

Impacto de las pruebas automatizadas con inteligencia artificial en la calidad del desarrollo de aplicaciones web: comparativa entre SELENIUMBASE, TESTIM y MABL.

Gallardo Cueva, Adela Sofía

Vicerrectorado de Investigación, Innovación y Transferencia Tecnológica

Centro de Estudios de Posgrado

Maestría en Ingeniería en Software

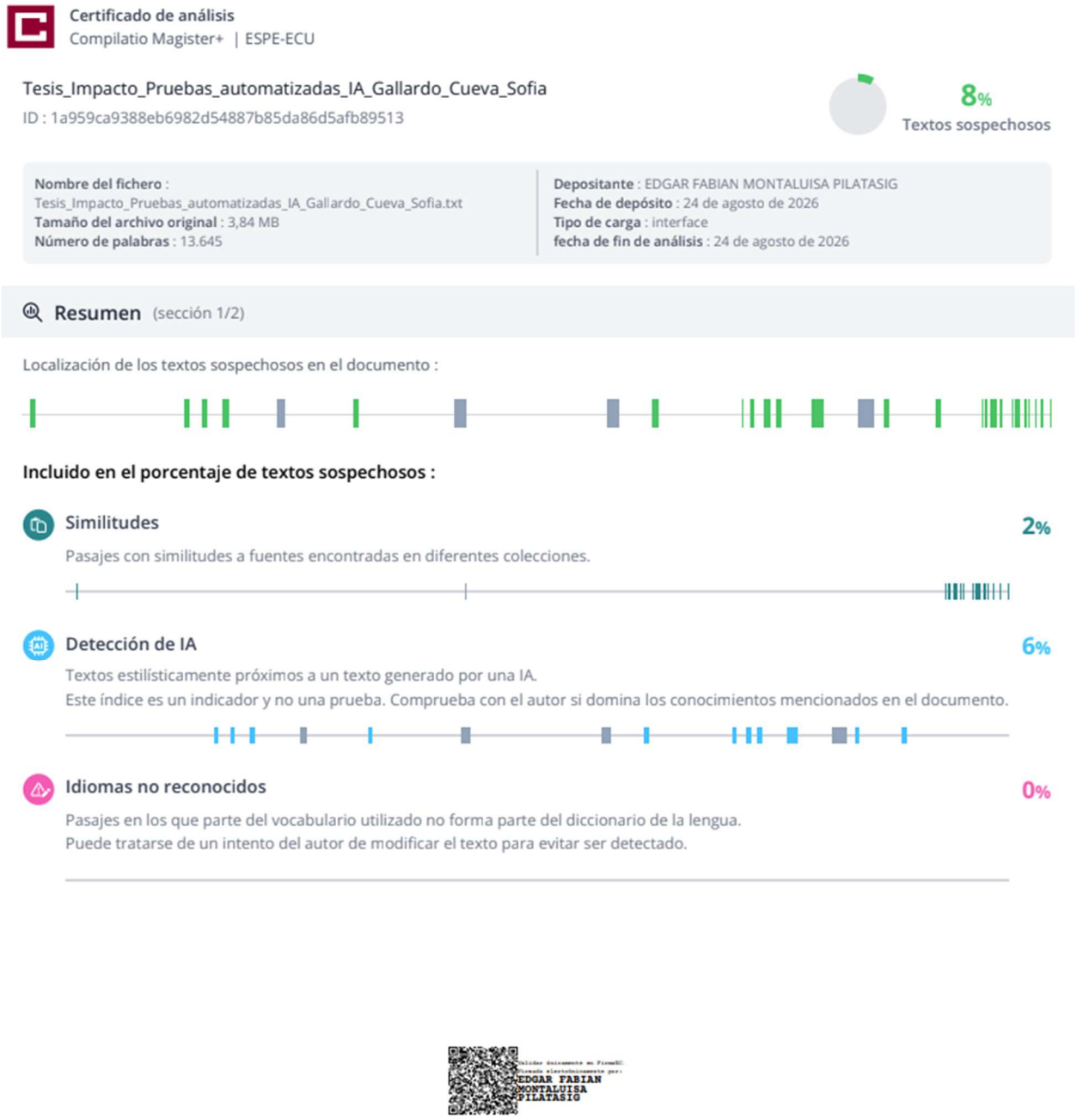
Tesis, previo a la obtención del título de Magíster en Ingeniería en Software

Msc. Montaluisa Pilatasig, Edgar Fabián

24 de agosto de 2026

Latacunga

Reporte de Verificación de Contenido



Certificado de análisis
Compilatio Magister+ | ESPE-ECU

Tesis_Impacto_Pruebas_automatizadas_IA_Gallardo_Cueva_Sofia
ID : 1a959ca9388eb6982d54887b85da86d5afb89513

8%
Textos sospechosos

Nombre del fichero :
Tesis_Impacto_Pruebas_automatizadas_IA_Gallardo_Cueva_Sofia.txt
Tamaño del archivo original : 3,84 MB
Número de palabras : 13.645

Depositante : EDGAR FABIAN MONTALUISA PILATASIG
Fecha de depósito : 24 de agosto de 2026
Tipo de carga : interface
fecha de fin de análisis : 24 de agosto de 2026

Resumen (sección 1/2)

Localización de los textos sospechosos en el documento :

Incluido en el porcentaje de textos sospechosos :

Similitudes 2%
Pasajes con similitudes a fuentes encontradas en diferentes colecciones.

Detección de IA 6%
Textos estilísticamente próximos a un texto generado por una IA.
Este índice es un indicador y no una prueba. Comprueba con el autor si domina los conocimientos mencionados en el documento.

