REVERSE ENGINEERING FOR SOFTWARE MAINTENANCE: A SYSTEMATIC REVIEW OF LITERATURE*

Raúl Antonio Aguilar Vera[**]
José De La Rosa Baeza Pérez[***]
Julio César Díaz Mendoza[****]
Antonio Armando Aguileta Güemez[*****]

Received: 03/10/2023 • Accepted: 14/02/2024
https://doi.org/10.22395/rium.v23n44a2

Abstract

Reverse engineering covers a wide range of software life cycle activities starting with the existing implementation, recovering, or recreating the design and deciphering the requirements implemented in the system. The objective of this article is to present a state of the art on Reverse Engineering techniques used in the context of software maintenance. For this purpose, a Systematic Literature Review was carried out, in which 47 primary studies from 4 databases, published in this century, were analyzed. Research questions were raised regarding the distribution of publications throughout this century and, about the type of maintenance on which they focus; Likewise, the techniques and metrics reported for reverse engineering processes in maintenance activities were analyzed. A significant increase (just over 200 %) could be observed in the second decade of this century compared to the first. On the other hand, it was observed that perfective maintenance is the one in which the most techniques were identified, although the majority are static. From the analysis of the selected studies, a total of 22 types of reengineering techniques used

* This article is the result of research developed at the Faculty of Mathematics of the Autonomous University of Yucatan, Mérida, Mexico. The results were presented at the Ibero-American Conference on Software Engineering and Knowledge Engineering 2023.

** Academic Body of Software Engineering for Education - Autonomous University of Yucatan, Merida, Mexico (E.mail: avera@correo.uady.mx) Orcid: https://orcid.org/0000-0002-1711-7016

*** Faculty of Mathematics - Autonomous University of Yucatan, Merida - Mexico (E.mail: josedelarosass@gmail.com)

**** Academic Body of Software Engineering for Education - Autonomous University of Yucatan, Merida, Mexico (E.mail: julio.diaz@correo.uady.mx) Orcid: https://orcid.org/0000-0002-3005-3432

***** Academic Body of Software Engineering for Education - Autonomous University of Yucatan, Merida, Mexico (E.mail: aaguilet@correo.uady.mx) Orcid: https://orcid.org/0000-0001-5155-3543

for software maintenance/evolution tasks were identified, reporting a set of eight types of metrics used to evaluate the quality of reengineering processes in the evolution of the software.

Keywords: Software Engineering, Software Maintenance, Systematic Literature Review, Reverse Engineering Techniques.

INGENIERÍA INVERSA PARA EL MANTENIMIENTO DEL SOFTWARE: UNA REVISIÓN SISTEMÁTICA DE LITERATURA

Resumen

La ingeniería inversa cubre una amplia gama de actividades del ciclo de vida del software comenzando con la implementación existente, recuperando o recreando el diseño y descifrando los requisitos implementados en el sistema. El objetivo de este artículo es presentar un estado del arte sobre técnicas de Ingeniería Inversa utilizadas en el contexto del mantenimiento de software. Para ello se realizó una Revisión Sistemática de la Literatura, en la que se analizaron 47 estudios primarios de 4 bases de datos, publicados en este siglo. Se plantearon interrogantes de investigación respecto de la distribución de las publicaciones a lo largo de este siglo y, sobre el tipo de mantenimiento en el que se enfocan; Asimismo, se analizaron las técnicas y métricas reportadas para los procesos de ingeniería inversa en las actividades de mantenimiento. En la segunda década de este siglo se pudo observar un aumento significativo (poco más del 200 %) en comparación con la primera. Por otro lado, se observó que el mantenimiento perfectivo es en el que más técnicas se identificaron, aunque la mayoría son estáticas. Del análisis de los estudios seleccionados se identificaron un total de 22 tipos de técnicas de reingeniería utilizadas para tareas de mantenimiento/evolución del software, reportando un conjunto de ocho tipos de métricas utilizadas para evaluar la calidad de los procesos de reingeniería en la evolución del software.

Palabra Clave: Ingeniería de Software, Mantenimiento de Software, Revisión Sistemática de Literatura, Técnicas de Ingeniería Inversa.

INTRODUCCIÓN

La Ingeniería de Software (IS) dispone desde hace prácticamente dos décadas de un cuerpo de conocimientos aceptado por la academia y por la industria del software; dicho cuerpo integra diez áreas de conocimiento relacionadas con los procesos de desarrollo y de gestión del software, las cuales fueron documentadas en 2004 [1] y actualizadas en 2014 [2]. Una de las áreas que ha sufrido una constante evolución es la conocida como "Mantenimiento Software"; fue abordada por primera vez en 1969 y dicho proceso se centra en la modificación de un producto software después de su entrega para corregir errores, mejorar funcionalidades, etcétera.

