Recibido 30 Dic. 2024
Aceptado 04 Feb. 2025
ReCIBE, Año 14 No. 1, Mzo. 2025

Facilidad de evolución y mantenimiento de software: una revisión de la literatura

Software evolvability and maintainability: a literatutre review

Mario Dorantes Hernández[1]
zs18019633@estudiantes.uv.mx

María Karen Cortés Verdín[1]
kcortes@uv.mx

Ángeles Arenas Valdés[1]
aarenas@uv.mx

Ángel J. Sánchez García[1]
angesanchez@uv.mx

[1] Facultad de Estadística e Informática,Universidad Veracruzana,Xalapa, Veracruz, México.

Resumen

Cuando no se toman en cuenta la facilidad de evolución o mantenimiento en el proceso de desarrollo de software, éste puede resultar en altos costos y tiempos prolongados de modificación, comprometiendo la calidad final y, por ende, la satisfacción del cliente. Estos dos atributos de calidad apoyan al proceso, tomando en cuenta los posibles cambios futuros no sólo en los requisitos, sino también en el entorno y la tecnología. Con el fin de conocer el estado actual de la investigación en facilidad de evolución y en facilidad de mantenimiento, se realizaron una revisión sistemática de la literatura y una síntesis temática. Se realizaron búsquedas y selección de estudios primarios en IEEEXplore, ACM, Springer y ScienceDirect, identificando un total de 37 estudios primarios. Por su parte, la síntesis temática permitió identificar los principales temas (relacionados con facilidad de evolución y mantenimiento de software) acerca de los cuales se investiga en la literatura. Los resultados muestran que la investigación es limitada. Si bien los autores encontrados en la literatura definen ambos atributos de calidad así como sus subatributos, características y algunos proponen métricas, el único artefacto considerado para lo anterior es el código. Las métricas de código más recurrentes fueron *Maintainability Index, Lines of Code, Cyclomatic Complexity y Coupling Between Objects*.

Palabras Clave: *Facilidad de evolución; Facilidad de mantenimiento; Calidad del software, Revisión sistemática; Síntesis temática.*

Abstract

When evolvability or maintainability are not considered during software development, the outcome may yield high costs and long modification hours, with an impact on the final quality and customer satisfaction. These two quality attributes contribute in considering future changes not only in requirements but also in environment and technology. A systematic literature review and a thematic synthesis were performed in order to determine the current state of evolvability and maintainability research. The search and selection processes were performed with IEEEXplore, ACM, Springer and ScienceDirect databases, obtaining a total of 37 primary studies. The thematic synthesis, in turn, allowed identifying leading themes or topics related to evolvability and maintainability. The results helped to determine that research is scarce. Although several authors provide definitions for both attributes along with subattributes, features and even metrics, the only artefact that is taken into account is code. The most frequently mentioned metrics were *Maintainability Index, Lines of Code, Cyclomatic Complexity y Coupling Between Objects*.

Keywords: *Evolvability, Maintainability, Software quality, systematics literature review, Thematic synthesis.*

1. Introducción

En la vida cotidiana, siempre se busca que, al adquirir un producto o servicio, éste sea de calidad. Una de sus numerosas definiciones es “satisfacer las necesidades desde el punto de vista cliente” (Gartnet et al, 1988), ya que se espera cumplir con las características necesarias para la satisfacción de los clientes.

En el contexto del software ocurre lo mismo, haciendo énfasis en atributos de calidad. Estos atributos se pueden trasladar a características que se espera tenga el software: rápida respuesta, eficiencia y confiabilidad, entre otros. Dos atributos que están muy relacionados son la facilidad de evolución y la facilidad de mantenimiento.

La facilidad de evolución en el software podría comprenderse como la capacidad de un producto para adaptarse a las distintas necesidades conforme avanza el tiempo, como lo son los cambios en el mercado o la organización que lo desarrolló. Este atributo, es siempre deseable al momento de desarrollar un sistema de software.

Este trabajo de investigación, también contempla la facilidad de mantenimiento de software. Se incluye este atributo para evitar dejar fuera información valiosa para el trabajo en cuestión. Este atributo, se refiere a “La facilidad con la que se puede modificar un sistema o componente de software para corregir fallas, mejorar el rendimiento u otros atributos, o adaptarse a un entorno modificado” (IEEE, 1990) Es así como se pude notar las similitudes de ambos conceptos, facilidad de evolución y facilidad de mantenimiento de software.

Con la presente investigación se busca determinar el estado actual de la facilidad de evolución y de la facilidad de mantenimiento, mediante una revisión sistemática de la literatura y una síntesis temática. Este documento se encuentra organizado de la siguiente manera: se presenta la revisión sistemática de la literatura, incluyendo preguntas de investigación, cadenas de búsqueda, criterios de inclusión así como los resultados de la misma. En la tercera sección, se describe el proceso seguido para realizar la síntesis temática, así como sus resultados. La discusión con respecto a los hallazgos se realiza en la cuarta sección, donde también se proponen algunas áreas de investigación. Finalmente, en la sección quinta, se dan las conclusiones así como el trabajo futuro.

Es importante mencionar que, dadas las limitaciones de espacio, los artefactos completos correspondientes a la revisión sistemática de la literatura y síntesis temática se encuentran en el siguiente enlace https://shorturl.at/CVPb0

2. Revisión sistemática de la literatura

Para la realización de la revisión sistemática de la literatura (RSL) se siguió el proceso establecido por (Kitchenham et al., 2015). Primero se presenta la conducción de la RSL y al final, las amenazas a la validez.

