UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Tecnura

https://revistas.udistrital.edu.co/index.php/Tecnura/issue/view/1326

DOI: https://doi.org/10.14483/22487638.24863

REVISIÓN

Automatización de la generación de casos de prueba de software 2018–2024: mapeo sistemático en bases de datos IEEE, ACM, Springer y Elsevier

Software test case generation automation 2018-2024: Systematic Mapping in IEEE/ACM/Springer/Elsevier

Daniel García-Arcos [1], Juan Carlos Pérez-Arriaga [2] y Karen Cortés-Verdín [3]

Fecha de Recepción: 12 de noviembre de 2025 **Fecha de Aceptación**: 26 de febrero de 2026

Cómo citar: D. García-Arcos, J. C. Pérez-Arriaga, y K. Cortés-Verdín, «Automatización de la generación de casos de prueba de software 2018–2024: mapeo sistemático en bases de datos IEEE, ACM, Springer y Elsevier», *Tecnura*, vol. 30, n.° 87, mar. 2026. 1–19. https://doi.org/10.14483/22487638.24863

Resumen

Contexto: la prueba de software es esencial para asegurar la calidad de productos de software. Sin embargo, la generación manual de casos de prueba es costosa, propensa a errores y consume tiempo considerable, lo que ha impulsado el desarrollo de métodos y herramientas para automatizar esta actividad.

Objetivo: analizar métodos y herramientas para la automatización de la generación de casos de prueba documentados entre 2018 y 2024, identificando características, retos en el empleo práctico y formas de evaluación.

Método: se aplicó la metodología de mapeo sistemático de Kitchenham et al., analizando 70 estudios primarios de *IEEE Xplore*, *ACM Digital Library*, *Science Direct* y *Springer Link*. Se empleó el enfoque *quasi-gold standard* para definir cadenas de búsqueda y síntesis narrativa de Popay et al. para el análisis de datos.

Resultados: la prueba basada en búsqueda y la prueba basada en modelos son las técnicas predominantes, con una emergente adopción de Modelos de Lenguaje Grande (LLM, *Large Language Models*). Los métodos y herramientas se orientan principalmente ha-cia APIs RESTful y aplicaciones Java, presentando limitaciones en el manejo de especificaciones y generación de *test oracles* efectivos.

Conclusiones: se identifican oportunidades significativas en pruebas de seguridad y rendimiento. Aunque el estudio no incluyó evaluación de calidad de estudios primarios, literatura gris ni *snowballing*, proporciona un panorama base para futuras investigaciones en generación automática de casos de prueba.

Palabras clave: prueba de software, prueba automática, automatización de pruebas, generación de pruebas, calidad de software.

1 Licenciado en Ingeniería de Software por la Universidad Veracruzana. Ingeniero de Automatización de Asegura-miento de la Calidad en C3 AI. ROR. Email: gs21013881@egresados.uv.mx

2 Maestro en Ciencias de la Computación y Profesor de Tiempo Completo en la Facultad de Estadística e Informática de la Universidad Veracruzana. ROR. Email: juaperez@uv.mx

3 Maestra en Ingeniería en Sistemas de Información (UMIST), Maestra en Ingeniería de Software (CIMAT, A. C.), Doctorado en Ciencias con Orientación en Ciencias de la Computación (CIMAT, A. C.). Hasta 2025 profesor de tiempo completo en la Universidad Veracruzana, Licenciatura en Ingeniería de Software, Xalapa, Veracruz, México. ROR. Email: karencortesv@gmail.com

Abstract

Context: Software testing is essential to ensure software product quality. However, manual test case generation is costly, error-prone, and time-consuming, driving the development of methods and tools to automate this activity.
Objective: To analyze methods and tools for automating test case generation reported between 2018 and 2024, identifying characteristics, practical implementation challenges, and evaluation approaches**.**
Methodology: Kitchenham et al.'s systematic mapping methodology was applied, analyzing 70 primary studies from IEEE Xplore, ACM Digital Library, ScienceDirect, and SpringerLink. The *quasi-gold standard* approach was used to define search strings, and Popay et al.'s narrative synthesis for data analysis.
Results: Search-based testing and model-based testing are the predominant techniques, with emerging adoption of Large Language Models (LLMs). Methods and tools mainly target RESTful APIs and Java applications, showing limitations in specification handling and effective test oracle generation.
Conclusions: Significant opportunities are identified in security and performance testing. Although the study did not include quality assessment of primary studies, grey literature, or *snowballing*, it provides a baseline for future research in automated test case generation.
Keywords: software testing; test automation; test generation; software quality.

Introducción

La prueba de software es una actividad esencial en el desarrollo de software, siendo el método más importante de aseguramiento de la calidad de este [1]. Tradicionalmente la prueba ha sido realizada de forma manual; sin embargo, ello ha implicado retos significativos para la industria, por lo cual la automatización de la prueba de software ha sido adoptada progresivamente [2]. El proceso de prueba está constituido por diferentes actividades: la planeación de la prueba, diseño e implementación de pruebas, ejecución de pruebas, el reporte de problemas encontrados durante la prueba y el seguimiento de defectos [3].

Específicamente, la generación de los casos de prueba implica la creación de conjuntos de pruebas útiles para evaluar el sistema bajo prueba [SUT, del inglés *System Under Test*]. Esta actividad ha sido descrita como uno de los problemas clave en la prueba de software [3] dado que demanda un esfuerzo considerable y tiene un impacto directo en la calidad del producto. La creación manual de los casos de prueba es un proceso lento y susceptible a errores, ya que pueden omitirse ciertas entradas o combinaciones de estas que conduzcan a comportamientos inesperados y pueden generarse casos de prueba redundantes o poco realistas., Además, la creación de casos de prueba que satisfagan los requerimientos de prueba no es sencilla, debido a la complejidad y diversidad de parámetros de entrada de las aplicaciones del mundo real [4]. Por ello, desde hace más de 20 años se han investigado métodos para la automatización de la generación de casos de prueba [5].

Ante este escenario, han surgido diversos enfoques para automatizar la generación de casos de prueba, empleando técnicas tales como algoritmos de búsqueda, modelos, especificaciones y, recientemente, inteligencia artificial generativa. Dado que se trata de un área relevante y en continuo desarrollo, resulta necesario realizar un mapeo sistemático de la literatura que permita identificar, categorizar y analizar la producción científica existente sobre la automatización de la generación de casos de prueba, con el fin de proporcionar un panorama de los métodos y herramientas reportados en años recientes.

El propósito original de este estudio fue identificar y sistematizar métodos y herramientas para la generación automática de casos de prueba, con miras a proponer procedimientos de automatización basados en los enfoques más eficientes y efectivos documentados en la literatura; por ello se registraron, cuando estaban disponibles, los métodos de evaluación y las métricas asociadas a cada propuesta. No obstante, el alcance del estudio fue ajustado durante su desarrollo, lo que limitó la posibilidad de realizar el análisis de la literatura gris y el *snowballing* de los estudios de literatura blanca. En consecuencia, los resultados presentados tienen carácter exploratorio y constituyen una base para futuras investigaciones en esta área, por lo que deben interpretarse con cautela.

Diversos estudios previos han abordado la generación automática de casos de prueba desde diferentes perspectivas. Verma et al. [6] realizaron una revisión sistemática de la literatura de las herramientas y técnicas para la generación de casos de prueba. En su estudio identifican 5 técnicas, presentando ventajas y desventajas de cada una. Aunque listan los conjuntos de datos utilizados para generar casos de prueba, no profundizan en cómo fueron usados o cuáles fueron los resultados tras usar estos conjuntos. También categorizan herramientas para la generación de casos de prueba, clasificándolas en cinco categorías: *open source*, académicas, comerciales y combinaciones de estas. El estudio brinda un panorama de las herramientas usadas para la generación de casos de prueba, pero se limita al periodo 2011-2021 y no profundiza en la efectividad que cada una de estas presentó o los retos que podrían surgir al emplearlas. Tampoco brinda directrices o un análisis que pueda conducir a la selección de alguna de estas.