Con el paso del tiempo, se han notado ciertas desventajas al aplicar este proceso, como costos muy elevados, exceso de esfuerzo al aplicar algún cambio, entre otras cosas; por esta razón, se han ido desarrollando nuevas técnicas para llevar a cabo las actividades vinculadas con dicha tarea, y al igual que en otras ingenierías, las técnicas de ingeniería inversa se han venido adaptando a los procesos vinculados con el mantenimiento de software.

Si bien la ingeniería inversa tiene su origen en el análisis de *hardware*, para mejorar productos propios, así como para analizar los productos de un competidor, en el caso de los procesos software se busca obtener una comprensión suficiente a nivel de diseño para ayudar al mantenimiento y así fortalecer la mejora o respaldar el reemplazo.

El presente estudio tiene como propósito presentar un estado del arte de las técnicas y métricas vinculadas con la ingeniería inversa, las cuales han sido utilizadas para el mantenimiento o evolución del software en las últimas dos décadas

1. MARCO TEÓRICO

El Mantenimiento del software integra la totalidad de las actividades necesarias para proporcionar soporte rentable al software; dichas actividades se realizan tanto durante la etapa previa a la entrega como durante la etapa posterior a la entrega [2]. De acuerdo con Piattini *et al.* [3] existen cuatro tipos de mantenimiento de software:

– Correctivo: tiene como objetivo localizar y eliminar posibles defectos de los programas.

– Adaptativo: consiste en modificar un programa debido a cambios de hardware o software; este tipo de mantenimiento puede ser desde un ligero cambio hasta reescribir todo el código.

– Perfectivo: son aquellas mejoras o nuevas funcionalidades que son requeridas por el usuario.

– Preventivo: se debe a las modificaciones del software para mejorar sus propiedades, sin alterar sus funcionalidades.

No obstante que se tiene identificado el propósito del mantenimiento, así como su tipología, a finales del siglo pasado aún se incurría en situaciones similares a las identificadas en la crisis del software, situaciones que se supone habían sido superadas al aplicar el proceso ingenieril. Por lo anterior, y en la búsqueda de nuevas formas de realizar los procesos de mantenimiento, se comenzó a manejar el término de "evolución de software" que Lehman y Ramil [4] definen como el comportamiento dinámico de los sistemas de programación a medida que se mantienen y mejoran a lo largo de su vida; así mismo, Lehman *et al*. [5] discuten un conjunto de métricas y leyes vinculadas con el crecimiento y evolución de los sistemas software. Dichas leyes son las siguientes:

– Cambio continuado: un programa debe adaptarse continuamente, de lo contrario se vuelve progresivamente menos satisfactorio.

– Complejidad creciente: a medida que un programa en evolución cambia, su estructura tiende a ser cada vez más compleja, a menos que se trabaje para mantenerlo o reducirlo.

– Evolución prolongada del programa: la evolución de los programas es un proceso autor regulativo. Debido a ciertos atributos del sistema son aproximadamente invariantes en cada entrega del sistema.

– Estabilidad organizacional: durante el tiempo de vida de un programa, su velocidad de desarrollo es aproximadamente constante e independiente de los recursos dedicados al desarrollo del sistema.

– Conservación de la familiaridad: durante el tiempo de vida de un sistema, el cambio incremental en cada entrega es aproximadamente constante.

– Crecimiento continuado: la funcionalidad ofrecida por los sistemas tiene que crecer continuamente para mantener la satisfacción de los usuarios.

– Decremento de la calidad: la calidad de los sistemas comenzará a disminuir a menos que dichos sistemas se adapten a los cambios en su entorno de funcionamiento.

– Realimentación del sistema: los procesos de evolución incorporan sistemas de realimentación multi-nivel y multi-bucle y estos deben de ser tratados como sistemas de realimentación para lograr una mejora significativa del producto.

Estas leyes son de mucha ayuda para realizar un mantenimiento planificado y realizar cambios pertinentes o actualizaciones, con la idea de mantener la vida útil por mucho más tiempo; sin embargo, el proceso de evolución software inicialmente

fue concebido para sistemas grandes y personalizados por grandes empresas y, de hecho, puede resultar inviable para las pequeñas y medianas empresas, debido a que se requiere recortar presupuesto para poder desarrollar nuevas funcionalidades en su software o desarrollar nuevos proyectos. Por esta razón se busca minimizar los costos de mantenimiento tratando extender la vida útil del software en vez de estar actualizando constantemente. No obstante, es aquí donde la ingeniería inversa puede ayudar en cumplir esta idea.

De acuerdo con Chikofsky y Cross [6], la ingeniería inversa se entiende como el proceso de analizar un sistema sujeto para identificar los componentes del sistema y sus interrelaciones, para así crear representaciones del sistema en otra forma o en un nivel superior de abstracción.

2. METODOLOGÍA DE ESTUDIO

Para conducir el estudio se utilizó como referencia la metodología propuesta en [6-8] para la realización de una Revisión Sistemática de Literatura (RSL). Las principales actividades de que constan las tres fases que integran dicha metodología son:

Planificación:

- Identificar la necesidad de la revisión.
- Formular las preguntas de investigación.
- Definir la estrategia de búsqueda.
- Establecer los criterios para selección de los estudios primarios.
- Definir la estrategia de evaluación de la calidad de los estudios seleccionados.