Idiomas no reconocidos 0%
Pasajes en los que parte del vocabulario utilizado no forma parte del diccionario de la lengua.
Puede tratarse de un intento del autor de modificar el texto para evitar ser detectado.

EDGAR FABIAN MONTALUISA PILATASIG

Msc. Montaluisa Pilatasig, Edgar Fabián

C.C.: 0501960900

Director

Certificación

Vicerrectorado de Investigación, Innovación y Transferencia Tecnológica

Centro de Estudios de Posgrados

Certificación

Certifico que la tesis, **"Impacto de las pruebas automatizadas con inteligencia artificial en la calidad del desarrollo de aplicaciones web: comparativa entre SeleniumBase, Testim y Mabl"** fue realizado por la **Ing. Gallardo Cueva, Adela Sofía**, el mismo que ha sido revisado y analizado en su totalidad por la herramienta de verificación de similitud de contenido; por lo tanto, cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustente públicamente.

Latacunga, 24 de agosto de 2026

EDGAR FABIAN MONTALUISA PILATASIG

.....

Ing. Montaluisa Pilatásig Edgar Fabián, Mgtr.

C.C.: 0501960900

Director Titulación

UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Vicerrectorado de Investigación, Innovación y Transferencia Tecnológica

Centro de Estudios de Posgrados

Responsabilidad de Autoría

Yo, **Gallardo Cueva, Adela Sofía** con cédula de ciudadanía N°0501803134, declaro que el contenido, ideas y criterios de la Tesis: **"Impacto de las pruebas automatizadas con inteligencia artificial en la calidad del desarrollo de aplicaciones web: comparativa entre SeleniumBase, Testim y Mabl"** es de mí autoría y responsabilidad, cumpliendo con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Latacunga, 24 de agosto de 2026

Adela Sofia
Gallardo Cueva
Time Stamping
Security Data

Gallardo Cueva, Adela Sofía

C.C.:0501803134

UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Vicerrectorado de Investigación, Innovación y Transferencia Tecnológica

Centro de Estudios de Posgrados

Autorización de Publicación

Yo, **Gallardo Cueva, Adela Sofía** con cédula de ciudadanía N°0501803134, autorizo a la Universidad de las Fuerzas Armadas ESPE, publicar la Tesis: **"Impacto de las pruebas automatizadas con inteligencia artificial en la calidad del desarrollo de aplicaciones web: comparativa entre SeleniumBase, Testim y Mabl"** en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mí responsabilidad.

Latacunga, 24 de agosto de 2026

Adela Sofia
Gallardo Cueva
Time Stamping
Security Data

Gallardo Cueva, Adela Sofía

C.C.:0501803134

Agradecimiento

Expreso mi sincero agradecimiento a la noble Institución ESPE-L, por los conocimientos impartidos la orientación y el apoyo brindado en este proceso de formación y en especial al Coronal Alexis Armas, por su apoyo y gestión en el proceso de titulación para que pudiéramos alcanzar la culminación de esta etapa académica.

Ing. Gallardo Cueva, Adela Sofia

Dedicatoria

Este proyecto va dedicado a todas las personas que de una manera u otra me brindaron su tiempo, paciencia y apoyo a lo largo de este proceso hasta llegar a su culminación. En especial a mis hijos Dayis, Cami y Juanes por toda la comprensión y paciencia en este camino. A mi familia que siempre me apoyaron en todo momento incluso en los momentos críticos que tuve que pasar.