Las preguntas de investigación (RQ) y su respectiva motivación, se listan a continuación.

RQ1- ¿Cuáles son las propuestas para definir la facilidad de evolución o facilidad de mantenimiento?

Motivación: Conocer los enfoques, propuestas que a lo largo del tiempo han existido para definir la facilidad de evolución o facilidad de mantenimiento en el contexto del software.

RQ2 - ¿Cuáles son los elementos o aspectos que contribuyen a la facilidad de evolución o facilidad de mantenimiento?

Motivación: Saber que elementos intervienen o definen la facilidad de evolución o facilidad de mantenimiento.

RQ3 - ¿Cómo se mide software fácil de evolucionar o fácil de mantener?

Motivación: Saber si existen métricas para medir la facilidad de evolución o facilidad de mantenimiento en el software.

RQ4- ¿Cómo se diseña software fácil de evolucionar o fácil de mantener?

Motivación: Saber si existen estrategias y/o patrones de diseño que consideren la facilidad de evolución o facilidad de mantenimiento.

RQ5- ¿Cuáles han sido las experiencias o resultados obtenidos al considerar la facilidad de evolución o facilidad de mantenimiento en el desarrollo de software?

Motivación: Saber si ha habido experiencias utilizando algún método o estrategia para la facilidad de evolución o facilidad de mantenimiento en el desarrollo de software.

Se seleccionaron las bibliotecas digitales de IEEE, ACM, Elsevier y SpringerLink debido a que su acceso es provisto por la institución. Por otra parte, estas fuentes están consideradas por (Kuhrmann et al., 2017) como “Appropriate Data Sources” para investigación en el campo de la Ingeniería de Software. Las cadenas de búsqueda fueron las siguientes:

Utilizada en IEEE Xplore y Springer Link:

(evolvability OR "software maintainability") AND (measurement OR measure OR metric OR "software design" OR pattern OR "design pattern" OR "architectural pattern" OR method OR process OR strategy OR framework) AND NOT (neural OR robotic* OR geneti* OR mechanical OR artificial OR evolutionary)

Utilizada en ACM Digital Library:

("software evolvability" OR "software maintainability") AND (measurement OR measure OR metric OR "software design" OR pattern OR "design pattern" OR "architectural pattern" OR method OR process OR strategy OR framework) AND NOT (neural OR robotic* OR geneti* OR mechanical OR artificial OR evolutionary)

Utilizada en Science Direct (Elsevier):

("Software Evolvability" OR "Software Maintainability") AND (metric OR "design pattern") AND NOT (mechanical OR artificial OR evolutionary)

A continuación, se presentan los criterios de inclusión y exclusión:

Criterios de inclusión

CI-1. Se considerarán solamente estudios primarios en inglés.

CI-2. El título debe hacer referencia a desarrollo de software.

CI-3. El título y/o abstract debe contener el término de búsqueda “Evolvability”, “Evolution” o “Maintainability”.

CI-4. Después de leer el abstract, se muestran indicios que el estudio contesta al menos una pregunta de investigación.

Criterios de exclusión

CE-1. Estudios primarios diferentes a un congreso o journal.

CE-2. No se considerarán estudios relacionados con el área de la biología.

CE-3. No es posible obtener acceso al material completo.

CE-4. Se trata de un estudio duplicado.

Para el procedimiento de selección de estudios primarios, se aplicaron los criterios anteriores en el siguiente orden: 1) Realizar la búsqueda con ayuda de la cadena de búsqueda en dada base de datos, 2) CI-1, 3) CE-1, 4) CI-2, 5) CI-3, 6) CE-2, 7) CI-4, 8) CE-3 y 9) CE-4.

Conforme a (Kitchenham et al., 2015), se realizó *bakcward* y *forward snowballling* para cada estudio primario (EP) seleccionado. En total se seleccionaron 37 EP. No es posible mostrar la tabla completa del proceso pero se presenta un extracto en la Tabla 1. Este extracto sólo contempla IEEEXplore debido a que es la fuente en la que se encontró el mayor número de EP.

Fuente/ Paso	1	2	3	4	5	6	7	8	9	Selección
IEEEXplore	497	496	479	401	372	369	19	19	19	19
Backward Snowballing	400	400	189	173	72	71	19	17	8	8
Forward Snowballing	133	133	116	108	45	44	8	6	1	1
		...			...			...		
Total estudios seleccionados										37

Tabla 1. Resultados del proceso de selección.

La distribución de los EP, de acuerdo a las bases de datos, se puede observar en la Figura 1. el 76% de los EP se encontraron en IEEEXplore, 13% en ACM, 11% en Springer Link y 0% en Elsevier.

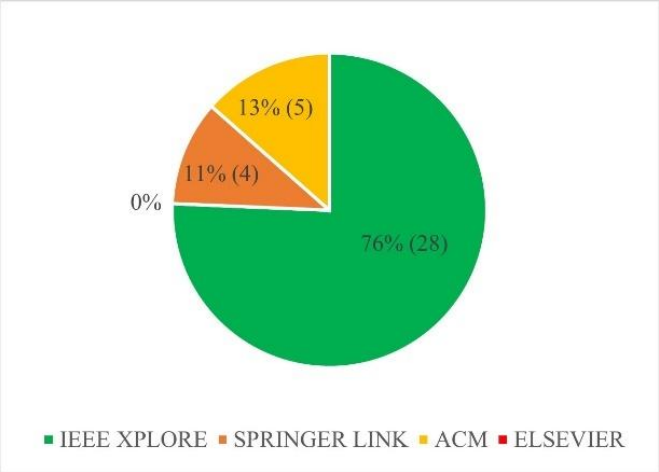

Figura. 1. Distribución de EP por bases de datos.