Potuzak y Lipka [7] categorizaron los métodos según la tecnología principal utilizada para la generación de casos de prueba: generación pseudoaleatoria, métodos basados en flujos de control, basados en especificaciones, basados en el análisis de ejecución, basados en la descripción de datos, en búsqueda y aprendizaje automático. El estudio presenta las herramientas de los métodos que indicaron tener alguna implementación, y obtuvieron 15 herramientas. Además, identifican los niveles de prueba hacia los que se enfocan los estudios primarios y la plataforma objetivo de cada método (Java, C/C++, Web, etc.). Si bien los resultados brindan una visión de las tendencias en la automatización de la generación de casos de prueba entre los años 2000 y 2022, los autores indican que no es una revisión sistemática de la literatura, sino un resultado intermedio de un trabajo exploratorio para llegar a ella.

Baqar y Khanda [8] examinan el rol de la inteligencia artificial [IA] en la generación y validación de casos de prueba y destacan técnicas de *machine learning* [ML], procesamiento de lenguaje natural [NLP del inglés ***natural language processing***] y modelos de lenguaje [LLM del inglés *Large Language Models*] para la automatización, creación y análisis de código, requisitos y datos históricos. Identifican herramientas como Test.ai, Functionize y Mabl, así como técnicas de auto reparación y optimización automática. El trabajo exploratorio resalta las limitaciones de la IA en la generación de casos de prueba, por ejemplo, la confiabilidad de la IA, el entrenamiento constante y cómo la efectividad depende directamente de la calidad de los datos de entrenamiento. Los autores resaltan sus preocupaciones respecto a privacidad y seguridad, así como los costos de inversión y recursos. Aunque el trabajo presenta un panorama amplio respecto a el papel de la IA en el futuro de la generación de casos de prueba de software, no indican cuál fue el método seguido o si se siguió un proceso sistemático.

Más recientemente, Navarro e Ibarra [9] realizaron un mapeo sistemático de la literatura sobre generación de casos de prueba, específicamente usando NLP. En el estudio revisan estudios publicados entre 2004 y 2024, e identifican las técnicas más utilizadas [*POS tagging*, *dependency parsing*, tokenización], herramientas principales [Stanford Core NLP, NLTK, spaCy] y modelos como GPT-3.5 Turbo y BERT, con lo cual reportan que la mayoría logra un nivel medio de automatización. El estudio profundiza en la entrada para los métodos, así como en el formato de los casos generados por las soluciones propuestas y el tipo de prueba hacia el que se orientan los casos de prueba; además identifica las métricas asociadas a los enfoques reportados, el desempeño que tienen cuando se usan y la cobertura de prueba. Este mapeo es el más parecido al presentado en este trabajo, proporcionando un análisis detallado respecto a NLP en la generación de casos de prueba.

Nuestra investigación complementa y extiende estos trabajos en varios aspectos. A diferencia de Verma et. al [6], que se limita al periodo 2011-2021, nuestro estudio abarca hasta 2024 e incorpora un análisis sistemático de las formas de evaluación, métricas empleadas y desafíos reportados. Asimismo, nuestro mapeo sigue la metodología de Kitchenham et. al. [10], lo cual garantiza un proceso de selección y extracción sistemático. Finalmente, si bien Navarro e Ibarra [9] se centran exclusivamente en técnicas basadas en NLP, nuestro trabajo ofrece un panorama más amplio al abarcar múltiples técnicas [SBST, MBT, LLMs, entre otras] y analizar, además, sus formas de evaluación, retos prácticos de implementación y características específicas tanto de métodos como de herramientas.

El documento se organiza de la siguiente forma: la sección 2 presenta el trabajo relacionado, la sección 3 detalla el método seguido para la realización de este mapeo sistemático. En la sección 4 se presentan los resultados encontrados, listando métodos y herramientas para la tarea de generación de casos de prueba. La sección 5 discute los hallazgos y sus implicaciones. Finalmente, la sección 6 presenta las conclusiones y trabajo futuro.

Es importante mencionar que, debido a limitaciones de espacio, los artefactos completos del mapeo sistemático se encuentran en el siguiente enlace: https://acortar.link/Ht3jG8.

Metodología

Para realizar el mapeo sistemático de la literatura se empleó la metodología descrita por Kitchehham et al. [10], que consiste en tres fases clave: (1) Planeación, (2) Conducción, y (3) Reporte de los resultados.

Planeación

En esta etapa, se definió una estrategia de búsqueda para guiar el mapeo, con el planteamiento de preguntas de investigación, fuentes seleccionadas y cadenas de búsqueda.

Preguntas de investigación (PI). Puesto que el objetivo es identificar, categorizar y analizar literatura sobre herramientas y métodos para la generación automática de casos de prueba, este estudio busca alcanzarlo respondiendo las siguientes preguntas de investigación:

PI1. ¿Qué métodos se emplean para la automatización de la generación de casos de prueba de software?

PI1.1. ¿Qué características tienen los métodos?

PI1.2. ¿Cuáles han sido los retos en el empleo de los métodos?

PI1.3. ¿Cómo se evalúa la efectividad de los métodos?

PI2. ¿Qué herramientas se reportan para la automatización de la generación de casos de prueba de software?

PI2.1. ¿Qué características tienen las herramientas?

PI2.2. ¿Cuáles han sido los retos en el empleo de las herramientas?

PI2.3. ¿Cómo se evalúa la efectividad de las herramientas?

Tabla 1. *Fuentes seleccionadas*

Fuente	Sitio web
IEEE Digital Library (IEEE Xplore)	https://ieeexplore.ieee.org/
ACM Digital Library	https://dl.acm.org/
Springer Link	https://link.springer.com/
Elsevier (Science Direct)	https://www.sciencedirect.com/

Fuente: elaboración propia.

Tabla 2. *Términos de búsqueda.*

Término	Términos relacionados
Method	Approach
Automation	Automatic, Automatically
Test case generation	Test generation
Software testing	-

Fuente: elaboración propia.

Selección de fuentes.

En la Tabla 1 se presentan las fuentes seleccionadas. La elección se fundamenta en el amplio uso por parte de revisores en el área de Ingeniería de Software. Además, IEEE y ACM abarcan en conjunto de publicaciones de gran relevancia en esta disciplina [6], lo que provee una base sólida para el hallazgo de literatura relevante.

Cadena de búsqueda.

Se definieron cadenas de búsqueda usando el enfoque quasi-gold standard (QGS) propuesto por Zhang et al. [11], que evalúa la sensibilidad y precisión. Los términos utilizados se muestran en la Tabla 2. Las cadenas seleccionadas se muestran en la Tabla 3, estas mismas pueden consultarse junto con los valores obtenidos en el enlace previamente proporcionado.

Tabla 3. *Cadenas de búsqueda (S=sensibilidad, P=precisión)*

Motor	Cadena	S	P
IEEE Xplore	"software" AND "testing" AND (automati* AND ("test case generation" OR "test generation")) AND ("approach" OR "method" OR "tool") AND NOT (autonomous OR vehic* OR safety-critical OR "internet of things" OR transportation OR embedded)	91.6 %	2.4 %
ACM Digital Library	"software testing" AND (("automatic" OR "automatically") AND ("test case generation" OR "test generation")) AND (("approach" OR "method") OR "tool") AND NOT (autonomous* OR vehic* OR safety-critical OR quantum OR cyber-physical OR automotive OR "control system" OR circuits OR "software product line")	80 %	1.4 %
Science Direct	"software testing" AND (("automatic" OR "automatically") AND ("test case generation" OR "test generation")) AND NOT (safety-critical OR "control system" OR cyber-physical)	100 %	3.8 %
Springer Link	"software" AND "testing" AND (automati* AND ("test case generation" OR "test generation")) AND NOT (autonomous OR vehic* OR safety-critical OR "internet of things" OR transportation)	80 %	1.6 %

Fuente: elaboración propia.

