la usabilidad de las vistas HTML.
- ✓ Asegurar que el código generado sea mantenible.

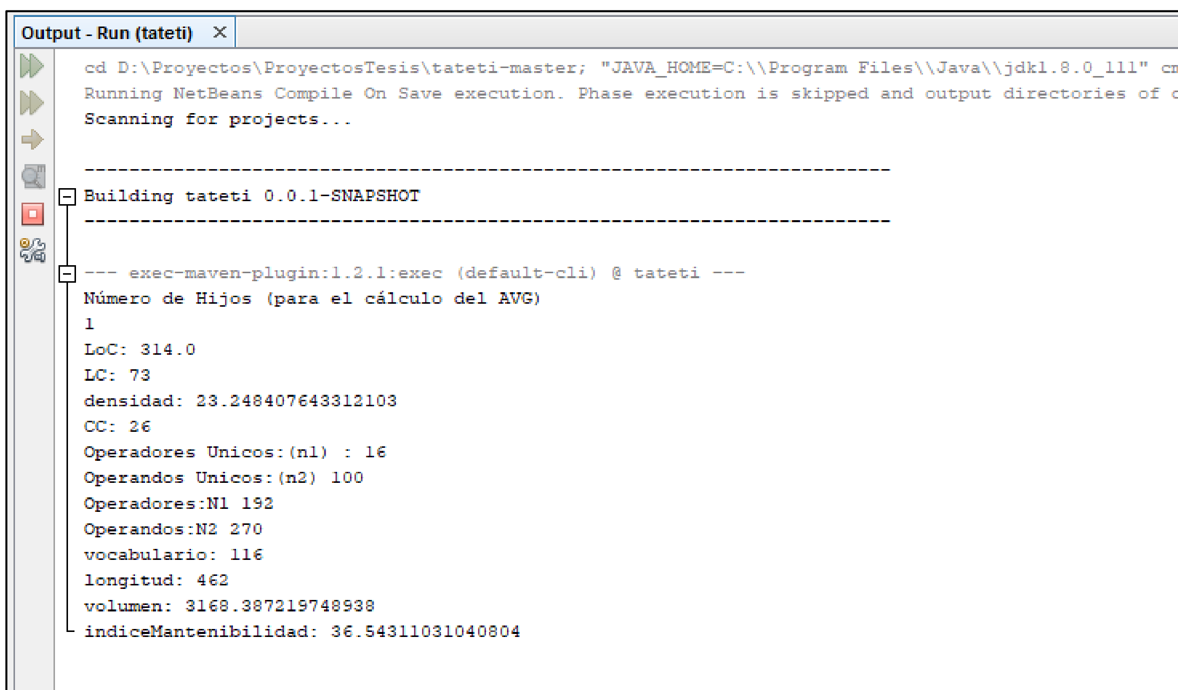
Descripción de los interesados

Nombre	**Descripción**	**Responsabilidades**
Desarrollador	Desarrollador de Software.	Encargado de desarrollar el Software generador de Código.
Tester	Encargado de realizar las pruebas de software.	Revisar que el software cumpla con los requerimientos. Revisar que el software funcione correctamente sin colgarse. Revisar que los mensajes de error sean claros y ayuden al usuario a resolverlos.
Programador (Usuario)	El usuario final del software.	Ser parte del equipo de pruebas de usuario.