Ejecución:

- Seleccionar los estudios primarios relevantes.
- Evaluar la calidad de los estudios primarios seleccionados.
- Extraer la información de los estudios primarios seleccionados.

Elaboración del reporte:

- Elaborar un informe técnico de la revisión.

Con el propósito de incrementar el número de estudios relevantes, la RSL incorporó la estrategia denominada bola de nieve hacia adelante [9]; dicha estrategia sería aplicada al primer conjunto de Estudios Primarios Seleccionados (EPS).

3. PLANIFICACIÓN

De acuerdo con la metodología propuesta, las tareas de que consta la primera fase son las siguientes:

3.1. Necesidad de la revisión

Con el propósito de elaborar un estado del arte de las técnicas y métricas utilizadas en las últimas dos décadas en torno a la aplicabilidad de la ingeniería inversa en los procesos de mantenimiento o evolución del software, se procedió a realizar una revisión en búsqueda de trabajos previos, misma que permitió identificar un par de RSL relacionados con el tema de reingeniería.

La primera RSL [10] explora y describe los enfoques de la ingeniería inversa basados en los conceptos principales de un paradigma impulsado en modelos, para describir sus transformaciones y descubrimientos al utilizar metamodelos más abstractos. De esta manera los autores proponen las siguientes preguntas: ¿Qué metamodelos utilizan los enfoques de ingeniería inversa impulsados por modelos? ¿Están definidos para resolver problemas específicos o se reutilizan para más de un propósito? ¿Qué herramientas utilizan los enfoques para su implementación? ¿Los enfoques proporcionan nuevas herramientas o reutilizan las herramientas existentes? ¿Cuál es el nivel de automatización de las transformaciones definidas en los enfoques MDRE?

Para llevar a cabo su búsqueda, seleccionaron cinco motores de búsqueda: IEEE Explore, ACM Digital Library, Web of Science, Scopus y Science Direct. Con dicho proceso se seleccionaron 51 estudios primarios publicados entre 2003 y 2016.

Como parte de sus resultados describen en detalle quince enfoques que aplican ingeniería inversa basada en modelos y proponen sugerencias sobre cómo elegir un enfoque de ingeniería inversa basado en modelos.

La segunda RSL [11] encontrada analiza investigaciones existentes relacionadas con el proceso de ingeniería inversa en líneas de productos de software, en particular, investigaron a la ingeniería inversa en el proceso de ingeniería de dominios (requisitos de dominio, diseño y realización de dominio, garantía de calidad de dominio). El estudio plantea tres preguntas de investigación: ¿Cómo identificar características comunes y de variabilidad en carteras de aplicaciones similares? ¿Cuáles son las actividades en ingeniería de dominio que maneja esta ingeniería inversa? Y, ¿cuál es la evidencia que motiva la adopción de los métodos de ingeniería inversa existentes y su limitación?

El estudio analiza 71 estudios primarios publicados entre 2008 y 2018 de las bases de datos IEEE Xplore, ScienceDirect y Scopus.

Como parte de los resultados se destaca el que se necesitan métodos de ingeniería inversa que sean capaces de identificar y mantener una correlación consistente entre las características de la ingeniería de aplicaciones y la ingeniería de dominio en el proceso de ingeniería inversa.

3.2. Preguntas de investigación

Con el objetivo de realizar un análisis de la investigación generada en los últimos veintidós años referente en el campo de la ingeniería inversa aplicada a la evolución de software, se plantearon las siguientes preguntas de investigación:

PI1. ¿Cuál es la distribución de los estudios primarios publicados en torno a la ingeniería inversa en la tarea de mantenimiento/evolución de software?

PI2. ¿Cuál es la distribución de los estudios primarios publicados, en función del tipo de mantenimiento analizado y del tipo de técnica utilizada?

PI3. ¿Cuáles son las técnicas de reingeniería que han sido utilizadas en mantenimiento/evolución de software?

PI4. ¿Cuáles son las métricas utilizadas para evaluar la calidad de los procesos de reingeniería en mantenimiento/evolución de software?

3.3. Estrategia de búsqueda

El conjunto de bases de datos seleccionadas para el proceso de búsqueda fue: IEEE Xplore, ACM Digital Library, ScienceDirect y SpringerLink. Para la realización de la búsqueda se creó una cadena con los términos que nos permitirán recopilar estudios sobre el tema. La cadena general construida es:

"Reverse Engineering" AND "Techniques" AND "Software"

Finalmente, con base en el conjunto de estos estudios seleccionados, se propuso el aplicar la técnica de bola de nieve hacia adelante, con la intención de incrementar y mejorar el conjunto de estudios primarios relevantes para la revisión.

3.4. Criterios de selección

Para la selección de estudios primarios se consideraron un conjunto de criterios de inclusión y de exclusión, los cuales servirán para evaluar la pertinencia de los estudios obtenidos con la cadena de búsqueda en cada una de las BD.