Al final de este documento, se encuentra un apéndice con información de cada EP y su correspondiente identificador. Con respecto al tipo de publicación, se encontró que el 86% corresponde a publicaciones en congreso mientras que sólo el 14% corresponde a journal. Se puede concluir que la mayoría de los EP no alcanzaron a probar o demostrar sus propuestas por lo que no fueron sometidos a un *journal*. La tabla completa de los EP seleccionados no se muestra aquí por cuestiones de espacio; sin embargo, puede consultarse en el enlace ya proporcionado.

A continuación, se presentan las respuestas a las preguntas de investigación.

RQ1 ¿Cuáles son las propuestas para definir la facilidad de evolución o de mantenimiento?

Los EP que proporcionan una definición de facilidad de evolución son: EP-05-IEEE, EP-06-IEEE, EP-10-IEEE y EP-12-IEEE. Los primeros dos EP, mencionan la integridad arquitectónica o la arquitectura como un elemento a considerar, mientras que los últimos dos coinciden en la necesidad de sobrevivir al cambio. Para la facilidad de mantenimiento, los EP son: EP-16-IEEE, EP-18-IEEE, EP-23-IEEE, EP-24-IEEE y EP-26-IEEE. Todas las definiciones en estos EP, hacen referencia a la facilidad con la que se puede modificar el software, ya sea para corregir fallas, hacer mejoras o adaptarse al entorno

RQ2 ¿Cuáles son los elementos o aspectos que contribuyen a la facilidad de evolución o facilidad de mantenimiento?

EP-01-IEEE, EP-05-IEEE, EP-02-IEEE, EP-04-IEE, EP-07-IEEE, EP-09-IEEE y EP-11-IEEE coinciden en presentar modelos de facilidad de evolución o mantenimiento en los que éstas se descomponen en varios niveles. En el primer nivel se encuentran las subcaracterísticas de la facilidad de evolución. Entre las subcaracterísticas se encuentran, por ejemplo para el caso de la facilidad de evolución, facilidad de análisis, facilidad de cambio, facilidad de prueba, extensibilidad, trazabilidad, adaptabilidad y portabilidad. Los siguientes niveles de modelo corresponden, en primer lugar, a subatributos. Nuevamente para el caso de la facilidad de

evolución, ene ste nivel se identifican, entre otros, facilidad de reutilización, facilidad de reemplazo, variabilidad y facilidad de configuración. El siguiente nivel corresponde a principios principalmente de diseño, tales como modularidad, separación de intereses, acoplamiento y encapsulación. Sólo el EP-01-IEEE define el nivel de métricas para algunos de estos principios. Por ejemplo *feature scattering* y *feature tangling* para el principio de separación de intereses. Todos los autores de los EP que contestan esta pregunta mencionan la importancia de cumplir con las subcaracterísticas a fin de lograr la facilidad de evolución o mantenimiento. Es importante mencionar que no todos los EP proporcionan definiciones para las subcaracterísicas o atributos. De entre todos los EP que contestan esta pregunta, se pudo determinar que, los elementos más mencionados son: facilidad de prueba, facilidad de cambio, facilidad de análisis, extensibilidad, portabilidad y estabilidad.

RQ3 ¿Cómo se mide el software fácil de evolucionar?

21 de los 37 EP proponen formas de medir. De estos, 18 incluyen métricas: EP-01-IEEE, EP-13-IEEE y EP-14-IEEE, EP-15-IEEE, EP-16-IEEE , EP-17-IEEE, EP-19-IEEE , EP-20-IEEE, EP-21-IEEE, EP-22-IEEE, EP-25-IEEE, EP-26-IEEE, EP-27-IEEE, EP-28-IEEE, EP-02-ACM, EP-03-ACM, EP-05-ACM y EP-04-SPRINGER. No todos estos EP definen las fórmulas o formas de calcular las métricas propuestas. sólo 6 EP lo hacen. Dadas las limitaciones de espacio, en la Tabla 2 sólo se listan 2 de los EP con las métricas propuestas. La tabla completa, se encuentra en el enlace previamente proporcionado.

Estudio	Métrica (s)	
EP-13-IEEE	CC - Cyclomatic Complexity	Comment Ratio
	Metrics by Halstead	Clone Coverage
	LOC - Lines of Code	Bug Patterns
	MI - Maintainability Index	Test Results
	Nesting Depth	Test Coverage
EP-16-IEEE	Maintainability Metric	

Tabla 2. Métricas propuestas por los EP.

RQ4 ¿Cómo se diseña el software fácil de evolucionar o mantener?

Con esta pregunta se intentaba encontrar estrategias, métodos, procesos o, al menos, mejores prácticas. Los EP que contestan esta pregunta se limitan a recomendar elementos que deben considerarse durante el desarrollo y, muy pocos de ellos recomiendan tomar en cuenta algunos patrones de diseño. Los EP no proporcionan evidencia de que sus recomendaciones sen correctas o efectivas. En la Tabla 3, se listan los elementos a considerar encontrados, mientras que en la Tabla 4, se listan los patrones de diseño sugeridos.

Estudio	Elementos	
EP-01-IEEE	Separación de intereses Consistencia Modularidad Exactitud Baja complejidad Completitud	Bajo acoplamiento Encapsulación Impacto del cambio Permeabilidad Tipificación
EP-03-IEEE	Extensibilidad Estabilidad de diseño Sustentabilidad Configurabilidad	
EP-02-Springer	Baja complejidad Abstracción Modularidad Cohesión Bajo acoplamiento Encapsulación Separación de intereses Herencia Simplicidad	Exactitud Consistencia Completitud Integridad conceptual Granularidad adecuada Mapeo coherente a conceptos

Tabla 3. Elementos para considerar para la Facilidad de evolución y mantenimiento.