Ing. Gallardo Cueva, Adela Sofia

Índice de contenido

Carátula 1

Reporte de verificación de contenido 2

Certificación 3

Responsabilidad de autoría 3

Autorización de publicación 5

Agradecimiento 6

Dedicatoria 7

Índice de contenido 8

Índice de figuras 11

Índice de tablas 13

Resumen 14

Abstract 15

Capítulo I 16

Características generales del proyecto 16

Antecedentes 16

Líneas de investigación 19

Sub-líneas de investigación 19

Área de influencia 19

Planteamiento del problema 19

Justificación, importancia y alcance del proyecto 21

Objetivo general 23

Objetivos específicos 24

Formulación de la hipótesis 24

Categorización de las variables de investigación 24

Capítulo II 25

Marco teórico 25

Calidad de software y modelos de referencia 25

Pruebas, agilidad y entrega continua 26

Inteligencia artificial en la automatización de pruebas 28

Seleniumbase 30

Testim 30

Mabl 31

Documentación y evidencia 32

Capítulo III 34

Metodología 34

Tipo y diseño de investigación. 34

Unidad de análisis y elementos evaluados 35

Operacionalización de variables 37

Técnicas e instrumentos 38

Revisión documental 38

Observación sistemática de la ejecución 38

Siembra de defectos y cambio de interfaz 39

Cuestionario de percepción 39

Procedimiento 41

El sistema bajo prueba 42

Ejecución de pruebas 46

Capítulo IV 50

Análisis y discusión de resultados 50

Como se leen las cifras 50

Evidencias de ejecución sobre el modulo 55

Análisis comparativo de las herramientas 56

Cobertura funcional 57

Mantenimiento ante el cambio de identificador 62

Percepción de uso 64

Conclusiones 71

Recomendaciones 73

Referencias 75

Apéndices 79

Índice de figuras

Figura 1 Postura teórica del estudio. 33

Figura 2 Diseño experimental del caso. 35

Figura 3 Flujo funcional del afiliado. 36

Figura 4 Flujo funcional del afiliado. 37

Figura 5 Inserción de casos. 39

Figura 6 Procedimiento de laboratorio en siete bloques. 42

Figura 7 Inicio de sesión. 42

Figura 8 Login Valido y ejecución de prueba. 43

Figura 9 Formulario de agendamiento. 44

Figura 10 Listado de citas. 45

Figura 11 Especificación de los 18 casos 46

Figura 12 Ejecución manual. 47

Figura 13 Ejecución SeleniumBase. 48

Figura 14 Ejecución Testim. 48

Figura 15 Ejecución Mabl. 49

Figura 16 Resumen de la matriz. 51

Figura 17 Resultados manual. 52

Figura 13 Resultados selenium. 53

Figura 14 Resultados Testim. 53

Figura 15 Ejecución Mabl. 54

Figura 16 Ejecución sistema de pruebas 55

Figura 17 DEF-01 (CP-04): con clave vacía y correo existente. 55

Figura 18 DEF-06 (CP-16): /citas sin sesión lista turnos 56

Figura 19 Tasa de detección de los seis defectos sembrados. 59

Figura 25 Tiempo total de ejecución de la matriz. 60

Figura 26 Falsos positivos en la matriz de 18 casos. 61

Figura 27 Casos en los que las condiciones no coinciden. 62

Figura 28 Casos Rotos y autocurados. 63

Figura 29 SeleniumBase: facilidad de uso. 64

Figura 30 SeleniumBase: detección de fallos. 65

Figura 31 Testim: facilidad de uso. 66

Figura 32 Testim: detección de fallos**.** 66

Figura 33 Testim: aporte al ciclo de pruebas**.** 67

Figura 34 Mabl: facilidad de uso. 68

Figura 35 Mabl: detección de fallos. 68

Figura 36 Mabl: aporte al ciclo de pruebas. 69

Figura 37 Automatización con supervisión humana 70

Índice de tablas

Tabla 1 Rasgos de las tres herramientas según la literatura revisada. 32

Tabla 2 Operacionalización de las variables dependientes. 38

Tabla 3 Ítems del cuestionario de percepción. 40

Tabla 4 Defectos sembrados en MediaAgend**.** 40

Tabla 5 Defectos sembrados denunciados por condición (1 = detectado). 58

Tabla 6 Impacto del cambio de id. 63

Resumen

La automatización de pruebas permite disminuir el esfuerzo necesario para repetir casos de prueba en aplicaciones web, aunque el uso de herramientas con funciones de inteligencia artificial no asegura por sí solo una mejora completa en la calidad del software. El estudio se desarrolló con un enfoque mixto, predominio cuantitativo, y un diseño descriptivo comparativo aplicado en un entorno de prueba. Para la evaluación se definieron dieciocho casos de prueba funcionales y seis defectos controlados vinculados con autenticación, selección de horarios, duplicidad de citas, códigos de confirmación, cancelación y control de sesiones. SeleniumBase se ejecutó directamente sobre el prototipo, mientras que Testim y Mabl se analizaron mediante condiciones modeladas. Las cuatro condiciones alcanzaron una cobertura funcional del 100 %. La detección de defectos fue del 83,3 % en la ejecución manual, SeleniumBase y Mabl, y del 66,7 % en Testim. El tiempo total del ciclo pasó de 49,92 minutos en la ejecución manual a 5,70 minutos con SeleniumBase, 3,92 minutos en el modelo de Testim y 4,30 minutos en el modelo de Mabl. Los resultados muestran que las automatizaciones reducen de forma importante el tiempo de ejecución y facilita la repetición de las pruebas, donde en conjunto con el diseño de los casos lo que mejora la cobertura y la detección de defectos, la definición correcta de los oráculos de prueba, del mantenimiento de los localizadores y de la revisión humana, mejoran el proceso de pruebas.

*Palabras clave***:** Aplicaciones web, calidad de software, inteligencia artificial, Mabl, pruebas automatizadas, pruebas funcionales, SeleniumBase, Testim.

Abstract

Test automation helps reduce the effort required to repeatedly execute test cases in web applications. However, the use of tools that include artificial intelligence features does not by itself guarantee a complete improvement in software quality. The study followed a mixed approach with quantitative predominance and a descriptive-comparative design applied in a testing environment. Eighteen functional test cases and six controlled defects were defined for the evaluation. These defects were related to authentication, schedule selection, duplicate appointments, confirmation codes, cancellation, and session control. SeleniumBase was executed directly on the prototype, while Testim and Mabl were analyzed through modeled conditions. All four conditions achieved 100% functional coverage. Defect detection reached 83.3% in manual execution, SeleniumBase, and Mabl, while Testim reached 66.7%. The total execution time decreased from 49.92 minutes in the manual condition to 5.70 minutes with SeleniumBase, 3.92 minutes in the Testim model, and 4.30 minutes in the Mabl model.

The results indicate that automation considerably reduces execution time and facilitates the repetition of test cases. The effectiveness of the testing process is also influenced by the design of the test cases, the correct definition of test oracles, locator maintenance, and human review when self-healing mechanisms are involved.

Keywords: Artificial intelligence, functional testing, Mabl, Selenium, software quality, Test automation, Testim, web applications.

Capítulo I

Características Generales del Proyecto

Antecedentes

La calidad del software constituye una preocupación permanente dentro de la ingeniería de software, debido a que los defectos no identificados durante las primeras etapas del desarrollo pueden ocasionar mayores costos de corrección, retrasos y afectaciones para los usuarios. Pressman y Sommerville sostienen que la calidad no debe evaluarse únicamente cuando el producto está terminado, sino que debe construirse y verificarse durante todo su ciclo de vida (Pressman, 2010), (Sommerville, 2011). Se señala que el costo de corregir un error aumenta cuando este se detecta en etapas avanzadas (Basili & Boehm, 2001). Estos planteamientos convirtieron las pruebas en una actividad esencial para comprobar el cumplimiento de los requisitos y reducir el riesgo asociado con la publicación de nuevas versiones.