Tabla 4. *Criterios de inclusión y exclusión.*

Criterio	Descripción del criterio de inclusión (CI) / criterio de exclusión (CE)
CI1	El estudio fue publicado entre los años 2018 y 2024.
CI2	El estudio está publicado en un *Journal*, Congreso o *Workshop*.
CI3	El estudio se encuentra escrito en inglés.
CI4	El título o *abstract* del estudio sugiere que este responde al menos una **PI**.
CI5	El estudio responde al menos una pregunta de investigación.
CE1	No se tiene acceso al estudio.
CE2	El estudio es duplicado.
CE3	El artículo no es un estudio primario.

Fuente: elaboración propia.

Criterios de selección.

Los criterios aplicados se muestran en la Tabla 4. Se seleccionaron aquellos estudios publicados entre 2018 y 2024 en un congreso, *journal* o workshop. La selección se restringe a estudios en idioma inglés. Por otra parte, se excluyen aquellos a los que no se tiene acceso, son duplicados y no son estudios primarios.

Tabla 5. *Proceso de selección de estudios primarios*

Base de datos	Etapa inicial	Etapa 1 (CI1, CI2)	Etapa 2 (CI3, CE1)	Etapa 3 (CI4, CE2, CE3)	Etapa 4 (CI5)
IEEE Xplore	1529	455	454	100	39
ACM Digital Library	1577	539	538	50	14
Science Direct	551	154	154	28	10
Springer Link	3450	238	111	19	7
Total	7107	1386	1257	197	70

Fuente: elaboración propia.

Proceso de selección de estudios primarios.

Se aplicaron los criterios de selección en cinco etapas secuenciales, primero con filtros automáticos y al final con lectura completa de textos. Las búsquedas se realizaron en noviembre de 2024 para *IEEE Xplore* y *ACM Digital Library*, y en febrero de 2025 para *Springer Link* y *Science Direct*. La Etapa 0 (búsqueda inicial) recuperó 7107 estudios de las cuatro bases de datos mediante la aplicación de las cadenas de búsqueda definidas. La Etapa 1 redujo significativamente este conjunto a 1386 estudios mediante la aplicación de filtros automáticos disponibles en todas las base de datos, específicamente: (1) filtros de periodo temporal para restringir la búsqueda a publicaciones entre 2018 y 2024 (criterio CI1), y (2) filtros de tipo de publicación para limitar los resultados a artículos de congresos, *journals* y *workshops*

(criterio CI2) (*conference* y *journal* en *IEEE Xplore*, *research article* en *ACM Digital Library*, Elsevier (*Science Direct*) y *Springer Link*). La Etapa 2 aplicó los criterios CI3 y CE1 relacionados al idioma del artículo (solo en inglés) y acceso (aceptando únicamente a los cuales se tuviera acceso). La Etapa 3 correspondió a los criterios CI4 (aquellos estudios en los que el título o *abstract* sugería responder al menos una PI), CE2 (estudios duplicados) y CE3 (artículos que no eran estudios primarios). Finalmente, la Etapa 4 consistió en una lectura completa para incluir aquellos que respondieran al menos una PI (CI5). La Tabla 5 detalla la evolución cuantitativa del proceso por base de datos y etapa, con un resultado de 70 estudios primarios finales.

En el proceso de selección de estudios primarios participaron tres investigadores, uno de ellos el principal. Este investigador principal se encargó de la elaboración de las cadenas de búsquedas y de la selección inicial de los estudios primarios (EP). Estos pasos fueron revisados con detalle por los otros dos investigadores. Se realizaron reuniones semanales para todo el proceso, en las cuales se revisaron los desacuerdos, cuando los hubo. Estos siempre se analizaron antes de tomar una decisión y, en caso de no alcanzarse un acuerdo se optó por realizar una votación.

Extracción y síntesis de datos.

La extracción se realizó mediante un formato estructurado acordado en las reuniones del equipo de investigadores. El formato de extracción incluyó los siguientes campos: ID, para identificar el EP; Título; Auto o autores; Año; Fuente, correspondiente al nombre del congre-so, *journal* o *workshop* en el que fue presentado; Tipo de publicación (congreso, *journal* o *workshop*); DOI; Palabras clave; y Resumen. Respecto a las preguntas el formato incluyó campos para dar respuesta a estas, tales como un campo para identificar el método o herramienta expuesto en el EP, entrada recibida, salida en cuanto al formato de los casos de prueba que son generados, técnica de generación de casos de prueba utilizada, nivel de prueba de los casos de prueba generados, retos en el empleo, la manera en que fue evaluado el método o herramienta, resultado y métricas utilizadas para la evaluación. El formato también puede ser encontrado en los artefactos a los que se puede acceder mediante el enlace proporcionado, igual que la extracción realizada en cada una de las bases de datos seleccionadas.

Para la síntesis se adoptó el enfoque narrativo de Popay et al. [12], adaptándolo con los siguientes pasos: (1) desarrollo de teoría sobre cómo, por qué y para quién es la síntesis; (2) síntesis preliminar y (3) exploración de relaciones entre datos. Con respecto al paso 1), para los estudiantes de la Licenciatura en Ingeniería de Software de la Facultad de Estadística e Informática de la Universidad Veracruzana, la automatización de pruebas presenta varios retos de diversos grados de dificultad. Se aprecia que la generación automática de casos de prueba es la actividad con mayor dificultad. Como se comentó en la Introducción, el objetivo fue identificar métodos y herramientas para la automatización de generación de casos de prueba. Por lo tanto, en el análisis de los métodos y herramientas encontrados, se consideran aspectos como: efectividad, indicadores o medidas de evaluación empleados y retos principalmente.

En cuanto al paso 2) "Desarrollo de una síntesis preliminar", se utilizaron las técnicas de descripciones textuales de los estudios (que pueden ser consultadas en las tablas incluidas en primer enlace provisto de los artefactos). En dichas descripciones se incluye la propuesta del estudio (método o herramienta) y los pasos generales que siguen para generar los casos de prueba, describiendo la entrada, salida, técnica para generar los casos de prueba, entre otros puntos. También se realizó tabulación y conteo de votos, como parte de las técnicas propuestas en [12]. Finalmente, el paso 3) "Exploración de relaciones entre datos" se realizaron gráficas y tabulación, estas técnicas permitieron identificar aspectos comunes entre los estudios primarios que respondieran cada una de las PI.

Conducción

En total se seleccionaron 70 estudios. En los artefactos que se pueden consultar en el enlace mencionado se incluye una adaptación del diagrama ROSES [13] con los resultados del proceso de selección, así como la información completa y detallada de la selección por base de datos.

En el Anexo A se muestra la información de cada estudio primario seleccionado.

Resultados

A continuación, se describen los resultados cuantitativos de la selección de estudios llevada a cabo, seguidos de los hallazgos para cada pregunta de investigación en las secciones 3.2 y 3.3. La mayoría de los estudios fueron publicados entre 2021 y 2024 (48 de 70, 68.5 %); se desta-ca 2021 con 14 publicaciones. Los años 2019 y 2020 muestran una baja producción (4 y 6 estudios, respectivamente). En cuanto al tipo de publicación, predominan los congresos (66 %), seguidos de *journals* (31 %) y workshops (3 %) (Figura **2**).

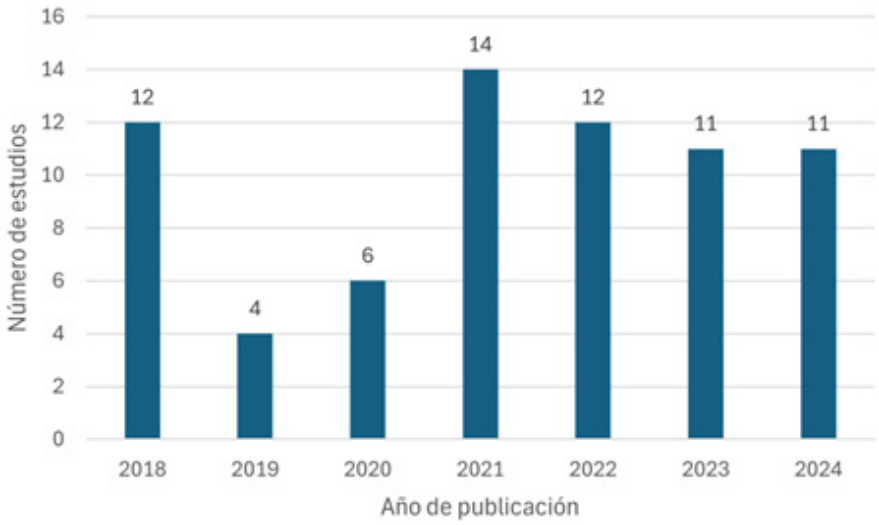

Figura 1. Publicaciones por año

Fuente: elaboración propia.