Criterios de inclusión:

– CI1. Estudios publicados en revistas especializadas.

– CI2. Estudios publicados en la ventana de tiempo de 2000 a 2021.
– CI3. Estudios publicados en idioma inglés.
– CI4. Estudios de investigación empíricos vinculados con las técnicas de ingeniería inversa para el mantenimiento de software.

Criterios de exclusión:

– CE1. Artículos duplicados o publicados con versiones similares entre los repositorios o base de datos.
– CE2. Estudios secundarios o terciarios.

3.5. Evaluación de la calidad

La evaluación de la calidad de los estudios primarios seleccionados es imperativa para asegurar que dichos estudios no comprometan la calidad de la investigación y que, por tanto, se tenga la seguridad de que los resultados sean válidos. En el presente estudio se utiliza como referencia el instrumento propuesto por Dybå y Dingsøyr [12], el cual utiliza los criterios más recurrentes en la literatura: credibilidad, relevancia y rigor, así como un cuarto criterio denominado calidad del informe. La planeación del mecanismo de evaluación es descrita en detalle por Baeza y Aguilar [13].

4. EJECUCIÓN

Las tareas de que consta la segunda fase de la metodología son las siguientes:

4.1. Selección de los estudios relevantes

Tabla 1. Cadenas configuradas por motor de búsqueda

Base de datos	Cadena configurada
IEEE Xplore	*("All Metadata": Reverse Engineering) AND ("All Metadata": Techniques) AND ("All Metadata": Software)*
ACM Digital Library	*[All: "reverse engineering"] AND [All: techniques] AND [[All: "software maintenance"] OR [All: "software evolution"]]*
Science Direct	*"Reverse Engineering" AND Techniques AND ("Software maintenance" OR "Software Evolution")*
Springer Link	*"Reverse Engineering" AND Techniques AND ("Software maintenance" OR "Software Evolution")*

Fuente: elaboración propia.

Para la extracción de los estudios primarios, la primera tarea consistió en configurar la cadena construida en la fase de planeación, en función del motor de búsqueda de cada una de las bases de datos seleccionada (véase la tabla 1).

Los resultados del proceso de selección, con los tres filtros definidos de acuerdo con la estrategia, se ilustran en la tabla 2.

Tabla 2. Proceso de Selección de los EP

Base de datos	Cadena conf.	C-Inclusión	C-Exclusión
IEEE Xplore	1383	83	15
ACM Digital Library	1038	64	8
Science Direct	258	72	2
Springer Link	676	56	9

Fuente: elaboración propia.

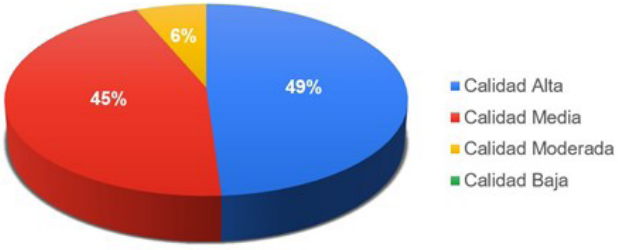
Finalizada esta primera etapa del proceso de selección, se procedió a la aplicación de la técnica bola de nieve hacia adelante a los 34 estudios primarios seleccionados. Para dicha tarea se utilizó el repositorio de *Google Académico*, ya que permite identificar en cada uno de los estudios el número de citaciones que ha tenido, y a su vez, brinda la posibilidad de revisar los resúmenes de todos los estudios que lo han citado, para luego aplicar los criterios de inclusión y exclusión definidos; de dicho proceso se obtuvo un total de 13 artículos adicionales, haciendo un total de 47 estudios primarios seleccionados para su revisión, análisis y síntesis; los datos generales de dichos estudios pueden ser consultados a través el siguiente enlace:

https://docs.google.com/spreadsheets/d/1XToKZv21BHBURbuVcebbtrU_ZO4QI-tTXVT5OQDX_bM/edit?usp=sharing

4.2. Evaluación de la calidad de los estudios seleccionados

Con base en el proceso de evaluación de la calidad se procedió a la valoración individual de cada uno de los 47 estudios seleccionados. En la figura 1 podemos observar que el 94 % de los EPS fue valorado entre calidad alta y media, y también es de identificar que ninguno de los estudios fue valorado como de calidad baja.

Figura 1. Evaluación de la calidad para los 47 EPS



Fuente: elaboración propia.

Con respecto a la evaluación global de los indicadores de calidad para cada uno de los cuatro criterios definidos, así como del análisis de correlación entre dichos indicadores, la misma puede ser consultada en [13].

5. REPORTE DE RESULTADOS

Finalmente, la tercera fase de que consta la metodologóa tiene como finalidad dar respuesta a las preguntas que condicen la investigación.