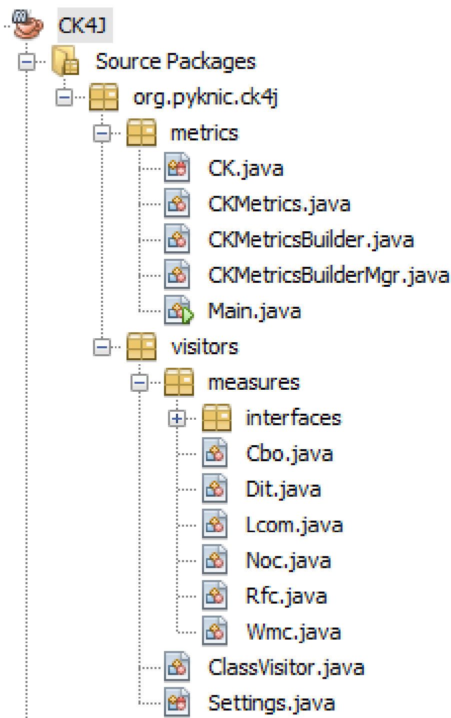
Anexo nro. 8: Análisis de Código

a) Análisis de código con Tateti Software





b) Análisis de Código con CK4J Software

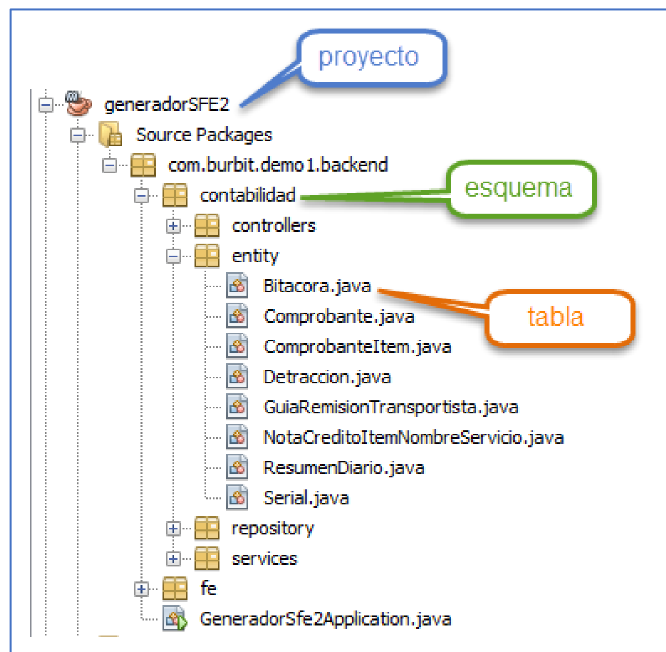



```java
public class Main {
    public static void main(String... params) {

        System.out.println("Starting CK4J.");
        String jarPath = "D:\\Entregables tesis\\demoProyectos\\evaluacion\\SFE Despues\\";
        jarPath +="SFE2-0.0.1-SNAPSHOT.jar";
        final CKMetricsBuilderMgr mgr = new CKMetricsBuilderMgr();

        System.out.println("Attempting to load '" + jarPath + "'.");
        try {
            final JarFile jar = new JarFile(new File(jarPath));
            mgr.visitAll(
                jar.stream()
                    .filter(e -> e.getName().endsWith(".class"))
                    .map(e -> {
                    try {
                        final ClassParser cp = new ClassParser(jar.getInputStream(e), e.getName());
```