Estudio	Patrón (es) de diseño
EP-16-IEEE	Factory Method Singleton Decorator
EP-05-ACM	Abstract Factory Decorator

Tabla 4. Patrones propuestos por los EP.

RQ5 ¿Cúáles han sido las experiencias o resultados obtenidos al considerar la facilidad de evolución o mantenimiento Enel desarrollo de software?

Sólo 3 EP contestaron RQ5. EP-04-IEEE reporta el método AREA para analizar la facilidad de evolución a nivel arquitectónico. Reportan mejoras en la organización que empleó AREA. Las mejoras se vieron reflejadas en la documentación de la arquitectura y en los modelos arquitectónicos. Todo esto ayudó a mejorar la trazabilidad de la evolución en la arquitectura de software. En EP-16-IEEE, los autores midieron la facilidad de evolución en dos soluciones de software, una con patrones de diseño y otra sin patrones de diseño. Sin embargo, argumentan que se deben seguir probando las métricas. Por último, en el EP-05-ACM, los autores incorporan patrones de diseño a un sistema de software con el fin de mejorar la facilidad de mantenimiento. Usaron *Abstract Factory* y *Decorator*, encontrando que, con el primero se obtiene código más ordenado y el cumplimiento con principios de diseño. Con *Decorator*, se incorporaron funcionalidades adicionales a un objeto, haciendo notoria la extensión de características y

comportamientos.

Con respecto a amenazas a la validez, se tienen dos. La primera, consiste en la falta de experiencia en la aplicación del método de la RSL y de la síntesis temática por parte de uno de los autores. Esto pudo ocasionar cometer errores. Sin embargo, al menos dos del resto de los autores son expertos en modelos de calidad, atributos de calidad como la facilidad de evolución, diseño y arquitecturas de software así como también en medición de software. Además, se realizaron revisiones a medida que se lograban avances, revisando cada uno de los pasos y artefactos del proceso como lo son la selección de EP, el *snowballing* y la extracción de datos.

3. Síntesis temática

La síntesis temática permite identificar, analizar y comparar hallazgos de EP de un tema en específico. A continuación, se presentan los objetivos que guiaron la síntesis temática de este trabajo, se describe el proceso seguido realizado así como los resultados.

El objetivo de esta síntesis temática es identificar, analizar y comparar los hallazgos de los estudios primarios seleccionados para este trabajo, obteniendo un modelo de orden superior de los principales temas de los mismos e identificar oportunidades de investigación. El proceso de esta síntesis es el propuesto por (Cruzes & Dyba, 2011). Este proceso se muestra en la Figura.2.

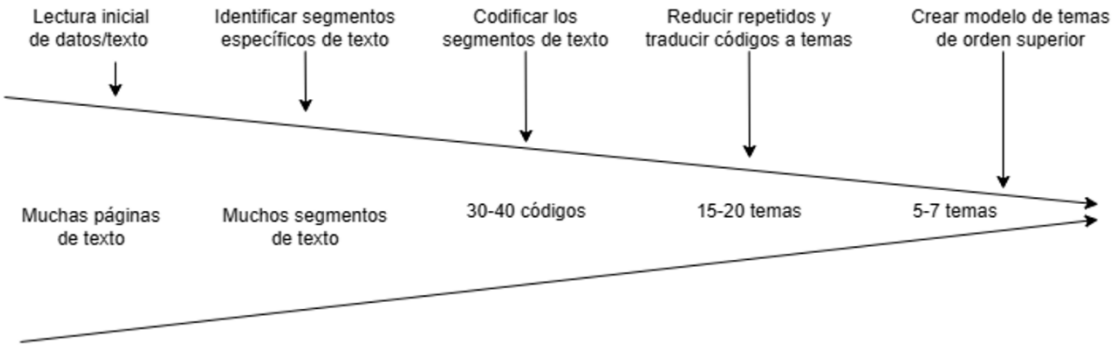

Figura 2. Pasos de la síntesis temática [8].

Una vez identificados los segmentos de texto, se llevó a cabo la codificación. De manera general, este enfoque permite tener una lista de códigos iniciales y agregar nuevos. Los códigos iniciales se obtuvieron de las preguntas de investigación y, durante el proceso de la síntesis, se agregaron nuevos códigos. A manera de ejemplo, la Tabla 5 muestra dos códigos y los segmentos de texto correspondientes. La codificación se realizó con la herramienta QualCoder (https://qualcoder.wordpress.com/). La lista completa de códigos y segmentos, se puede consultar en el enlace provisto anteriormente.

Estudio	Segmento de texto	Código
EP-09-IEEE	Software evolvability is both a business issue as well as a technical issue, since the stimuli of changes can come from both perspectives, including change of business models and business objectives, changes in environment, quality requirements, functional requirements, underlying technologies as well as emerging technologies	Estimulo de cambio
EP-03-SPRINGER	In designing for change, we recognise the need for a combination of engineering and evolutionary considerations which will provide an architecture capable of meeting current demands while also providing means of accommodating likely changes throughout its lifespan.	Arquitectura de software

Tabla 5. Ejemplos de segmentos y códigos en la síntesis temática.

A continuación, se redujeron lo códigos repetidos y se realizó la traducción de los códigos a temas. Una vez listos los temas, se elaboró el modelo de orden superior, el cual se muestra en la Figura 3.