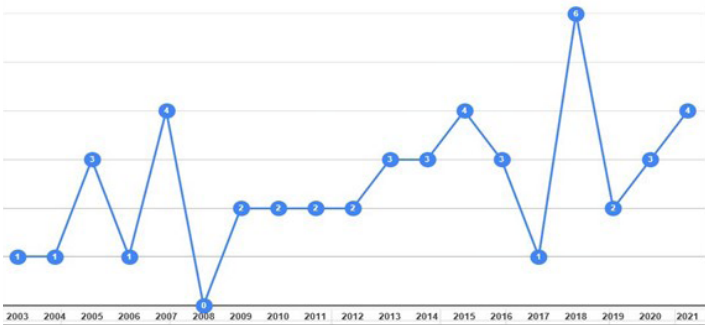
5.1. Respuesta a las preguntas de investigación

Con base en el análisis de los 47 estudios primarios, se presentan a continuación las respuestas a las preguntas de investigación (PI) definidas en la fase de planeación:

PI1. *¿Cuál es la distribución de los estudios primarios publicados en torno a la ingeniería inversa en la tarea de mantenimiento/evolución de software?*

La figura 2 ilustra la distribución de los EPS, y si bien no se observa una clara tendencia en la producción de los estudios vinculados con el tema de la ingeniería inversa para el mantenimiento de software, podemos observar que en la primera década del presente siglo se tuvieron un total de 14 estudios generados, mientras que en la segunda década el número se triplica, teniendo en promedio tres artículos por año.

Figura 2. Distribución de los EPS por año de publicación

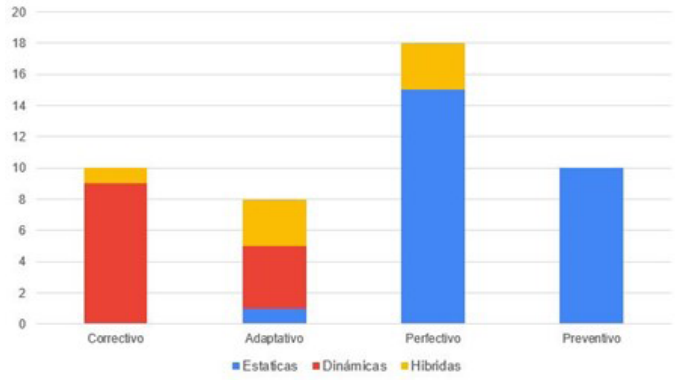


Fuente: elaboración propia.

PI2. *¿Cuál es la distribución de los estudios primarios publicados, en función del tipo de mantenimiento analizado y del tipo de técnica utilizada?*

Con base en los cuatro tipos de mantenimiento de software, la figura 3 ilustra el número de estudios que abordan los tipos de mantenimiento en función de los tipos de técnica.

Figura 3. Distribución de los EPS por tipo de mantenimiento y tipo de técnica



Fuente: elaboración propia.

Podemos observar que, en el caso del mantenimiento correctivo, las técnicas dinámicas son las que prevalecen. En el caso del mantenimiento adaptativo, si bien las dinámicas son las más utilizadas, existe mayor variedad en cuanto al tipo de técnica recurrida; por su parte, en cuanto al mantenimiento perfectivo y preventivo, las técnicas estáticas son las que presentan mayor moda. Incluso en la última categoría solamente se identifican dicho tipo de técnicas.

PI3. *¿Cuáles son las técnicas de reingeniería que han sido utilizadas en mantenimiento/evolución de software?*

Del análisis a los 47 EPS se pudieron identificar un total de 22 tipos de técnicas de reingeniería utilizados para tareas de mantenimiento/evolución de software.

– *Búsquedas automatizadas.* Los algoritmos de búsqueda de patrones se pueden clasificar en diferentes categorías: aquellos que buscan estructuras clave mínimas, aquellos que buscan estructuras de clase, aquellos que utilizan lógica difusa y aquellos que se basan en métricas. Cada algoritmo puede tener valores específicos para evaluar la calidad del proceso de reconocimiento.

– *Pruebas automatizadas.* Las herramientas de captura/reproducción son herramientas interactivas que facilitan la automatización de secuencias de comandos manuales al proporcionar soporte mediante un mecanismo que permite al probador registrar las interacciones con la interfaz de usuario. Estas interacciones se guardan como casos de prueba que se pueden reproducir automáticamente, lo que reduce la carga de trabajo manual para el probador.

– *Detección de patrones de cambio.* Los patrones de cambio describen la relación entre dos o más archivos que suelen ser modificados simultáneamente durante el desarrollo

o el mantenimiento de sistemas de software. A diferencia de las estrategias de refactorización, el conocimiento obtenido de estos patrones puede ser reutilizado para orientar la gestión de cambios futuros y la evolución de las arquitecturas de software.

– *Detección de olores en el código*. La identificación y unificación de olores simples resulta beneficiosa en el mantenimiento del software, ya que permite obtener aún mayores beneficios al elevar el nivel de análisis del código. Algunos olores pueden ser eliminados mediante refactorización, sustituyéndolos por llamadas a funciones o macros, o empleando técnicas de meta-nivel no convencionales como la programación orientada a aspectos, con el fin de evitar los efectos perjudiciales de los olores.