A continuación, se presentan los resultados para cada pregunta de investigación planteada.

Métodos para la automatización de la generación de casos de prueba de soft-ware (PI1)

En total, se identificaron 30 métodos, enfoques o algoritmos utilizados para la generación automática de casos de prueba. Cada método o enfoque busca generar casos de prueba a partir de una entrada específica. A continuación, se presentan las respuestas a las subpreguntas plan-teadas, las cuales corresponden a las características, retos en el uso y forma de evaluación de los métodos.

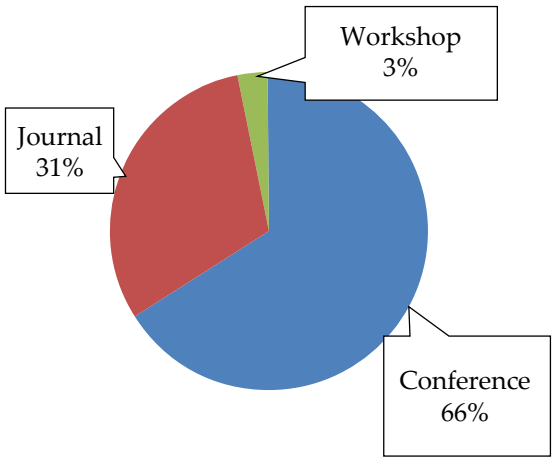

Figura 2. Publicaciones por tipo

Fuente: elaboración propia.

Características de los métodos (PI1.1).

Las características de los métodos fueron analizadas considerando el tipo de entrada, el formato de salida, la técnica de prueba utilizada y el nivel del caso de prueba generado.

Entrada recibida. Los métodos emplean distintos tipos de entrada, categorizándolas en modelos y diagramas (p.ej., diagramas de actividad o modelos más específicos como de proceso de negocio y USL (*Use Case Specification Language*), especificaciones (formales o de API como OpenAPI y GraphQL), código fuente, casos de uso y requisitos (en XML o lenguaje natural), datos de entrenamiento y archivos de configuración. La mayoría de los métodos se basan en código fuente o modelos para la generación de los casos de prueba.

Formato de los casos de prueba generados. Se identificaron diversos formatos. Los casos de prueba basados en secuencias o acciones, como peticiones HTTP o de pasos de casos de uso fue el mencionado con más frecuencia en [EP8, EP14, EP25, EP32, EP34, EP37, EP44, EP59, EP67, EP64] (véase el Anexo A de Estudios Primarios seleccionados para la lista completa de estos y su identificación), seguido del formato

de casos de prueba basados en valores en datos, referenciado en [EP5, EP9, EP17, EP38, EP39, EP55]. Adicionalmente se categorizaron los casos de prueba estructurados (p.ej., en lenguaje Gherkin), código fuente y consultas (GraphQL). Algunos estudios no especifican el formato.

Técnicas de prueba utilizadas. Las técnicas más frecuentes son la prueba basada en búsqueda (SBST) y la basada en modelos (MBT). También se identificaron técnicas basadas en aprendizaje automático, especificaciones, requerimientos, propiedades y prueba aleatoria. La Tabla 6 muestra las técnicas identificadas.

Nivel de los casos de prueba. La mayoría de los métodos identificados en los estudios generan casos a nivel de sistema, reportado en [EP4, EP8, EP14, EP25, EP29, EP34, EP37, EP59, EP67]. También se reportaron casos de prueba de unidad en [EP5, EP18-19, EP32, EP60, EP64] y de integración en [EP32].

Tabla 6. *Técnica de generación de casos de prueba utilizada por los métodos*

Técnica de generación de casos de prueba	Estudios primarios
Prueba basada en búsqueda	[EP6, EP10, EP19, EP25, EP32, EP34, EP49, EP56, EP58-59, EP64, EP67]
Prueba basada en modelos	[EP4-5, EP8, EP13, EP29, EP39, EP70]
Prueba basada en aprendizaje automático	[EP17-18, EP60]
Prueba basada en especificaciones	[EP9, EP14]
Prueba basada en requisitos	[EP37, EP57]
Ejecución simbólica	[EP55]
Prueba aleatoria	[EP22]
Prueba basada en propiedades	[EP22]

Fuente: elaboración propia.

Retos en el empleo de los métodos. Los retos identificados incluyen limitaciones de modelos de IA y de la IA en sí misma, pues el rendimiento de estos puede afectar la calidad de los casos de prueba [EP4] (véase el Anexo de estudios primarios seleccionados para la lista completa de estos y su identificación), especialmente si no son entrenados o mantenidos adecuadamente. También se encontraron desafíos en las especificaciones y su estructura, particularmente porque podrían ser muy abstractas, incompletas o tener formas infinitas de implementación [EP18]. Adicionalmente, al implementar los métodos se enfrentan retos técnicos que requieren condiciones específicas relacionadas al formato de entrada o el estado del sistema bajo prueba [EP9, EP22, EP44]. Finalmente, los casos de prueba generados pueden ser de baja calidad al depender de la IA [EP4], llegar a ser repetitivos y no cumplir con su objetivo [EP34] de revelar fallas reales [EP55].

Forma de evaluación de los métodos. En la mayoría de los estudios se realizaron experimentos de comparación con otros métodos o algoritmos [EP6, EP9, EP10, EP18, EP19, EP34, EP38, EP39, EP49, EP56, EP60, EP70, EP57], particularmente, los estudios [EP6] y [EP9] evaluaron mediante un experimento de simulación y un experimento controlado, respectivamente. Seguido se encuentran los estudios empíricos, realizados en [EP25, EP32, EP55, EP59, EP67, EP64, EP58] y estudios de caso [EP8, EP13, EP17, EP22, EP37, EP44]. Los estudios [EP4, EP5, EP14, EP29] no reportan una evaluación de los métodos propuestos. En los artefactos que se pueden consultar mediante el enlace provisto en la introducción se encuentran las tablas con la información detallada.

Métricas y criterios utilizados para la evaluación. Las evaluaciones identificadas utilizaron distintos tipos de métricas, agrupadas en tres categorías: cobertura, que mide el alcance de los casos generados sobre el código o comportamiento; efectividad, que evalúa la capacidad del método para detectar fallos o generar casos útiles; y eficiencia, que considera el tiempo de ejecución, uso de recursos y volumen de pruebas. Por motivos de espacio, la Tabla 7 resume únicamente las métricas más comunes.

Tabla 7. *Criterios y métricas utilizadas para la evaluación de los métodos*

Criterio	Subcategoría	Métricas	Estudios primarios
Cobertura	Cobertura de código	Line coverage	[EP25, EP60, EP67]
		Branch coverage	[EP10, EP25, EP32, EP49, EP64, EP67]
Efectividad	Detección de fallas	Capacidad de detección de fallas	[EP22, EP38, EP6, EP25]
		Puntaje de mutación [EP60] o cobertura de mutación [EP64]	[EP55, EP60, EP64]
	Calidad de los casos de prueba generados	Casos de prueba / valores de entrada de prueba correctos	[EP60, EP38]

Fuente: elaboración propia.

Herramientas para la automatización de la generación de casos de prueba de software (PI2)

Se identificaron 38 herramientas, distribuidas en 40 estudios primarios. Las herramientas se listan en los artefactos provistos en el enlace mencionado. En la siguiente sección se presen-tan las respuestas a las subpreguntas planteadas. Características de las herramientas (PI2.1). Se abordan las características desde los aspectos de entrada recibida, formato de los casos de prueba generados, técnica de prueba utilizada, tipo de caso de prueba generado y tipos de software que se prueba.