Inicialmente, muchas organizaciones ejecutaban sus pruebas de forma predominantemente manual. Este enfoque continúa siendo útil para explorar el sistema, revisar aspectos de usabilidad y analizar situaciones que requieren criterio humano. No obstante, presenta limitaciones cuando los mismos casos deben repetirse después de cada modificación. En proyectos desarrollados mediante metodologías ágiles, cada sprint puede incorporar nuevas funcionalidades y cambios sobre componentes existentes, lo que incrementa progresivamente la cantidad de pruebas de regresión. Humble y Farley explican que una estrategia de entrega continua necesita pruebas repetibles y de bajo costo; de lo contrario, la verificación manual puede convertirse en un obstáculo para la frecuencia de las entregas (Humble & Farley, 2010).

La automatización apareció como una alternativa para ejecutar acciones y verificaciones de forma sistemática. Ammann y Offutt plantean técnicas para diseñar casos de prueba mediante particiones de equivalencia, valores límite y modelos de comportamiento, con el fin de evitar que la

automatización se limite únicamente a registrar una secuencia de clics (Ammann & Offutt, 2016). Fewster y Graham señalan además que automatizar puede reducir el esfuerzo requerido en ejecuciones repetitivas, aunque sus beneficios dependen de aspectos como la selección adecuada de los casos, la estabilidad del sistema y el mantenimiento de los scripts (Fewster & Graham, 1999). Por esta razón, una prueba automatizada no debe entenderse como un recurso sin costos, ya que necesita datos apropiados, localizadores estables, resultados esperados y aserciones que permitan determinar si el comportamiento obtenido es correcto.

En las aplicaciones web, uno de los principales problemas de mantenimiento se relaciona con los localizadores utilizados para interactuar con los elementos de la interfaz. Un cambio en el identificador de un botón, en la estructura del DOM o en la posición de un componente puede provocar que un caso deje de ejecutarse, aunque la funcionalidad continúe disponible. Leotta, Stocco, Ricca y Tonella estudiaron esta fragilidad y propusieron estrategias para generar localizadores XPath más resistentes a los cambios (Leotta et al., 2013). En otra investigación, los autores compararon las pruebas programables con los mecanismos de captura y reproducción, y observaron que el enfoque basado en código ofrece mayor control y capacidad de mantenimiento, aunque exige más conocimientos técnicos (Leotta et al., 2013). Donde se determinó que el retorno de la inversión de las herramientas de prueba de interfaces depende del mantenimiento necesario y no únicamente del ahorro obtenido durante la primera ejecución (Dobslaw et al., 2019).

Como solución a dichas dificultades, en los últimos años han surgido plataformas que incorporan funciones presentadas como inteligencia artificial. Entre sus principales características se encuentran los localizadores basados en múltiples atributos, la autocuración o self-healing, la validación visual y la generación asistida de pruebas. Garousi et al. analizaron más de cincuenta herramientas de este tipo e identificaron beneficios vinculados con la reducción del esfuerzo y la capacidad de adaptación ante cambios menores en la interfaz (Garousi et al., 2024). No obstante, también encontraron problemas

relacionados con clasificaciones incorrectas, falsos positivos y dificultades para interpretar determinadas reglas de negocio. Ricca, Marchetto y Stocco obtuvieron resultados similares al revisar la literatura profesional sobre inteligencia artificial aplicada a las pruebas, señalando que estas soluciones han tenido un crecimiento importante, aunque todavía existen pocas comparaciones realizadas bajo condiciones homogéneas (Ricca et al., 2024).

La incorporación de inteligencia artificial tampoco elimina el problema del oráculo de prueba. Barr et al. definen este problema como la dificultad de establecer si el resultado obtenido por el sistema corresponde realmente con el comportamiento esperado (Barr et al., 2015). Una herramienta puede localizar un elemento alternativo y continuar la ejecución, pero esto no garantiza que esté interactuando con el control correcto. En consecuencia, un caso puede aparecer como aprobado a pesar de que la autocuración haya dirigido la prueba hacia un componente diferente. Pysmennyi, Kyslyi y Kleshch identificaron riesgos similares al evaluar agentes de inteligencia artificial aplicados al aseguramiento de la calidad, entre ellos la inestabilidad de las ejecuciones, la escasa explicabilidad y el comportamiento de caja negra (Pysmennyi et al., 2025). Estos antecedentes muestran que la supervisión humana continúa siendo necesaria.

Dentro de este contexto se consideran SeleniumBase, Testim y Mabl. SeleniumBase es un framework programable basado en Selenium WebDriver y pytest que permite controlar el navegador mediante código Python, incorporar aserciones y generar reportes, Testim un generador de pruebas autónomas mediante IA que garantiza la calidad de la aplicación al usar agenten que no requieren de código para validación y Mabl un agente hibrido de consola e interfaz que permite la rápida ejecución de pruebas a través de un agente personalizado.