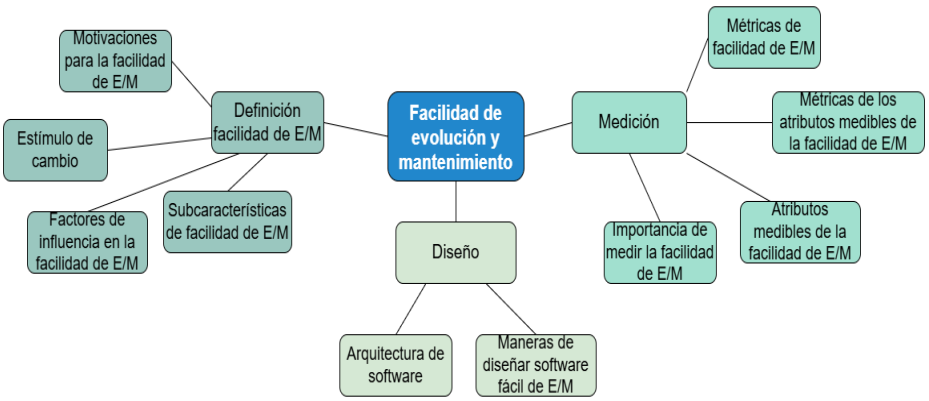

Figura 3. Modelo de orden superior.

Se puede observar que en el centro se encuentran la facilidad de evolución y la facilidad de mantenimiento que son los temas principales de este trabajo. En la Tabla 6, se describe cada uno de los temas del modelo y en la Tabla 7 se listan los códigos, con la frecuencia o número de menciones y los EP en los cuales ocurren dichas menciones.

Tema	Descripción
Definición de facilidad de evolución y facilidad de mantenimiento	Tema en el cual se mencionan las definiciones encontradas para facilidad de evolución y facilidad de mantenimiento del software. También se identifican las motivaciones y los estímulos de cambio para la facilidad de evolución y mantenimiento, así como los factores de influencia y subcaracterísticas de dichos elementos.
Diseño	Tema en el cual se identifica el papel del diseño en la facilidad de evolución y mantenimiento del software. Así como el impacto de la arquitectura en la facilidad de evolución / mantenimiento y las maneras de diseñar software fácil de evolucionar y mantener.
Medición	Tema en el cual se identifica la importancia de medir la facilidad de evolución y mantenimiento, tal como las métricas relacionadas con estos elementos. También se identifican los atributos medibles de la facilidad de evolución y mantenimiento, así como las métricas relacionadas con los atributos medibles.

Tabla 6. Temas del modelo de orden superior.

Código	Descripción	Frecuencia	Estudios	
Estímulo de cambio	Identifica los segmentos que mencionan los estímulos de cambio en el software, como pueden ser cambios en el entorno, en los requisitos, la organización, el proceso, etc.	3	EP-02-IEEE EP-09-IEEE EP-11-IEEE	
Factores de influencia en la facilidad de E/M	Identifica los factores de la facilidad de evolución / mantenimiento del software que tienen un impacto, como pueden ser la calidad del desarrollo, la integridad de la documentación, la madurez del proceso, la motivación de los miembros del equipo, entre otros.	3	EP-02-SPRINGER EP-24-IEEE EP-25-IEEE	
Importancia de medir la facilidad de E/M	Segmentos que resaltan la importancia de medir la facilidad de E/M del software.	3	EP-20-IEEE EP-26-IEEE	
Maneras de diseñar software fácil de E/M	Trata de las formas de diseñar software fácil de evolucionar / mantener.	4	EP-01-IEEE EP-16-IEEE EP-02-SPRINGER	
Métricas de los atributos medibles de la facilidad de E/M	Segmentos que mencionan métricas de software relacionadas con atributos medibles de la facilidad de evolución / mantenimiento de un sistema de software.	5	EP-01-IEEE EP-14-IEEE	
Arquitectura de software	Segmentos que mencionan la arquitectura de un sistema de software y su impacto en la facilidad de evolución / mantenimiento.	6	EP-01-IEEE EP-07- IEEE EP-09-IEEE EP-03-SPRINGER	
Motivaciones para la facilidad de E/M	Identifica las motivaciones por las cuales es importante que un sistema de software sea fácil de evolucionar / mantener.	7	EP-09-IEEE EP-11-IEEE EP-12-IEEE EP-17-IEEE EP-02-SPRINGER EP-03-SPRINGER	
Atributos medibles de la facilidad de E/M	Segmentos que mencionan atributos medibles relacionados a la facilidad de evolución / mantenimiento de un sistema de software, como pueden ser la cohesión, acoplamiento, separación de intereses, modularidad, complejidad, etc.	7	EP-07-IEEE	
Definición facilidad E/M	Identifica los segmentos que mencionan la definición de facilidad de evolución / mantenimiento del software.	15	EP-05-IEEE EP-06-IEEE EP-08-IEEE EP-10-IEEE EP-12-IEEE EP-16-IEEE EP-18-IEEE	EP-23-IEEE EP-24-IEEE EP-26-IEEE EP-27-IEEE EP-01-SPRINGER EP-02-SPRINGER EP-03-SPRINGER EP-01-ACM
Métricas de la facilidad de E/M	Segmentos que mencionan métricas relacionadas con que tan fácil es evolucionar / mantener un sistema de software.	23	EP-05-IEEE EP-13-IEEE EP-16-IEEE EP-17-IEEE EP-20-IEEE EP-25-IEEE EP-27-IEEE	EP-28-IEEE EP-04-SPRINGER EP-02-ACM EP-03-ACM
Subcaracteristicas de facilidad de E/M	Segmentos que mencionan subcaracteristicas de facilidad de evolución / mantenimiento de un sistema de software.	45	EP-01-IEEE EP-03-IEEE EP-04-IEEE EP-05-IEEE EP-07-IEEE EP-09-IEEE EP-10-IEEE EP-15-IEEE EP-02-SPRINGER EP-03-SPRINGER EP-01-ACM EP-04-ACM	EP-16-IEEE EP-19-IEEE EP-21-IEEE EP-22-IEEE EP-23-IEEE EP-24-IEEE EP-25-IEEE EP-27-IEEE EP-01-SPRINGER

Tabla 7. Códigos con su descripción y frecuencia.