Entrada recibida. Las herramientas reciben distintos tipos de entrada para la generación de los casos de prueba: especificaciones de API (p.ej., Swagger u OpenAPI), código fuente (en lenguajes como Java, R, C++, C, Python, etc.) y especificaciones formales haciendo uso de SOFL (Structured Object-Oriented

Formal Language). Adicionalmente, se encontró que algunas reciben la aplicación completa [EP15, EP21, EP30-31, EP43, EP45-46], modelos [EP1, EP16, EP20] y archivos de configuración [EP33, EP35, EP42].

Formato de los casos de prueba generados. Las herramientas generan los casos de prueba en diversos formatos: basados en transiciones y secuencias [EP1, EP7, EP15-16, EP31, EP36, EP42-43, EP45-46, EP48, EP63, EP68] y como casos de prueba Junit [EP21, EP23, EP26, EP40, EP50-53]. Además, se reportaron casos de prueba como secuencias de peticiones HTTP, ba-sados en tecnologías o *frameworks* específicos (p.ej., libfuzzer y Gtest) y basados en valores. Una herramienta genera casos de prueba como subexpresiones ERE (extended regular expres sion)[EP20].

Tabla 8. *Técnicas de generación de casos de prueba utilizada por las herramientas*

Técnica de generación de casos de prueba	Estudios primarios
Prueba basada en búsqueda	[EP7, EP33, EP35-36, EP40-41, EP47, EP53, EP65]
Prueba basada en Modelos de Lenguaje Grande	[EP51-53, EP61]
Prueba basada en modelos	[EP16, EP20, EP30, EP68]
Prueba basada en aprendizaje por refuerzo	[EP31, EP43]
Prueba combinatoria	[EP21, EP24]
Prueba aleatoria	[EP12, EP27]
Prueba por mutación (basada en búsqueda)	[EP12, EP61]

Fuente: Autores.

Técnicas de prueba utilizadas. La técnica utilizada con más frecuencia es la prueba basada en búsqueda (SBST) y la prueba basada en LLM. Además, se reportaron las técnicas de prueba basada en modelos (MBT), basada en aprendizaje por refuerzo, prueba combinatoria, etc. Algunos estudios no reportan el tipo de técnica de prueba usada. La Tabla 8 muestra las técnicas reportadas.

Nivel de los casos de prueba. El nivel de prueba con mayor ocurrencia es el unitario, reportado por [EP21, EP23, EP28, (EP47, EP66), EP50-53, EP61, EP63] seguido del de sistema, reportado por [(EP35, EP40, EP65), EP42-43, EP45, EP48] Finalmente el de integración fue reportado por los estudios [EP15-16, (EP35, EP40), EP36] y el de aceptación en la herramienta del estudio [EP62].

Tipo de software. Se identificaron herramientas enfocadas en aplicaciones web [EP31, EP42, EP68], móviles [EP30, EP43, EP45], y en APIs o servicios [EP24, EP26, (EP27, EP41), (EP35, EP40, EP65), EP54]. En cuanto a los lenguajes de programación del software bajo prueba, Java fue el más común [EP12, EP21, EP23, (EP35, EP40, EP65), EP50-53, EP63], seguido por Python [(EP47, EP66), EP61]. También se identificaron casos de prueba dirigidos a software desarrollado en C, C++, JavaScript, Eiffel y C#. La Tabla 9 y la Tabla 10 presentan la información reportada en los EP.

Retos en el empleo de las herramientas. Las limitaciones incluyen la generación de pruebas, restricciones específicas de herramienta, limitaciones en el manejo de estructuras del lenguaje, validación y verificación, y configuración. Algunas herramientas no generan casos en ciertas condiciones [EP2, EP3, EP61] o producen pruebas innecesarias o con errores sintácticos [EP23, EP52]. Otras, como Pynguin, presentan dificultades para aislar la ejecución de pruebas [EP47, EP66] o dependen de listas predefinidas de funcionalidades [EP46]. Entre las limitaciones específicas se encuentran el soporte limitado de tipos de datos [EP2], el acoplamiento restringido [EP15] y la falta de interacción con bases de datos o módulos binarios [EP40, EP66]. En cuanto a las estructuras del lenguaje, se reportan problemas con plantillas y bibliotecas STL en C++ [EP28], inferencia de tipos en Python [EP4, EP66], y polimorfismo en software orientado a objetos [EP36]. En la validación, los oráculos de se basan en criterios simplistas, como códigos de respuesta HTTP [EP26, EP41], lo que afecta la precisión del resultado. Finalmente, algunas herramientas no cumplen los objetivos de prueba [EP61] o requieren configuraciones complejas [EP35, EP40].

Tabla 9. *Tipo de software, tecnología o plataforma hacia la que se orientan los casos de prueba*

Tipo de software	Estudios primarios
Aplicaciones web	[EP31, EPP42, EP68]
Aplicaciones móviles	[EP30, EP43, EP45]
Aplicaciones *serverless*	[EP16]
APIs y servicios web (APIs RESTful)	[EP24, EP26-P27, EP35, EP40-41, EP54, EP65]

Fuente: Autores.

Tabla 10. *Lenguaje o frameworks que prueban los casos de prueba generados por las herramientas*

Lenguaje	Estudios primarios
Java	[EP12, EP21, EP23, EP35, EP40, EP50-53, EP63, EP65]
Python	[EP47, EP61, EP66]
C	[EP33]
C++	[EP28]
JavaScript	[EP68]
Eiffel	[EP69]
C# (.NET)	[EP15]

Fuente: Autores.

Forma de evaluación de las herramientas. Los experimentos de comparación con otras herramientas es la forma más común, usada en [EP2, EP7, EP12, EP20, EP24, EP30-31, EP33, EP45, EP47, EP51-52, EP61, EP63, EP68-69], seguido de los estudios empíricos en [EP21, EP26-27, EP36, EP40, EP42-43, EP46, EP48, EP66] y los estudios de caso en [EP1, EP3, EP11, EP16,

EP35, EP41, EP62]. En los estudios [EP23, EP28, EP50, EP65] se realizaron otros tipos de evaluación. La Tabla con la información detallada se encuentra en los artefactos provistos en el enlace incluido en la introducción. Métricas y criterios utilizados para la evaluación. Las métricas más usadas se listan en la Tabla 11.

Tabla 11. *Criterios y métricas utilizadas para la evaluación de las herramientas.*

Criterio	Subcategoría	Métricas	Estudios primarios
Cobertura	Cobertura de código	Statement coverage Line coverage Instruction coverage	[EP28, EP31, EP35, EP40, EP43, EP51, EP68]
		Branch coverage	[EP28, EP31, EP33, EP43, EP47, EP63, EP69]
		Code coverage	[EP21, EP30, EP41, EP50]
Efectividad	Detección de fallas	Número de fallas detectadas/ reveladas/ encontradas	[EP40, EP43, EP26, EP42, EP46]
		Puntaje de mutación	[EP2, EP11, EP61]
	Calidad de los casos de prueba generados	Precisión	[EP1, EP48]
		Recall	[EP1, EP48]

Fuente: Autores.

Discusión

Nuestro estudio presenta limitaciones importantes que deben considerarse al interpretar los resultados. Primero, la exclusión de literatura gris puede haber omitido herramientas industriales significativas. Esta decisión pudo sesgar los resultados hacia enfoques académicos, dejando fuera soluciones empleadas en empresas tecnológicas que no publican formalmente sus avances. Esto podría implicar que el panorama real de la automatización que se lleva a cabo en la industria esté más avanzado de lo que reflejan nuestros resultados. Segundo, la ausencia de evaluación de calidad de estudios primarios impide distinguir entre evidencia robusta y preliminar. Esto puede haber afectado la validez de algunas tendencias observadas, particularmente aquellas basadas en estudios con muestras reducidas o sin replicación experimental. Finalmente, la falta de técnicas de *snowballing* podría haber conducido a la exclusión de estudios importantes.