En Ecuador, la evidencia disponible sobre automatización de pruebas todavía es limitada. Ponce y Naranjo desarrollaron en la Universidad de las Fuerzas Armadas ESPE una evaluación de Selenium aplicada al producto Gestor G5 Trust. Los autores reportaron beneficios en la ejecución de pruebas

funcionales y de regresión, además de un análisis de costo-beneficio favorable (Ponce Guerron & Naranjo Erazo, 2021). Este trabajo constituye uno de los antecedentes institucionales más cercanos a la presente investigación; sin embargo, se concentró en SeleniumBase y no realizó una comparación con plataformas que incorporan mecanismos de inteligencia artificial. Por su parte, Serna señala que las pruebas deben incluirse desde las primeras etapas del desarrollo para evitar incrementos en los tiempos y costos (Serna Montoya, 2013).

Líneas de Investigación

- Sistemas Electrónicos y Computacionales

Sub-Líneas de Investigación

- Sistemas Digitales
- Control de Calidad de software

Área de Influencia

La ausencia de comparativas entre herramientas para control de calidad justifica la evaluación planteada en la investigación. Por dicha razón el estudio recae en MediAgenda, un módulo que reproduce las funciones principales del proceso de agendamiento médico. Sobre el mismo se estableció una matriz común de dieciocho casos y se incorporaron seis defectos controlados. La ejecución manual y Selenium se aplicaron directamente sobre el prototipo, mientras que Testim y Mabl se estudiaron mediante condiciones modeladas por no disponer de licencias operativas. De esta manera, la investigación busca aportar evidencia sobre cobertura funcional, detección de defectos, tiempo de ejecución y percepción de uso, diferenciando claramente los resultados medidos de aquellos obtenidos mediante modelación.

Planteamiento del problema

En la industria del software, los defectos que no se identifican durante las pruebas pueden producir consecuencias económicas y operativas importantes. Basili y Boehm señalan que el costo de corregir un fallo aumenta a medida que este se detecta en etapas más avanzadas del desarrollo (Basili &

Boehm, 2001). Asimismo, Myers, Sandler y Badgett explican que las pruebas no garantizan que un sistema esté completamente libre de errores, pero permiten reducir el riesgo cuando los casos se diseñan de manera adecuada y pueden ejecutarse de forma repetible (Myers et al., 2011). Bertolino añade que las actividades de prueba pueden representar una proporción considerable del costo de desarrollo y que, a pesar de esa inversión, muchas organizaciones todavía cuentan con infraestructuras de prueba insuficientes (Bertolino, 2007). Aunque los costos varían entre organizaciones, estos estudios coinciden en que la detección tardía de defectos continúa siendo un problema relevante.

- **Problema central**

El problema que enfrenta la empresa CONFIAMED SA. Es el control del código que generan, lo que eleva los gastos afectando los presupuestos de producción y afectando su calidad, en este contexto, tener pruebas automatizadas no cubren al 100% la funcionalidad del producto a ser probado, hay casos de pruebas que no pueden ser automatizados, debido a que ciertas herramientas tienen limitaciones o no pueden cubrir ciertos casos de pruebas. En estos casos se siguen ejecutando las pruebas manuales. El costo de adquirir una herramienta es un factor importante al momento de elegir la herramienta, ya que, en muchas ocasiones, son evaluadas por las empresas de desarrollo de software, y al no justificar los costos no son adquiridas.

En otros casos, existen herramientas sin costo, pero con reducidas funcionalidades para poder automatizar las pruebas. Es importante indicar que también influye el tipo de framework, lenguaje de programación, formularios dinámicos, herramientas de desarrollo, que en la mayoría de los casos no son compatibles con estas herramientas de automatización de pruebas.

Justificación, importancia y alcance del proyecto

Las normas ISO/IEC/IEEE 29119 y los lineamientos de ISTQB entienden las pruebas de software como un proceso integrado por actividades de planificación, diseño, ejecución y evaluación. Asimismo, establecen distintos niveles de prueba, entre ellos las pruebas unitarias, además de tipos como las pruebas funcionales y de regresión (ISO/IEC/IEEE, 2013), (ISTQB, 2023). En entornos de producción real existe una clara diferencia entre la creación de pruebas de manera manual y la automatización de estas. Cuando las pruebas dependen únicamente de la intervención humana, cada nueva versión obliga a repetir un conjunto cada vez mayor de casos, lo que puede dificultar el cumplimiento de los tiempos definidos para el sprint (Humble & Farley, 2010).

Fewster y Graham sostienen que la automatización resulta beneficiosa cuando se seleccionan correctamente los casos, se mantiene la suite de pruebas y se consideran los costos reales de implementación y mantenimiento (Fewster & Graham, 1999). Su efectividad también depende del oráculo de prueba, es decir, del criterio empleado para determinar si el resultado obtenido es correcto. Por esta razón, un script que verifica un mensaje o un elemento de la interfaz será confiable únicamente si la aserción utilizada representa de forma adecuada el comportamiento esperado del sistema (Barr et al., 2015).

Como respuesta a las necesidades de regresión, el mercado ha incorporado herramientas que anuncian funciones basadas en inteligencia artificial. Garousi et al. identifican características como la autocuración de localizadores, la validación visual y la generación asistida de pruebas. Sin embargo, también advierten sobre la aparición de falsos positivos y las dificultades de estas herramientas para interpretar el contexto del negocio (Garousi et al., 2024). Ricca et al. confirman el crecimiento de estas soluciones en la práctica profesional, aunque señalan que las evaluaciones disponibles todavía presentan diferencias metodológicas importantes (Ricca et al., 2024). Por tanto, se requieren estudios

que comparen distintas herramientas mediante una matriz común de casos y métricas que incluyan tanto el tiempo de ejecución como el esfuerzo de mantenimiento.