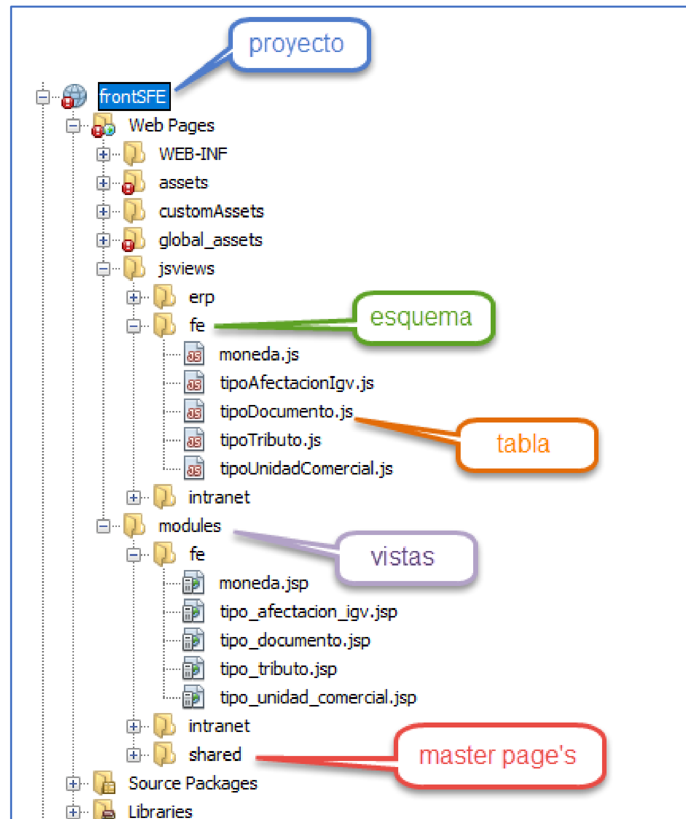
```
- Run (CK4J) ×
cd D:\Entregables tesis\CK4J-master; "JAVA_HOME=C:\\Program Files\\Java\\jdk1.8.0_111" cmd /c "\"\"D:\\Programs\\Netbeans\\
Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of dependency projects (with
Scanning for projects...

------------------------------------------------------------------------
Building CK4J 1.0-SNAPSHOT
------------------------------------------------------------------------

--- exec-maven-plugin:1.2.1:exec (default-cli) @ CK4J ---
Starting CK4J.
Attempting to load 'D:\Entregables tesis\demoProyectos\evaluacion\SFE Despues\SFE2-0.0.1-SNAPSHOT.jar'.
Clase:    com.burbit.sf2.backend.fe.entity.TipoAfectacionIgv : (WMC: 9, NOC: 0, RFC: 10, CBO: 8, DIT: 1)
Clase:    com.burbit.sf2.backend.fe.services.TipoAfectacionIgvServiceImpl : (WMC: 7, NOC: 0, RFC: 15, CBO: 6, DIT: 1)
Clase:    com.burbit.sf2.backend.fe.repository.ITipoAfectacionIgvRepository : (WMC: 0, NOC: 0, RFC: 0, CBO: 1, DIT: 1)
Clase:    com.burbit.sf2.backend.fe.controllers.TipoAfectacionIgvController : (WMC: 7, NOC: 0, RFC: 25, CBO: 14, DIT: 1)
Clase:    com.burbit.sf2.backend.fe.services.ITipoAfectacionIgvService : (WMC: 6, NOC: 0, RFC: 6, CBO: 1, DIT: 1)
Closing down.
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 1.673s
Finished at: Thu Nov 14 07:49:45 COT 2019
Final Memory: 5M/245M
------------------------------------------------------------------------
```

Anexo nro. 9: Estructura del Proyecto Generado

Arquitectura Backend:

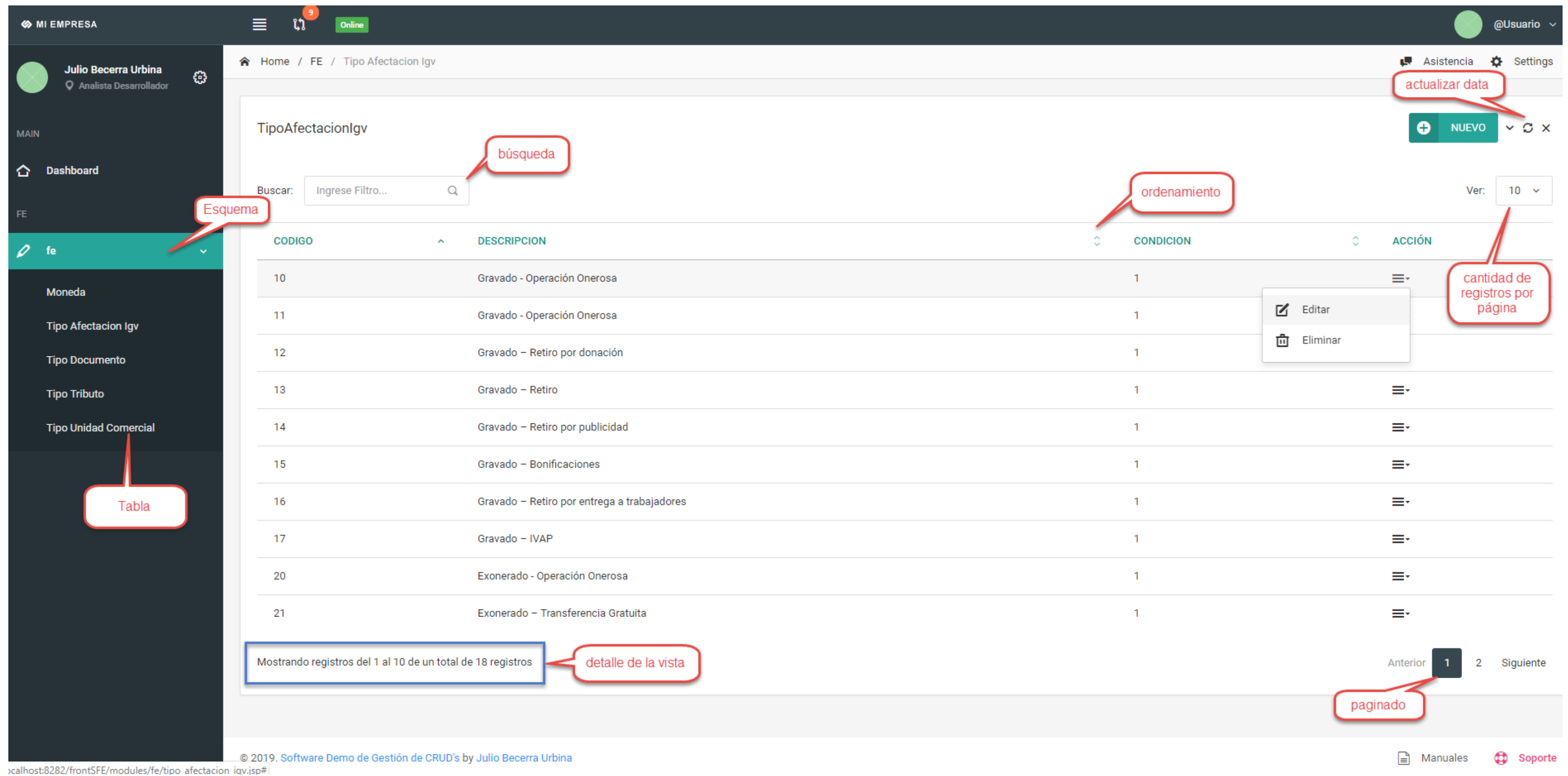


Estructura Frontend:



Anexo nro. 10: Vistas del Proyecto Generado

En el siguiente gráfico se muestra el panel de vista de datos.



En el siguiente gráfico se muestra el formulario para ingresar datos.

Empleado

REGRESAR

Examples of standard form controls supported in an example form layout. Individual form controls automatically receive some global styling. All textual `<input>`, `<textarea>`, and `<select>` elements with `.form-control` are set to `width: 100%;` by default. Wrap labels and controls in `.form-group` for optimum spacing. Labels in horizontal form require `.col-form-label` class.

Cargo: CHOFER CAMION

Nombre *: Julio

Apellido Paterno *: Becerra

Apellido Materno *: Urbina

Email *: julio@mail.com

Direccion *: San Fernando #654

Fecha Nacimiento *: 2019-07-03

Dni *: 12345687

Sexo *

Condicion *: – 1 +

Atención! Este es un espacio para colocar información relevante a **Gestión de Empleado**.

GUARDAR