En comparación con trabajos previos, este mapeo sistemático amplía el alcance, periodo de análisis y fuentes consultadas, ofreciendo una visión más actualizada del campo. Por ejemplo, Navarro e Ibarra [9] se centran exclusivamente en técnicas basadas en NLP y, aunque abarca un periodo amplio, limita su análisis a ese dominio específico. Baqar y Khanda [8] discuten la generación de pruebas asistida por IA, pero no siguen un protocolo sistemático ni definen preguntas de investigación. Por su parte, Verma et al. [6] incluyen 125 publicaciones y ofrecen una visión general de técnicas y herramientas, pero analiza un periodo de tiempo diferente y no recopila la manera en que se evaluaron herramientas. Potuzak y

Lipka [7] tampoco plantean preguntas de investigación al clasificar su estudio como un trabajo exploratorio con el objetivo de llegar una RSL. En cambio, Rodrigues et al. [14] sí aplica un enfoque metodológico riguroso, pero restringido a la generación de casos desde requisitos. Frente a estos antecedentes, el presente estudio abarca el periodo 2018–2024, utiliza cuatro fuentes principales (IEEE, ACM, *Springer* y *Science Direct*) y formula ocho preguntas de investigación que permiten mapear tanto métodos como herramientas, además de analizar las métricas y enfoques de evaluación empleados.

Considerando lo anterior, el mapeo sistemático identificó 30 métodos y 38 herramientas para la generación automática de casos de prueba, lo cual revela que la prueba basada en búsqueda (SBST) y la prueba basada en modelos (MBT) dominan el campo, mientras que los modelos de lenguaje grande emergen como una técnica prometedora, pero con limitaciones significativas como errores sintácticos, falta de precisión y limitaciones al extraer información del código fuente. Los hallazgos también revelan una tendencia hacia la generación de pruebas para APIs RESTful e interfaces gráficas, enfocándose en lenguajes como Java o Python.

La predominancia de SBST y MBT indica que el campo de la prueba de software ha madurado hacia enfoques sistemáticos y consolidados. Resultados similares fueron obtenidos por Verma et al. [6] y Rodrigues et al. [14], quienes identificaron a MBT y SBST como las técnicas más empleadas, principalmente por su capacidad para incrementar la calidad del producto, reducir costos y acelerar el lanzamiento al mercado. No obstante, este estudio revela una evolución significativa: la emergencia en el uso de LLMs con herramientas como ChatUniTest o TestSpark, lo cual complementa los trabajos de Baqar y Khanda [8] y Navarro e Ibarra [9], en donde se refleja una transición en el paradigma de automatización. Los resultados señalan que, si bien los LLMs amplían la accesibilidad de la generación automática de casos de prueba (por ejemplo, permiten generar casos de prueba a partir de descripciones en lenguaje natural), aún no igualan la precisión de métodos tradicionales.

La concentración de las herramientas en lenguajes como Java y Python o en APIs REST sugiere una brecha en la cobertura de otras tecnologías o tipos de software. Esto puede observarse en el trabajo de Verma et al. [6], en el cual las 24 herramientas identificadas, 13 se enfocan en Java. Del mismo modo, aunque algunas propuestas abordan pruebas de seguridad [EP37, EP59] o rendimiento [EP8, EP34], son escasas las iniciativas que integran estas herramientas en entornos CI/CD o que generan *test oracles* automáticos con alta confiabilidad. En este último punto, aún es común el uso de validaciones superficiales, como verificar únicamente códigos HTTP [EP26, EP41].

Para profesionales de la industria, estos hallazgos sugieren que: (1) la elección de herramientas o métodos debe considerar el lenguaje de programación, tipo de prueba y nivel de automatización deseado, (2) equipos que hacen uso de tecnologías como JavaScript, C#, entre otras, así como otros tipos de software o plataformas (aplicaciones de escritorio, p.ej.) enfrentan una brecha de herramientas y métodos para la

generación de casos de prueba que representa una oportunidad de inversión, y (3) la adopción de LLMs para generación de pruebas debe realizarse con validación humana rigurosa debido a los problemas de precisión identificados, además, su elección debe considerar este esfuerzo extra.

Los hallazgos presentados en nuestro estudio confirman y amplían tendencias identificadas en los trabajos relacionados, pero revelan cambios significativos. Coincidimos con Verma et. al. [6] y Potuzak y Lipka [7] en que SBST y MBT son técnicas dominantes, sin embargo, nuestro estudio documenta una transición emergente hacia LLMS que no fue capturada en esos trabajos debido a su ventana temporal más temprana o su naturaleza exploratoria. Esta identificación de LLMs como técnica emergente complementa las observaciones de Baqar y Khanda [8] sobre el rol de la IA, pero nuestro análisis sistemático añade evidencia concreta sobre sus limitaciones prácticas (errores sintácticos, validaciones superficiales con oráculos simplistas) que no fueron cuantificadas en su trabajo exploratorio. Respecto a Navarro e Ibarra [10], que se enfocaron exclusivamente en NLP, nuestro estudio provee un contexto más amplio al posicionar estas técnicas dentro del ecosistema completo de automatización, revelando que, aunque NLP/LLMs aumentan la accesibilidad, aún no superan la precisión de métodos tradicionales.

Conclusiones y trabajo futuro

La creciente complejidad del software moderno y la presión por ciclos de desarrollo acelerados han intensificado la necesidad de automatizar la generación de casos de prueba, con la transformación de esta actividad de un proceso manual propenso a errores hacia un desafío en pruebas de software que requiere soluciones efectivas.

Es importante destacar que, aunque este mapeo sistemático puede servir de base a futuras investigaciones en esta área, los resultados presentados no son concluyentes. El estudio se basa exclusivamente en literatura indexada en bases académicas, sin incluir literatura gris ni aplicar técnicas de *snowballing*, y no se realizó una evaluación exhaustiva de la calidad de los estudios primarios. Estas limitaciones pueden generar sesgos o vacíos en la evidencia analizada, por lo que los hallazgos deben interpretarse con cautela.

El mapeo sistemático de 70 estudios primarios revela un panorama de la generación automática de casos de prueba caracterizado por la consolidación de técnicas establecidas y la emergencia de determinados enfoques. La prueba basada en búsqueda (SBST) y la prueba basada en modelos (MBT) se han establecido como técnicas sólidas en el campo. Al mismo tiempo, la incorporación de modelos de lenguaje grande (LLMs) representa un cambio que, aunque prometedor, enfrenta desafíos significativos de precisión y confiabilidad. El análisis evidencia una especialización hacia tecnologías como Java y APIs RESTful, mientras que importantes brechas persisten en tecnologías como aplicaciones de escritorio u otros lenguajes de programación, pruebas de seguridad e integración con pipelines CI/CD.

Nuestro estudio contribuye al campo al proporcionar una caracterización sistemática de 30 métodos y 38 herramientas reportadas entre 2018 y 2024, identificando no solo sus características técnicas, sino también los retos prácticos en su implementación y las métricas empleadas para su evaluación. Los hallazgos ofrecen a investigadores un mapa de tendencias actuales y brechas de investigación, mientras que a profesionales les proporcionan criterios para seleccionar técnicas apropiadas según el contexto tecnológico de sus proyectos.

Es fundamental reconocer que este mapeo requiere complementarse con la literatura gris, técnicas de *snowballing* y una evaluación rigurosa de la calidad de los estudios primarios. Las investigaciones futuras deberían: (1) incluir estos elementos para fortalecer la validez de los hallazgos e identificar sesgos o lagunas en la evidencia analizada; (2) explorar la adopción real de estas técnicas y herramientas en contextos industriales diversos, particularmente en lenguajes de programación y tipos de sistemas actualmente subrepresentados; (3) realizar estudios comparativos rigurosos entre LLMs y métodos tradicionales; (4) desarrollar métricas de calidad para casos de prueba generados automáticamente que vayan más allá de la cobertura de código, considerando aspectos como mantenibilidad; y (5) investigar la integración efectiva de herramientas de generación automática en pipelines CI/CD. Al centrarse en estas áreas, los investigadores pueden profundizar en la comprensión de los desafíos de la generación automática de casos de prueba y su aplicación en entornos reales y variados, con el fin de mejorar la efectividad general de las prácticas de aseguramiento de calidad del software.