En Ecuador, la adopción de la automatización de pruebas aún enfrenta limitaciones. Ponce y Naranjo, señalaron que la ejecución manual continúa siendo frecuente debido a la complejidad de automatizar y a la dificultad de justificar económicamente la adquisición de determinadas herramientas (Ponce Guerron & Naranjo Erazo, 2021). Aunque este antecedente no constituye un diagnóstico nacional, muestra la necesidad de contar con datos locales sobre tiempos, cobertura y detección de defectos que permitan respaldar las decisiones de automatización.

Serna sostiene que las pruebas deben incorporarse desde las primeras etapas del desarrollo, debido a que su aplicación tardía incrementa los plazos y los costos del producto (Serna Montoya, 2013). Posteriormente, Serna, Martínez y Tamayo calificaron la madurez de la automatización de pruebas como “adolescente”, puesto que su aplicación todavía depende en gran medida de las personas encargadas del proceso y no siempre abarca todo el plan de pruebas (Serna et al., 2017). Esta situación puede observarse en equipos que seleccionan o descartan herramientas como Selenium, Testim o Mabl sin aplicar previamente un protocolo común de evaluación. En consecuencia, las decisiones pueden basarse en demostraciones comerciales o apreciaciones individuales, en lugar de resultados comparables.

Los portales de agendamiento utilizados en ámbitos como salud, seguros y servicios públicos constituyen un escenario pertinente para estudiar este problema. Aunque la reserva de una cita parece un proceso sencillo, incluye dependencias entre especialidades, profesionales, fechas y horarios, además de controles de sesión y códigos de confirmación. Cualquier modificación en la estructura de la interfaz puede afectar los localizadores empleados por las pruebas automatizadas. Al mismo tiempo, el crecimiento de los casos de regresión puede dificultar su ejecución manual dentro de cada sprint.

Humble y Farley explican que la ausencia de una regresión eficiente puede retrasar las entregas de software (Nass et al., 2021). En un sistema de citas, esta situación tiene consecuencias directas: una reserva duplicada, una cancelación que no libera el horario o una consulta sin autenticación pueden afectar el funcionamiento del servicio. En este sentido, la investigación no busca determinar únicamente si existen funciones de inteligencia artificial aplicadas a las pruebas, sino establecer cómo se comportan distintas alternativas de automatización ante un mismo flujo funcional y bajo criterios comparables.

El caso planteado inicialmente corresponde a la función de agendamiento de citas del portal de la empresa patrocinadora. Para evaluarlo se deben considerar la especificación funcional, los casos de prueba, la ejecución manual, la automatización con las tres herramientas seleccionadas y el registro de tiempos, defectos y actividades de repetición de pruebas. Donde, no todos los casos tienen la misma facilidad para ser automatizados y las herramientas presentan diferencias en costo, aprendizaje y capacidad para responder a cambios en la interfaz (Runeson & Host, 2009). También puede ocurrir que el fallo de un script detenga la ejecución de otros casos, mientras que la percepción sobre la facilidad de uso puede ser distinta al tiempo registrado durante las pruebas.

La autocuración de un localizador no garantiza que el resultado funcional sea correcto. La herramienta puede encontrar un elemento alternativo y continuar la ejecución, pero todavía es necesario comprobar el oráculo del caso; por ejemplo, verificar que el código mostrado en la confirmación coincida con el registrado en el listado de citas (Alegroth et al., 2015). Por esta razón, la comparación considera no solo el tiempo de ejecución, sino también la detección de defectos, la cobertura funcional y el comportamiento de las herramientas ante cambios en la interfaz.

Objetivo general

- Evaluar el impacto de las pruebas automatizadas con inteligencia artificial en la calidad del desarrollo de aplicaciones web: Comparativa entre SeleniumBase, Testim y Mabl.

Objetivos específicos

- Caracterizar las funcionalidades, ventajas y limitaciones de las herramientas SELENIUMBASE, TESTIM y MABL, en el contexto de la automatización de pruebas con inteligencia artificial.
- Evaluar el efecto de la automatización de pruebas en la cobertura funcional, la detección de defectos y el tiempo de ejecución, mediante su aplicación sobre el flujo de agendamiento considerado en el estudio.
- Analizar la valoración relacionada con la efectividad, facilidad de uso y beneficios asociados con las herramientas consideradas en la investigación.

Formulación de la hipótesis

La aplicación de herramientas de automatización de pruebas puede contribuir a mejorar determinados aspectos del proceso de pruebas de una aplicación web, especialmente en la reducción del tiempo de ejecución y en la repetición de los casos, aunque su efecto sobre la cobertura y la detección de defectos depende del diseño de la suite y de las características de cada herramienta.

Categorización de las variables de investigación

- **Variable independiente:** Uso de herramientas de pruebas automatizadas con inteligencia artificial: SeleniumBase, Testim y Mabl).
- **Variable dependiente:**
    - Cobertura de pruebas: Grado en el que las pruebas automatizadas ejercen control sobre las funcionalidades del sistema, medido en función del porcentaje de líneas de código o módulos del portal que son evaluados por los casos de prueba.
    - Tasa de errores/defectos: Cantidad de errores o defectos identificados en el sistema durante la ejecución de las pruebas automatizadas.

Capítulo II

Marco Teórico

Calidad de software y modelos de referencia

La discusión sobre calidad de software dejó hace tiempo de reducirse a un simple adjetivo. ISO/IEC 9126 organizó la calidad del producto en características funcionales y durante varios años fue uno de los referentes más utilizados dentro de la ingeniería de software (ISO/IEC, 2011). A dicho cambio la ISO/IEC 25010 reforzo, actualizó e integró dentro de la familia SQuaRE dichos indicadores (Romano et al., 2021). Donde, la calidad no debe revisarse únicamente al final del desarrollo, sino que debe construirse y verificarse durante todo el ciclo de vida del software (Pressman, 2010).