4. Discusión

En esta sección se discuten los hallazgos de la RSL y de la síntesis temática, haciendo propuestas de áreas de interés o investigación.

En el análisis de estos estudios se pudo detectar que, conforme a los EP, la subcaracterística que contribuye más a la facilidad de evolución y facilidad de mantenimiento del software es la facilidad de prueba. Además de ésta, en orden de mayor a menor, se tienen: facilidad de cambio, facilidad de análisis, extensibilidad y portabilidad.

En cuanto a aspectos o factores negativos para la facilidad de mantenimiento o facilidad de evolución, sólo (Brcina & Riebisch, 2009) identifica la complejidad y el acoplamiento. Es bien sabido que principios básicos de diseño básicos como bajo acoplamiento y baja complejidad contribuyen a la modularidad del software. El software modular facilita los cambios y, por consiguiente, la facilidad de evolución y la facilidad de mantenimiento.

Con respecto a métricas, es interesante notar que las encontradas se basan únicamente en código. El poder medir la facilidad de evolución o la facilidad de mantenimiento con, por ejemplo, algún artefacto de diseño, ayudaría a la detección temprana de problemas. Una evaluación temprana del diseño ayudaría a una mejor toma de decisiones. Las métricas de código más reportadas en la literatura son: índice de facilidad de mantenimiento (MI), líneas de código (LOC), complejidad ciclomática (CC) y acoplamiento entre objetos (CBO) (Wagey et al., 2015).

Con respecto a la forma de diseñar software fácil de evolucionar o mantener, los EP sólo hacen mención del uso de patrones de diseño y de considerar algunos atributos. Es de llamar la mención que no se mencione la arquitectura de software como un aspecto importante, al menos, para la facilidad de evolución. Se sabe que la arquitectura de software define la estructura por lo que proporciona una base sólida para la facilidad de evolución.

Con respecto a las experiencias en el desarrollo de software fácil de evolucionar o mantener, no se pudieron localizar EP. Es de entender que una evaluación de este tipo, aunque sea en un contexto académico. Por otra parte, en la mayoría de las organizaciones de desarrollo de software se prioriza la entrega rápida de software, con poco tiempo para realizar un diseño robusto que sea capaz de soportar cambios.

Los resultados aquí presentados son importantes por las siguientes razones:

- Se obtiene un panorama amplio acerca de la facilidad de evolución y mantenimiento.
- Gracias a los resultados del a RSL y de la síntesis temática, es posible conocer los elementos que influyen en la facilidad de evolución y de mantenimiento del software.
- Es así, que se proponen las siguientes áreas de interés que pueden dar lugar a investigaciones en facilidad de evolución y de mantenimiento.
- Se hace necesaria la evaluación de aquellas propuestas de métodos, estrategias o enfoques para la facilidad de evolución o de mantenimiento en el desarrollo de software.
- Continuar con el estudio de subcaracterísticas, a fin de manera apoyar las decisiones correspondientes durante el desarrollo de software.
- Hacen falta las correspondientes propuestas de métricas y métodos o estrategias para artefactos distintos al código. Por ejemplo, para el diseño y evaluación de arquitecturas de software.

- Si el papel de la arquitectura de software es tan importante en la calidad final del sistema, se hace necesario estudiar las maneras de incluir, modelar y evaluar la facilidad de evolución o de mantenimiento en la misma.

- Continuar con la investigación del uso de patrones de diseño que contribuyan a alcanzar la facilidad de evolución o de mantenimiento del software.

5. Conclusiones y trabajo futuro

La motivación viene por los tipos de diagnósticos que se pueden realizan a los pacientes, ya que en la actualidad existen muchas variaciones y cada uno tiene diferentes datos a analizar. Para poder categorizar de mejor manera los datos tomaremos como foco principal los 2 diagnósticos más comunes que recolectan datos por medio de sensores. Por la misma razón, los tipos de datos se han clasificado en las siguientes categorías: Cerebral y de Movimiento

Se han analizaron la facilidad de evolución y facilidad de mantenimiento del software, mediante una revisión sistemática de la literatura y una síntesis temática. Se presentó el proceso de la RSL y de la síntesis temática conforme a (Cruzes & Dyba, 2011) (Kitchenham et al., 2015), respectivamente. Fue posible identificar los elementos contribuyentes a la facilidad de evolución y facilidad de mantenimiento, así como un conjunto de métricas para su evaluación. No se encontraron métodos, estrategias, procesos ni siquiera mejores prácticas a realizar para satisfacer estos atributos. Las propuestas encontradas tienen aún pendiente la validación correspondiente. En lo que se refiere a las métricas, sólo existen las que miden código. Otros artefactos no han sido considerados hasta ahora.