Referencias

[1] M. Sharma, C. Sharma, y S. Sharma, "Creation of an Improved Software Testing Framework for the Cloud Environment," en 2022 International Conference on 4th Industrial Revolution Based Technology and Practices (ICFIRTP). IEEE, 2022, pp. 187–192. doi: https://doi.org/10.1109/ICFIRTP56122.2022.10063210

[2] K. R. Halani, Kavita, y R. Saxena, "Critical Analysis of Manual Versus Automation Testing," en 2021 International Conference on Computational Performance Evaluation (ComPE). IEEE, 2021, pp. 132–135. doi: https://doi.org/10.1109/ComPE53109.2021.9752388

[3] IEEE Computer Society, "SWEBOK: Guide to the Software Engineering Body of Knowledge," versión 4.0, IEEE Computer Society, 2024. [En línea]. Disponible en: https://www.computer.org/education/bodies-of-knowledge/software-engineering

[4] E. N. Narciso, M. E. Delamaro, y F. De Lourdes Dos Santos Nunes, "Test case selection: A systematic literature review," International Journal of Software Engineering and Knowledge Engineering, vol. 24, núm. 4, pp. 653–676, may. 2014. doi: https://doi.org/10.1142/S0218194014500259

[5] P. Fröhlich y J. Link, "Automated test case generation from dynamic models," Lecture Notes in Computer Science, vol. 1850, pp. 472–491, 2000. doi: https://doi.org/10.1007/3-540-45102-1_23

[6] A. S. Verma, A. Choudhary, y S. Tiwari, "Software Test Case Generation Tools and Techniques: A Review," International Journal of Mathematical, Engineering and Management Sciences, vol. 8, núm. 2, pp. 293–315, abr. 2023. doi: https://doi.org/10.33889/IJMEMS.2023.8.2.018

[7] T. Potuzak y R. Lipka, "Current Trends in Automated Test Case Generation," en Proceedings of the 18th Conference on Computer Science and Intelligence Systems (FedCSIS 2023), 2023, pp. 627–636. doi: https://doi.org/10.15439/2023F9829

[8] M. Baqar y R. Khanda, "The Future of Software Testing: AI-Powered Test Case Generation and Validation," en Lecture Notes in Networks and Systems, vol. 1424. Springer, 2025. doi: https://doi.org/10.1007/978-3-031-92605-1

[9] J. Navarro y R. Ibarra, "Automatic test case generation using natural language processing: A systematic mapping study," Information and Software Technology, vol. 189, art. 107929, ene. 2026. doi: https://doi.org/10.1016/j.infsof.2025.107929

[10] B. A. Kitchenham, D. Budgen, y P. Brereton, Evidence-Based Software Engineering and Systematic Reviews, 1.ª ed. Boca Raton, FL: Chapman and Hall/CRC, 2015. doi: https://doi.org/10.1201/b19467

[11] H. Zhang, M. A. Babar, y P. Tell, "Identifying relevant studies in software engineering," Information and Software Technology, vol. 53, núm. 6, pp. 625–637, jun. 2011. doi: https://doi.org/10.1016/j.infsof.2010.12.010

[12] J. Popay et al., "Guidance on the Conduct of Narrative Synthesis in Systematic Reviews: A Product from the ESRC Methods Programme," versión 1.0, ESRC Methods Programme, Lancaster University, 2006. doi: https://doi.org/10.13140/2.1.1018.4643

[13] N. R. Haddaway, B. Macura, P. Whaley, y A. S. Pullin, "ROSES RepOrting standards for Systematic Evidence Syntheses: pro forma, flow-diagram and descriptive summary of the plan and conduct of environmental systematic reviews and systematic maps," Environmental Evidence, vol. 7, núm. 1, pp. 1–8, mar. 2018. doi: https://doi.org/10.1186/s13750-018-0121-7

[14] A. Rodrigues, J. Vilela, y C. Silva, "A Systematic Mapping Study on Techniques for Generating Test Cases from Requirements," en Proceedings of the 9th International Conference on Internet of Things, Big Data and Security (IoTBDS 2024), 2024, pp. 141–148. doi: https://doi.org/10.5220/0012551900003705

Anexo A. Lista de estudios primarios seleccionados.

ID	Título	Autores	Año
EP1	STATETest: An Automatic Test Case Generation Framework for State Transition Testing	Adisak Intana; Arthiyaporn Sawedsuthiphan	2023
EP2	A Tool to Support Vibration Testing Method for Automatic Test Case Generation and Test Result Analysis	Kenya Saiki; Shaoying Liu; Hiroyuki Okamura; Tadashi Dohi	2021
EP3	SYNTest: Prototype of Syntax Test Case Generation Tool	Adisak Intana; Monchanok Thongthep; Phatcharee Thepnimit; Phaplak Saethapan; Tanawat Monpipat	2020
EP4	A new approach for automatic test case generation from use case diagram using LLMs and prompt engineering	Lahbib Naimi; El Mahi Bouziane; Mohamed Manaouch; Abdeslam Jakimi	2024
EP5	An Approach to Automatic Test Case Generation for Unit Testing	Pan Liu; Zhenning Xu; Jun Ai	2018
EP6	Automatic Software Test Case Generation and Optimization Based on Multi-objective Genetic Algorithm	Wei Yunhua; Jin Yuanwu	2024
EP7	ACTUM – tool for automatic class testing using meta-heuristics	G. Neetu	2022
EP8	Model-Based Test Case Generation Approach for Mobile Applications Load Testing using OCL Enhanced Activity Diagrams	Amira Ali; Huda Amin Maghawry; Nagwa Badr	2021
EP9	Automatic Test Case and Test Oracle Generation Based on Functional Scenarios in Formal Specifications for Conformance Testing	Shaoying Liu; Shin Nakajima	2020
EP10	Research on Multi-objective Test Case Generation Based on Cuckoo Search	He Haixian; Feng Jing	2018
EP11	Automatic Generation of Test Cases from Formal Specifications using Mutation Testing	Román Jaramillo Cajica; Raúl Ernesto González Torres; Pedro Mejía Álvarez	2021
EP12	An Experimental Tool for Search-Based Mutation Testing	Muhammad Bilal Bashir; Aamer Nadeem	2018
EP13	Automated Test Case Generation from Activity Diagram using Depth-First Search	Ignasius Teguh Raharjo Rubiyo; Muhammad Johan Alibasa; Rosa Reska Riskiana	2023

EP14	Automated Test Cases Generation From Requirements Specification	Mohammed Lafi; Thamer Alrawashed; Ahmad Munir Hammad	2021
EP15	IntegrationDistiller: Automating Integration Analysis and Testing of Object-Oriented Applications	Mehrdad Saadatmand	2019
EP16	Automatic Test Case Generation for Serverless Applications	Stefan Winzinger; Guido Wirtz	2022
EP17	Guided Test Case Generation through AI Enabled Output Space Exploration	Christof Budnik; Marco Gario; Georgi Markov; Zhu Wang	2018
EP18	Generating Software Tests for Mobile Applications Using Fine-Tuned Large Language Models	Jacob Hoffmann; Demian Frister	2024
EP19	The Applied Research of Improved Genetic Algorithm in Data Automatic Generation of Software Test	Ming Huang; SiNing Liu; Xuan Jiao	2021
EP20	MTTool: A Tool for Software Modeling and Test Generation	Pan Liu; Zhenning Xu	2018
EP21	TackleTest: A Tool for Amplifying Test Generation via Type-Based Combinatorial Coverage	Rachel Tzoref-Brill; Saurabh Sinha; Antonio Abu Nassar; Victoria Goldin; Haim Kermany	2022
EP22	Automatic Property-based Testing of GraphQL APIs	Stefan Karlsson; Adnan Čaušević´; Daniel Sundmark	2021
EP23	Leveraging Code-Test Co-evolution Patterns for Automated Test Case Recommendation	Samiha Shimmi; Mona Rahimi	2022
EP24	Combinatorial Testing of RESTful APIs	Huayao Wu; Lixin Xu; Xintao Niu; Changhai Nie	2022
EP25	Improving Test Case Generation for REST APIs Through Hierarchical Clustering	Dimitri Stallenberg; Mitchell Olsthoorn; Annibale Panichella	2021
EP26	RESTTESTGEN: Automated Black-Box Testing of RESTful APIs	Emanuele Viglianisi; Michael Dallago; Mariano Ceccato	2020
EP27	Empirical Comparison of Black-box Test Case Generation Tools for RESTful APIs	Davide Corradini; Amedeo Zampieri; Michele Pasqua; Mariano Ceccato	2021
EP28	CITRUS: Automated Unit Testing Tool for Real-world C++ Programs	Robert Sebastian Herlim; Yunho Kim; Moonzoo Kim	2022
EP29	A Transformation-Based Method for Test Case Automatic Generation from Use Cases	Chu Thi Minh Hue; Dang Duc Hanh; Nguyen Ngoc Binh	2018
EP30	Fastbot: A Multi-Agent Model-Based Test Generation System	Tianqin Cai; Zhao Zhang; Ping Yang	2020