En paralelo, CMMI se enfoca principalmente en la madurez de los procesos y no en las características propias del producto (Chrissis et al., 2011). Aunque ambos enfoques aparecían relacionados en el plan de tesis, en este trabajo cumplen funciones diferentes. ISO/IEC 25010 permite analizar lo que se espera del portal, por ejemplo, que una cita se registre correctamente, que la sesión se mantenga y que la suite de pruebas pueda mantenerse cuando cambia algún elemento de la interfaz. CMMI, en cambio, sirve como referencia para comprender por qué un área de calidad con un nivel de madurez todavía “adolescente”, como lo plantean Serna, Martínez y Tamayo (ISO/IEC, 2011), puede seguir dependiendo en gran medida del trabajo manual para impulsar la automatización.

Por esta razón, un nivel bajo de madurez no se resuelve únicamente comprando una herramienta o una licencia. También se refleja en la existencia de protocolos, métricas, procedimientos de seguimiento y personal capaz de interpretar los resultados obtenidos.

Las métricas que aparecen con frecuencia en esta literatura incluyen la densidad de defectos, la cobertura de pruebas, el tiempo de ciclo y el costo de mantenimiento de los scripts. Basili y Boehm ya advertían que los defectos detectados en etapas tardías del desarrollo tienen un costo mayor (ISO/IEC, 2016). Myers, Sandler y Badgett también recuerdan una limitación fundamental de las pruebas de

software. Una prueba no demuestra que el sistema esté libre de errores, sino que permite comprobar que, bajo determinados casos, ciertos errores no aparecieron (ISO/IEC, 2016). ISO/IEC 25023 complementa este enfoque al traducir las características de ISO/IEC 25010 en medidas más operativas, como la completitud y la corrección funcional (ISO/IEC, 2016).

Pruebas, agilidad y entrega continua

ISTQB e ISO/IEC/IEEE 29119 plantean las pruebas de software como un proceso compuesto por actividades de planificación, análisis, diseño, implementación, ejecución y reporte. También distinguen diversos niveles de pruebas, además de diferentes tipos de pruebas, entre ellas las funcionales y las de regresión (ISO/IEC, 2016), (ISO/IEC, 2011). IEEE 829, aunque corresponde a un estándar anterior, continúa siendo una referencia importante para comprender qué información debe documentarse dentro de un plan de pruebas (ISO/IEC, 2001).

Bertolino organizó buena parte de la agenda de investigación alrededor de problemas como los criterios de prueba, los oráculos, los costos y el retesting después de los cambios. Su planteamiento deja claro que la automatización completa debe entenderse como un objetivo de investigación y no como una capacidad garantizada por una herramienta comercial (ISO/IEC, 2016).

En las metodologías ágiles, el trabajo se desarrolla mediante iteraciones cortas. En Scrum, esta dinámica se organiza principalmente a través de los sprints (Mesbah, van Deursen, & Roest, 2012). Crispin y Gregory señalan que las pruebas forman parte de este proceso y dejan de realizarse únicamente al finalizar el desarrollo, ya que se integran al trabajo correspondiente a cada historia de usuario (Crispin & Gregory, 2009). Humble y Farley relacionan esta práctica con la entrega continua y explican que, si las pruebas de regresión no pueden ejecutarse de forma automática y repetible, la integración y la entrega continua reducen buena parte de su utilidad práctica (Humble & Farley, 2010).

En términos operativos, esto puede traducirse en un job capaz de iniciar el navegador, ejecutar la suite y publicar un reporte. SeleniumBase se adapta a este escenario porque se integra con pytest.

Testim y Mabl también ofrecen funcionalidades orientadas a entornos de nube y procesos de integración continua. Garousi et al. y Ricca et al. identifican estas capacidades como características frecuentes de las plataformas de testing apoyadas en inteligencia artificial, pero no las presentan como una solución automática para todos los problemas de mantenimiento (Garousi & Mantyla, 2016), (Garousi et al., 2024)

MediAgenda se ejecuta de forma local mediante el puerto 5000. No se implementó ni se instrumentó un pipeline real de la empresa patrocinadora, por lo que las conclusiones no describen el comportamiento de las herramientas dentro de su infraestructura. Más adelante, especialmente en las recomendaciones, sí puede analizarse cómo la fragilidad de una suite puede incrementar el costo de mantener una ejecución automatizada dentro de un proceso de integración continua (Wohlin et al., 2012).

El trabajo se ubica específicamente en el nivel de pruebas de sistema, con énfasis en pruebas funcionales y una parte de regresión cuando se introducen cambios controlados, como la modificación de identificadores de elementos de la interfaz. No se realizan pruebas unitarias sobre el backend ni pruebas directas sobre API. Esto responde al objetivo central del estudio, que consiste en comparar herramientas orientadas a la automatización de una interfaz web.

Ammann y Offutt plantean técnicas como la partición de equivalencia, el análisis de valores frontera y el uso de modelos para diseñar los casos de prueba de manera ordenada. El objetivo es evitar que una suite automatizada se limite a reproducir una secuencia de clics (Ammann & Offutt, 2016). En MediAgenda, este enfoque se aplica mediante casos positivos, como un inicio de sesión válido o la creación correcta de una cita, y casos negativos, entre ellos una contraseña incorrecta, una fecha vencida, un cupo ocupado o el intento de acceder directamente a una URL sin mantener una sesión activa.