Con respecto a las limitantes del trabajo, éstas se refieren, en primer lugar, al desconocimiento del proceso de RSL y síntesis temática y de los temas de facilidad de evolución y mantenimiento por parte de uno de los autores, quien desarrollaba su trabajo recepcional. Sin embargo, como ya se ha explicado, el resto de los autores cuenta con el conocimiento y habilidades necesarias para solventar esta situación. Otra limitante que debe mencionarse es el acceso a las fuentes o base de datos provistas por la institución. Sin embargo, fue posible conseguir los EP con acceso restringido.

A partir de los resultados, se identifican los siguientes trabajos futuros.

- Investigar y evaluar las métricas encontradas para evaluar la facilidad de evolución y de mantenimiento de software.

- Proponer, basado en los hallazgos ya presentados, alguna métrica de facilidad de evolución o mantenimiento para arquitecturas de software, considerando también un caso de estudio.

- Investigar el uso de patrones de diseño que contribuyan a la facilidad de evolución o de mantenimiento del software.

Referencias

Brcina, R., Bode, S., & Riebisch, M. (2009). Optimisation process for maintaining evolvability during software evolution. Proceedings of the 16th Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (pp. 196–205). https://doi.org/10.1109/ECBS.2009.20

Cruzes, D. S., & Dybå, T. (2011). Recommended steps for thematic synthesis in software engineering. Proceedings of the International Symposium on Empirical Software Engineering and Measurement (pp. 275–284). https://doi.org/10.1109/ESEM.2011.36

Gartner, W., Deming, W., Naughton, M., & Gitlow, H. S. (1988). The Deming theory of management. The Academy of Management Review, 13, 138. https://doi.org/10.2307/2583562

IEEE. (1990). IEEE standard glossary of software engineering terminology (IEEE Std 610.12-1990). https://doi.org/10.1109/IEEESTD.1990.101064

Kitchenham, B. A., Budgen, D., & Brereton, P. (2015). Evidence-based software engineering and systematic reviews. CRC Press.

Kuhrmann, M., Fernández, D. M., & Daneva, M. (2017). On the pragmatic design of literature studies in software engineering: An experience-based guideline. Empirical Software Engineering, 22, 2852–2891. https://doi.org/10.1007/s10664-016-9492-y

Wagey, B. C., Hendradjaya, B., & Mardiyanto, M. S. (2015). A proposal of software maintainability model using code smell measurement. Proceedings of the 2015 International Conference on Data and Software Engineering (pp. 25–30). https://doi.org/10.1109/ICODSE.2015.7436966

APÉNDICE. LISTA DE ESTUDIOS PRIMARIOS SELECCIONADOS

#	ID	Título	Autores	Año	Tipo de publicación
1	EP-01-IEEE	Optimisation Process for Maintaining Evolvability during Software Evolution	Robert Brcina, Stephan Bode, Matthias Riebisch	2009	Congreso
2	EP-02-IEEE	Analysis of Software Evolvability in Quality Models	H. P. Breivold; I. Crnkovic	2009	Congreso
3	EP-03-IEEE	A framework for assessing the evolvability characteristics along with sub-characteristics in AOSQ model using fuzzy logic tool	P. Kumar; S. K. Singh	2017	Congreso
4	EP-04-IEEE	Analyzing Software Evolvability of an Industrial Automation Control System: A Case Study	H. P. Breivold; I. Crnkovic; R. Land; M. Larsson	2008	Congreso

5	EP-05-IEEE	Analyzing Software Evolvability	H. P. Breivold; I. Crnkovic; P. J. Eriksson	2008	Congreso
6	EP-06-IEEE	Optimum technology insertion into systems based on the assessment of viability	P. A. Sandborn; T. E. Herald; J. Houston; P. Singh	2003	Journal
7	EP-07-IEEE	Using dependency model to support software architecture evolution	H. P. Breivold; I. Crnkovic; R. Land; S. Larsson	2008	Congreso
8	EP-08-IEEE	Towards a practical maintainability quality model for service-and-microservice based systems	Justus Bogner, Stefan Wagner, Alfred Zimmermann	2017	Congreso
9	EP-09-IEEE	Evaluating software evolvability	H. P. Breivold, I. Crnkovic, and P. Eriksson	2007	Congreso
10	EP-10-IEEE	Dynamic and static views of software evolution	S. Cook, H. Ji, and R. Harrison	2001	Congreso
11	EP-11-IEEE	Using Software Evolvability Model for Evolvability Analysis	Breivold, H.P., and Crnkovic, I	2008	Congreso
12	EP-12-IEEE	Evolvability as a quality attribute of software architectures	S. Ciraci and P. van den Broek	2006	Congreso
13	EP-13-IEEE	On Automatically Collectable Metrics for Software Maintainability Evaluation	J. -P. Ostberg; S. Wagner	2014	Congreso
14	EP-14-IEEE	Analyzing the Effect of Design Patterns on Software Maintainability: A Case Study	S. Rochimah; P. G. Nuswantara; R. J. Akbar	2018	Congreso
15	EP-15-IEEE	A Quantitative Approach to Software Maintainability Prediction	L. Ping	2010	Congreso
16	EP-16-IEEE	Evaluating the Impact of Design Patterns on Software Maintainability: An Empirical Evaluation	H. K. Jun; M. E. Rana	2021	Congreso