EP31	A Reinforcement Learning Approach to Generating Test Cases for Web Applications	Xiaoning Chang; Zheheng Liang; Yifei Zhang; Lei Cui; Zhenyue Long; Guo-quan Wu	2023
EP32	Enhanced Genetic Algorithm for Automatic Generation of Unit and Integration Test Suite	Bui Thi Mai Anh	2020
EP33	OCELOT: A Search-Based Test-Data Generation Tool for C	Simone Scalabrino; Giovanni Grano; Dario di Nucci; Michele Guerra; Andrea de Lucia; Harald C. Gall	2018
EP34	Automated Performance Testing Based on Active Deep Learning	Ali Sedaghatbaf; Mahshid Helali Moghadam; Mehrdad Saadatmand	2021
EP35	EvoMaster: Evolutionary Multi-context Automated System Test Generation	Andrea Arcuri	2018
EP36	Generating Class-Level Integration Tests Using Call Site Information	Pouria Derakhshanfar; Xavier Devroey; Annibale Panichella; Andy Zaidman; Arie van Deursen	2023
EP37	A Natural Language Programming Approach for Requirements-Based Security Testing	Phu X. Mai; Fabrizio Pastore; Arda Goknil; Lionel C. Briand	2018
EP38	ARTE: Automated Generation of Realistic Test Inputs for Web APIs	Juan C. Alonso; Alberto Martin-Lopez; Sergio Segura; José María García; Antonio Ruiz-Cortés	2022
EP39	SPECMATE: Automated Creation of Test Cases from Acceptance Criteria	Jannik Fischbach; Andreas Vogelsang; Dominik Spies; Andreas Wehrle; Maximilian Junker; Dietmar Freudenstein	2020
EP40	RESTful API Automated Test Case Generation with EvoMaster	Andrea Arcuri	2019
EP41	RESTler: Stateful REST API Fuzzing	Vaggelis Atlidakis; Patrice Godefroid; Marina Polishchuk	2019
EP42	Cytestion: Automated GUI Testing for Web Applications	Thiago Santos de Moura; Everton L. G. Alves; Hugo Feitosa de Figueirêdo; Cláudio de Souza Baptista	2023
EP43	Deep Reinforcement Learning based Android Application GUI Testing	Eliane Collins; Arilo Neto; Auri Vincenzi; José Maldonado	2021
EP44	Automated Black-Box Testing of Mass Assignment Vulnerabilities in RESTful APIs	Davide Corradini; Michele Pasqua; Mariano Ceccato	2023

EP45	Paladin: Automated Generation of Reproducible Test Cases for Android Apps	Yun Ma; Yangyang Huang; Ziniu Hu; Xusheng Xiao; Xuanzhe Liu	2019
EP46	Augusto: Exploiting Popular Functionalities for the Generation of Semantic GUI Tests with Oracles	Leonardo Mariani; Mauro Pezzè; Daniele Zuddas	2018
EP47	Pynguin: Automated Unit Test Generation for Python	Stephan Lukasczyk; Gordon Fraser	2022
EP48	Avgust: automating usage-based test generation from videos of app executions	Yixue Zhao; Saghar Talebipour; Kesina Baral; Hyojae Park; Leon Yee; Safwat Ali Khan; Yuriy Brun; Nenad Medvidovic´; Kevin Moran	2022
EP49	Automatic Test Case Generation Method Based on Improved Whale Optimization Algorithm	Jing Wang; Weidong Zhao	2021
EP50	Towards a Test Case Generation Tool Based on Functional Requirements	Daniel David Fernandes; Rita Suzana Pitangueira Maciel	2021
EP51	ChatUniTest: A Framework for LLM-Based Test Generation	Yinghao Chen; Zehao Hu; Chen Zhi; Junxiao Han; Shuiguang Deng; Jianwei Yin	2024
EP52	HITS: High-coverage LLM-based Unit Test Generation via Method Slicing	Zejun Wang; Kaibo Liu; Ge Li; Zhi Jin	2024
EP53	TestSpark: IntelliJ IDEA's Ultimate Test Generation Companion	Arkadii Sapozhnikov; Mitchell Olsthoorn; Annibale Panichella; Vladimir Kovalenko; Pouria Derakhshanfar	2024
EP54	RapiTest: Continuous Black-Box Testing of RESTful Web APIs	Duarte Felício; José Simão; Nuno Datia	2023
EP55	Path-directed source test case generation and prioritization in metamorphic testing	Chang-ai Sun; Baoli Liu; An Fu; Yiqiang Liu; Huai Liu	2022
EP56	Automation of software test data generation using genetic algorithm and reinforcement	Mehdi Esnaashari; Amir Hossein Damia	2021
EP57	Test case information extraction from requirements specifications using NLP-based unified boilerplate approach	Jin Wei Lim; Thiam Kian Chiew; Moon Ting Su; Simying Ong; Hema Subramaniam; Mumtaz Begum Mustafa; Yin Kia Chiam	2024
EP58	Test suite generation with the Many Independent Objective (MIO) algorithm	Andrea Arcuri	2018
EP59	Parallel evolutionary test case generation for web applications	Weiwei Wang; Shumei Wu; Zheng Li; Ruilian Zhao	2023

EP60	A3Test: Assertion-Augmented Automated Test case generation	Saranya Alagarsamy; Chakkrit Tantithamthavorn; Aldeida Aleti	2024
EP61	Effective test generation using pre-trained Large Language Models and mutation testing	Arghavan Moradi Dakhel; Amin Nikanjam; Vahid Majdinasab; Foutse Khomh; Michel C. Desmarais	2024
EP62	Automatic creation of acceptance tests by extracting conditionals from requirements: NLP approach and case study	Jannik Fischbach; Julian Frattini; Andreas Vogelsang; Daniel Mendez; Michael Unterkalmsteiner; Andreas Wehrle; Pablo Restrepo Henao; Parisa Yousefi; Tedi Juricic; Jeannette Radduenz; Carsten Wiecher	2023
EP63	Diversity-driven unit test generation	Marcus Kessel; Colin Atkinson	2022
EP64	Toward granular search-based automatic unit test case generation	Fabiano Pecorelli; Giovanni Grano; Fabio Palomba; Harald C. Gall; Andrea De Lucia	2024
EP65	Building an open-source system test generation tool: lessons learned and empirical analyses with EvoMaster	Andrea Arcuri; Man Zhang; Asma Belhadi; Bogdan Marculescu; Amid Golmohammadi; Juan Pablo Galeotti; Susruthan Seran	2023
EP66	An empirical study of automated unit test generation for Python	Stephan Lukasczyk; Florian Kroiß; Gordon Fraser	2023
EP67	Resource and dependency based test case generation for RESTful Web services	Man Zhang; Bogdan Marculescu; Andrea Arcuri	2021
EP68	Model-based Automated Testing of JavaScript Web Applications via Longer Test Sequences	Pengfei Gao; Yongjie Xu; Fu Song; Taolue Chen	2022
EP69	Seeding Contradiction: a Fast Method for Generating Full-Coverage Test Suites	Li Huang; Bertrand Meyer; Manuel Oriol	2024
EP70	Automatic test cases generation from business process models	Arezoo Yazdani Seqerloo; Mohammad Javad Ami- ri; Saeed Parsa; Mahnaz Koupaee	2018