– *Técnicas de localización*. Se orientan en la identificación de los elementos del código fuente que implementan funcionalidades observables por el usuario. En el ámbito de la evolución del software, muchas tareas requieren la documentación, configuración, adición, eliminación o mejora de estas funcionalidades, lo que implica la necesidad de localizar el código correspondiente.

– *Agrupamiento de software*. Su objetivo es realizar la agrupación automática y no supervisada de artefactos de software, como funciones, clases o archivos, en estructuras de nivel superior como paquetes, componentes o subsistemas, basándose en su similitud.

– *Agregación*. La primera característica de la agregación se relaciona con su estructura estática, que incluye la transitividad, la anti simetría y la irreflexividad. Tanto la propiedad de anti simetría como la de irreflexividad se refieren a nivel de instancia. La propiedad de anti simetría establece que un todo no puede ser parte de su parte, mientras que la irreflexividad sostiene que un objeto no puede ser simultáneamente el todo y la parte. La segunda característica se refiere a las relaciones vinculantes de por vida entre el objeto completo y el objeto parcial. La propiedad del todo sobre la parte es la tercera característica.

– *Evoluciones arquitectónicas*. Dada la importancia de la evolución y la arquitectura en muchos sistemas, se pueden considerar dos perspectivas: en primer lugar, el diseño arquitectónico debe ser capaz de facilitar la evolución del sistema; en segundo lugar, el proceso de evolución debe ser consciente de la arquitectura del sistema y respaldar su papel conceptual.

– *Especificaciones Z*. Este enfoque combina estrategias de mapeo existentes y permite asignar métodos basados en medidas de calidad. La idea principal es equilibrar los valores de acoplamiento de todos los métodos dentro y entre las clases UML.

- *Minería de aspectos.* Se utiliza para recopilar datos. Este enfoque es efectivo y conceptualmente claro, pero presenta un importante inconveniente: la integración de las herramientas AOP con el sistema de compilación resulta ser un desafío significativo. Además, nos ayuda a identificar preocupaciones transversales que van más allá de las convencionales, como el registro y el manejo de errores, lo cual puede generar preocupaciones que son transversales pero difíciles de modular con la tecnología de aspecto actual.

- *Historiales de cambio.* El enfoque general de este método consiste en extraer y comparar las diferentes versiones del esquema de la base de datos almacenadas en el sistema de control de versiones. Esto permite generar lo que se conoce como esquema histórico global, una representación visual y navegable de cómo ha evolucionado el esquema de la base de datos a lo largo del tiempo.

- *Representación intermedia.* La representación intermedia (IR) debe ser factible y permitir la recuperación de un código fuente preciso que pueda ser analizado por ingenieros. Esto facilitará la recopilación y obtención de un binario reescrito funcional. Una representación viable asegura que el programa pueda ser probado y modificado utilizando técnicas de depuración estándar.

- *Remodularización.* Permite mejorar el diseño de un sistema de software y corregir cualquier defecto causado por la evolución del software. Se han propuesto diversos enfoques para ayudar a los desarrolladores durante el proceso de remodularización de un sistema de software. La mayoría de estos enfoques se basan en la suposición de que los desarrolladores buscan encontrar un equilibrio óptimo entre cohesión y acoplamiento al modularizar las clases de sus sistemas.

- *Vistas polimétricas.* Son útiles para comprender la estructura y detectar problemas de un sistema de software en las etapas iniciales de la ingeniería inversa. Se ha desarrollado una metodología basada en *clusters* de diferentes vistas polimétricas para brindar apoyo y guía a los ingenieros de software en las primeras etapas de la ingeniería inversa de sistemas de software a gran escala.

- *Reutilización*. Es considerada como una estrategia clave en el diseño y desarrollo de portales web colaborativos. Su utilización ha demostrado mejorar la mantenibilidad de los proyectos web, así como reducir el número de fallos durante las pruebas. Esto representa un paso significativo hacia la ingeniería de aplicaciones web.

- *Agrupación*. Tiene como objetivo disminuir la complejidad de un sistema de software grande al reemplazar un conjunto de artefactos con un clúster, que actúa como una abstracción que representa a todos los artefactos agrupados dentro de

él. De esta manera, se logra una descomposición más comprensible. No obstante, también implica una reducción en la cantidad de información transmitida por la representación agrupada del sistema de software.

– *Detección de patrones*. La comprensión del diseño de un sistema es fundamental para su posterior mantenimiento y desarrollo. Sin embargo, debido a la naturaleza de los sistemas de software, que suelen ser grandes, complejos y desarrollados en plazos ajustados con requisitos cambiantes, es común que el diseño esté insuficientemente documentado. Por tanto, resulta crucial respaldar la comprensión del software con herramientas de análisis de diseño, como los patrones de diseño; además, la identificación de anti-patrones, es decir, la detección de errores de diseño también puede ser valiosa para comprender un sistema de software de manera más completa.