17	EP-17-IEEE	An integrated measure of software maintainability	K. K. Aggarwal; Y. Singh; J. K. Chhabra	2002	Congreso
18	EP-18-IEEE	Software maintainability-a new 'ility'	D. A. Sunday	1989	Congreso
19	EP-19-IEEE	A proposal of software maintainability model using code smell measurement	B. C. Wagey; B. Hendradjaya; M. S. Mardiyanto	2015	Congreso
20	EP-20-IEEE	Why Is It Important to Measure Maintainability and What Are the Best Ways to Do It?	C. Chen; R. Alfayez; K. Srisopha; B. Boehm; L. Shi	2017	Congreso
21	EP-21-IEEE	A Controlled Experiment for Evaluating the Impact of Coupling on the Maintainability of Service-Oriented Software	M. Perepletchikov; C. Ryan	2011	Journal
22	EP-22-IEEE	Evaluating maintainability of android applications	A. A. Saifan; A. Al-Rabadi	2017	Congreso
23	EP-23-IEEE	Evaluating Software Maintainability Using Fuzzy Entropy Theory	L. Cai; Z. Liu; J. Zhang; W. Tong; G. Yang	2010	Congreso
24	EP-24-IEEE	Research on maintainability evaluation of service-oriented software	Ma Zhe; Ben Kerong	2010	Congreso
25	EP-25-IEEE	A Practical Model for Measuring Maintainability	I. Heitlager, T. Kuipers, and J. Visser	2007	Congreso
26	EP-26-IEEE	A maintainability estimation model and tool	A. Hincheeranan and W. Rivepiboon	2012	Journal
27	EP-27-IEEE	A comparative Study of Maintainability Index of Open Source Software	Ganpati, Anita, A. Kalia, and H. Singh.	2012	Journal
28	EP-28-IEEE	Measuring Maintainability Index of a Software Depending on Line of Code Only	Najm, N.	2014	Journal

29	EP-01-SPRINGER	Evolvability Characterization in the Context of SOA	Jose L. Arciniegas H. & Juan C. Dueñas L.	2010	Congreso
30	EP-02-SPRINGER	Impact Evaluation for Quality-Oriented Architectural Decisions regarding Evolvability	Stephan Bode & Matthias Riebisch	2010	Congreso
31	EP-03-SPRINGER	Defining systems architecture evolvability - a taxonomy of change	Rowe, D., Leaney, J., Lowe, D.	1998	Congreso
32	EP-04-SPRINGER	A Bayesian Belief Network for Modeling Open Source Software Maintenance Productivity	Stamatia Bibi, Apostolos Ampatzoglou, Ioannis Stamelos	2016	Congreso
33	EP-01-ACM	A Generic Method for Identifying Maintainability Requirements Using ISO Standards	Khalid T. Al-Sarayreh, Asma Labadi & Kenza Meridji	2015	Congreso
34	EP-02-ACM	Questioning software maintenance metrics: a comparative case study	Dag I.K. Sjoberg, Bente Anda & Audris Mockus	2012	Congreso
35	EP-03-ACM	Design property metrics to maintainability estimation: a virtual method using functional relationship mapping	T.R. Gopalakrishnan Nair, Sri Aravindh & R.Selvarani	2010	Congreso
36	EP-04-ACM	Demystifying maintainability	M. Broy, F. Deissenboeck, and M. Pizka	2007	Congreso
37	EP-05-ACM	Impact of Abstract Factory and Decorator Design Patterns on Software Maintainability: Empirical Evaluation using CK Metrics	Kurmangali A, Rana M and Ab Rahman W.	2022	Congreso

Tabla 8. Estudios primarios seleccionados.

NOTAS BIOGRÁFICAS

Mario Eduardo Dorantes Hernández es un egresado de la Facultad de Estadística e Informática, llevando la licenciatura en ingeniería de software. Realizó una monografía relacionada con la Facilidad de Evolución y de Mantenimiento de Software. A lo largo de su formación académica, ha desarrollado un interés particular en la facilidad de evolución y mantenimiento del software, enfocándose en las definiciones de estas características, así como en las diversas formas de diseñarlos y medirlos.

María Karen Cortés Verdín Licenciado en Informática por la Universidad Veracruzana, Maestra en Ciencias en Ingeniería de Sistemas de Información UMIST en Reino Unido, Maestra en Ingeniería de Software y Doctora en Ciencias de la Computación por el CIMAT, A.C. Profesor de tiempo completo en la Facultad de Estadística e Informática de la Universidad Veracruzana en la Licenciatura en Ingeniería de Software. Integrante del Cuerpo académico Ingeniería y Tecnología de Software. Candidata al SNII. Perfil PRODEP deseable desde 2014. Su trabajo incluye calidad de software, líneas de productos de software y arquitecturas de software principalmente.

María de los Ángeles Arenas Valdés es Licenciada en Informática por la Universidad Veracruzana, con Maestría en Ciencias de la Computación por la Fundación Arturo Rosenblueth, Profesor de Tiempo Completo en la Licenciatura en Ingeniería de Software de la Universidad Veracruz, cuenta con reconocimiento del programa Profesional Docente PRODEP

Ángel Juan Sánchez García es Licenciado en Informática, Maestro en Inteligencia Artificial, Especialista en Métodos Estadísticos y Doctor en Inteligencia Artificial por la Universidad Veracruzana, México. Actualmente es profesor en la Facultad de Estadística e Informática de la Universidad Veracruzana. Es responsable del Cuerpo académico Ingeniería y Tecnología de Software. Es miembro desde 2018 del Sistema Nacional de Investigadoras e Investigadores del CONAHCYT (actualmente Nivel 1, área 8) y cuenta con el reconocimiento del Programa de Desarrollo Profesional Docente (PRODEP) desde 2018. Su trabajo de investigación incluye las áreas de Aprendizaje automático e Inteligencia Artificial aplicada a la Ingeniería de software