– *Detección de fusiones y divisiones.* Estas técnicas de refactorización son actividades comunes en el desarrollo activo y el mantenimiento de software y suelen utilizarse para mantener el código base en un estado saludable y ágil. Los desarrolladores de software suelen emplear la combinación y la división para reducir la complejidad del sistema, facilitando su comprensión y evolución.

– *Sistemas basados en agentes*. Esta técnica tiene como objetivo permitir la comparación y el contraste de los marcos de agentes existentes y futuros, así como proporcionar una base para identificar áreas que requieren estandarización dentro de la comunidad de agentes.

– *Predicción de defectos.* Permite una asignación eficiente de los recursos de prueba y la toma de decisiones informadas con respecto a la mejora de la calidad del lanzamiento. Durante mucho tiempo, los ingenieros de software han mostrado interés en predecir la calidad de los sistemas de software en desarrollo. Un enfoque específico consiste en predecir los defectos de software y su posible ubicación, como, por ejemplo, identificar los módulos que podrían ser particularmente propensos a tener defectos.

– *Mantenimiento centrado en protocolos.* La extracción de palabras clave del protocolo del seguimiento de la red es una tarea crítica en la ingeniería inversa de protocolos. Dado que los mensajes de diferentes tipos presentan conjuntos distintos de palabras clave de protocolo, agrupar la carga útil de tráfico desconocido en grupos y analizar cada uno de ellos resulta un método efectivo para mejorar la precisión en la extracción de palabras clave de protocolo.

- *Apps colaborativas*. La idea central de una técnica de inversión colaborativa es utilizar estructuras de datos correspondientes como referencias para analizar y acceder a distintos campos de entrada.

PI4. *¿Cuáles son las métricas utilizadas para evaluar la calidad de los procesos de reingeniería en la evolución del software?*

Muchos de los estudios analizados, desarrollaron métricas *ad hoc* para evaluar criterios específicos vinculados con la evolución del software; otros por el contrario utilizaron métricas un poco más conocidas, pero que fueron implementadas con características particulares. A continuación, se describen brevemente dichas métricas identificadas en la revisión de la literatura.

– *Recuperación y precisión*. Estas métricas son ampliamente utilizadas en todas las técnicas de recuperación de información. La métrica de precisión evalúa la corrección de un enfoque, mientras que la métrica de recuperación mide su integridad. La métrica de puntuación F combina la precisión y la recuperación para evaluar su efecto combinado.

– *Métricas para patrones.* Estas métricas se calculan para cada uno de los patrones deseados y, posteriormente, se asigna una firma a cada patrón, que es esencialmente una lista de valores correspondientes a las métricas descritas anteriormente. Además, se calculan las métricas para cada clase, incluyendo todas las subclases del sistema evaluado. Los autores describen ciertas métricas asociadas a cada patrón: métricas OO, métricas estructurales y métricas de procedimiento.

– *Métricas de reutilización y eficiencia*. Para cuantificar los esfuerzos necesarios a fin de implementar un cambio en la arquitectura del software, se utiliza una métrica llamada "Operaciones de Cambio Total" (TCO), que consiste en contar el número de operadores de cambio requeridos para realizar dicho cambio; esta métrica se adopta para contar las declaraciones u operaciones necesarias en la implementación del cambio. En el diseño y la evolución del software, se utilizan métricas tradicionales como el acoplamiento y la cohesión para medir los impactos del cambio y analizar la flexibilidad del software. Sin embargo, no existe una métrica estándar disponible para cuantificar los esfuerzos necesarios o la cantidad de cambios requeridos para implementar un cambio específico y medir la eficiencia y la reutilización de los patrones de cambio de arquitectura.

– *Validaciones cruzadas.* La validación cruzada de 10 veces se utiliza para evaluar modelos de aprendizaje automático dividiendo el conjunto de datos en 10 partes y repitiendo el proceso 10 veces. En cada repetición, una parte distinta de los datos se selecciona como conjunto de prueba, mientras que las demás partes se utilizan para entrenar el modelo.

– *Abanico de métodos.* La métrica utilizada en la minería de aspectos se basa en el concepto de "abanico de métodos", el cual representa la cantidad de métodos que llaman a otro método. Esta métrica recopila el conjunto de llamadas potenciales para cada método, y su cardinalidad proporciona el valor del *fan-in* requerido. Sin embargo, el valor real del abanico de métodos depende de cómo se consideren los métodos polimórficos, tanto como llamadores y como destinatarios.

– *Bunch.* Se utiliza una representación del sistema conocido como Gráfico de Dependencia del Módulo (MDG), que se compone de un grafo G = (V, E). En este grafo, los nodos V representan componentes de código y los bordes E representan las relaciones entre dichos componentes, como las llamadas a métodos. A través de un algoritmo de agrupación de módulos de software, se obtiene una partición del MDG (ponderada o no ponderada).

– *AMOGA.* Se propone generar un modelo de estado de la interfaz de usuario (UI) en el que los bordes del modelo representan casos de prueba. Con el fin de demostrar la utilidad del modelo generado, se ha desarrollado una herramienta de generación de pruebas que permite generar casos de prueba.

– *Evaluaciones de desempeño.* En primer lugar, se definen tres medidas base para evaluar el rendimiento de las herramientas y enfoques de ingeniería inversa:

Cl. Representa la fracción de clases detectadas correctamente.

Sub. Indica la fracción de subpaquetes descompuestos correctamente.

Rel. Mide la fracción de relaciones detectadas con éxito, es decir, las relaciones que se detectaron correctamente y son del tipo correcto.

Cada una de estas medidas se define como una función que toma un sistema para realizar ingeniería inversa (s) y un resultado de ingeniería inversa (r), que es un diagrama de clases UML estructural obtenido mediante un enfoque de ingeniería inversa aplicado a (s).

6. CONCLUSIONES

De la RSL realizada se seleccionaron un conjunto de 47 estudios primarios, evaluados y clasificados en un 94 % de los mismos como de calidad alta o media. En cuanto a su distribución en el presente siglo, no se observa una tendencia en los mismos, pero se observa que en la segunda década el número de publicaciones se triplica respecto de los de la primera década, en promedio de 3 publicaciones por año de 2011 a la fecha. Del análisis a los EPS, se logró identificar un conjunto de 22 técnicas, las cuales fueron

descritas brindando la posibilidad al ingeniero de software en el futuro elegir la que le beneficie para agilizar el proceso de entender el sistema o mejorar ciertas características a las cuales se requiere dar mantenimiento. Por otro lado, se analizó la manera en la cual cada uno de los estudios llevó a cabo la implementación de la técnica para cumplir y satisfacer sus requerimientos. Finalmente, se ofrecen en este estudio un conjunto de ocho métricas que han sido utilizadas para evaluar la calidad de los procesos de reingeniería en la evolución del software; se pudo observar que la mayoría han sido desarrolladas *ad hoc* a los procesos analizados por cada estudio.

Con base en la investigación realizada, se logró identificar tópicos de interés actuales; ciertos estudios se fijaron en verificar, analizar y probar aspectos de GUI, UI, entradas y salidas del sistema en relación con las aplicaciones web y móviles. Este es un buen punto para destacar, debido a que gran parte de los estudios analizados se centran en brindar mejoras en relación con el código, la documentación o diseño, y aunque muy pocos programadores se centran en dichos aspectos, es importante debido a que se deberían probar aquellas interacciones que podría tener un usuario, y así evitar fallas o problemas de seguridad.

REFERENCIAS

[1] Bourque P, Dupuis R, editors. *Swebok. Guide to the Software Engineering Body of Knowledge*, 2004 Version. Los Alamitos, CA: IEEE Computer Society, 2004.

[2] Bourque P, Farley R, editors. *Swebok V3.0. Guide to the Software Engineering Body of Knowledge*. Los Alamitos, CA: IEEE Computer Society, 2014.

[3] Piattini M, Villalba J, Ruiz J, Bastanchury M, Polo M, Martínez M *et al*. *Mantenimiento del software. Modelos, técnicas y métodos para la gestión del cambio.* Alfaomega & Ra-Ma, 2000.

[4] Lehman MM, Ramil JF. Software evolution—Background, theory, practice. *Information Processing Letters*, 2003 Oct; 88 (1-2): 33-44.

[5] Lehman MM, Ramil JF, Wernick PD, Perry DE, Turski WM. Metrics and Laws of Software Evolution-The Nineties View. In: *Proceedings Fourth International Software Metrics Symposium*, IEEE, 1997: 20-32.

[6] Chikofsky E, Cross J. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 1990 Jan; 7 (1): 13-17.

[7] Genero M, J. Cruz-Lemus J, Piattini M. Métodos de investigación en ingeniería de software. Madrid: Ed. Ra-Ma, 2014.

[8] Kitchenham B, Charters S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*, Version 2.3. EBSE Technical Report, 2007.

[9] Wohlin C. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In: *EASE '14: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014 May, Art. 38: 1-10.

[10] Raibulet C, Fontana F, Zanoni M. Model-Driven Reverse Engineering Approaches: A Systematic Literature Review. *IEEE Access*, 2017 May; 5: 14516-14542.

[11] Hasbi M, Budiardjo E, Wibowo W. Reverse engineering in software product line – A systematic literature review. In: *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence, CSAI 2018 - 2018 the 10th International Conference on Information and Multimedia Technology, ICIMT 2018*. Association for Computing Machinery, 2018: 174-179.

[12] Dybå T, Dingsøyr T. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 2008 Aug; 50 (9-10): 833-859.

[13] Baeza J, Aguilar R. Quality Assessment for Selected Primary Studies in a Systematic Literature Review: A Case Study. In: *Proceedings of the 2022 Mexican International Conference on Computer Science (ENC), Xalapa, Veracruz, México,* 24-26 August 2022. DOI: 10.1109/ENC56672.2022.9882921