Serna también insiste, desde el contexto hispano, en incorporar las actividades de prueba

desde las primeras etapas relacionadas con los requisitos, ya que dejar estas tareas únicamente para el final puede aumentar los tiempos y costos del proyecto (Kaner et al., 2002). En este estudio, las actividades propuestas por ISTQB se traducen de manera concreta. El análisis y el diseño corresponden a la matriz de dieciocho casos. La implementación comprende los scripts desarrollados en SeleniumBase y las configuraciones realizadas en Testim y Mabl.

La ejecución manual no desaparece con la automatización. Sigue siendo necesaria para actividades como la exploración del sistema, la evaluación de usabilidad y la revisión de escenarios poco frecuentes. El problema aparece cuando, con cada nueva versión, la cantidad de casos de regresión aumenta y el retesting comienza a consumir una parte importante del tiempo disponible.

Se sitúa la automatización precisamente en aquellos escenarios que son repetibles, siempre que se seleccionen correctamente los casos que vale la pena automatizar y se considere desde el inicio el costo de mantenimiento de las pruebas (Fewster & Graham, 1999).

Una herramienta que use inteligencia artificial para reparar automáticamente un locator y continuar la ejecución no soluciona por sí sola este problema. Si no se comprueba que la aserción sigue evaluando el comportamiento correcto, la ejecución puede continuar sin asegurar que la verificación realizada siga siendo válida.

Inteligencia artificial en la automatización de pruebas

A partir de 2020, el mercado de herramientas de testing empezó a incorporar con mayor fuerza funciones como el self-healing, los localizadores inteligentes, las pruebas visuales y la generación asistida de casos. Garousi et al. realizaron un análisis sistemático de más de cincuenta herramientas y posteriormente evaluaron dos de ellas sobre software de código abierto. Aunque Testim y Mabl no formaron parte de esa evaluación empírica, las herramientas analizadas pertenecen al mismo tipo de soluciones. Los autores encontraron ventajas como una ejecución más rápida y una reducción del esfuerzo de mantenimiento, pero también identificaron problemas como clasificaciones incorrectas, una

comprensión limitada del dominio y un costo adicional de supervisión, ya que el tester debe revisar las decisiones tomadas por la inteligencia artificial (Hernandez Sampieri et al., 2014). Ricca, Marchetto y Stocco analizaron durante varios años literatura gris relacionada con este tipo de herramientas y confirmaron que la inteligencia artificial aplicada al testing ya tiene una presencia importante en la práctica profesional. Sin embargo, también señalan que todavía existe poca evidencia empírica obtenida bajo condiciones comparables (Ricca et al., 2024). Pysmennyi, Kyslyi y Kleshch evaluaron agentes de inteligencia artificial aplicados al aseguramiento de calidad, especialmente en tareas como la generación de casos y escenarios de extremo a extremo, y reportaron un 8,3 % de ejecuciones inestables en su prueba de concepto. Además, identificaron limitaciones relacionadas con el funcionamiento de caja negra, la falta de explicabilidad y la posibilidad de que el modelo modifique un caso para hacerlo coincidir con el resultado esperado, en lugar de detectar correctamente el problema (Mesbah, van Deursen, & Lenselink, 2012).

Expertos del área concluyen que la automatización total de las pruebas entre los objetivos todavía difíciles de alcanzar dentro de la disciplina (Bertolino, 2007). En dicho caso, el resultado puede ocultar un defecto y generar un problema relacionado con el oráculo de prueba (Barr et al., 2015). Por ese motivo, esta investigación no utiliza el porcentaje de autocuración anunciado por los proveedores como una medida directa de efectividad, sino que observa cuántos casos se rompen y cuántos son identificados por la herramienta como reparados, comparándolos con SeleniumBase, donde la corrección del localizador se realiza de forma explícita por quien desarrolla el script.

Conviene separar tres usos de la inteligencia artificial que suelen presentarse como si fueran uno solo. El primero corresponde a los localizadores que combinan varios atributos y pueden ajustarse a partir de ejecuciones anteriores, como ocurre en Testim y Mabl, y de otra forma en Healenium sobre Selenium. El segundo se relaciona con la comparación visual de pantallas. El tercero abarca la generación de casos de prueba o datos mediante modelos de lenguaje, línea en la que se encuentra el

trabajo de Pysmennyi et al. (Pysmennyi et al., 2025). Esta investigación se concentra principalmente en el primer uso y, en menor medida, en la validación visual disponible en Mabl. La matriz de pruebas no se genera mediante un LLM; los dieciocho casos se definen a partir de los criterios de aceptación establecidos para el estudio.

SeleniumBase

Selenium WebDriver se ha consolidado como una de las principales opciones para controlar el navegador mediante código. La literatura sobre pruebas web, incluidos los trabajos de Leotta y la tesis de Ponce y Naranjo, se refiere principalmente a Selenium y no siempre de forma específica a SeleniumBase (Presler-Marshall et al., 2019), funciona como una capa en Python construida sobre ese ecosistema e incorpora facilidades para las esperas, aserciones de página, generación de reportes y escritura de scripts más breves. En esta investigación se utiliza como representante de la automatización programable y de código abierto, sin depender de una licencia SaaS.

La principal ventaja es que el caso de prueba se mantiene como código, puede versionarse, revisarse e integrarse con pytest y con un pipeline (Leotta et al., 2013). La desventaja aparece cuando los localizadores son poco estables, porque la suite empieza a deteriorarse con los cambios de la aplicación (Leotta et al., 2016), (Leotta et al., 2013). En un equipo que no cuenta con personal con experiencia en Python, esa ventaja puede convertirse en una barrera de adopción. Esta investigación compara precisamente esa dificultad con la facilidad de uso que prometen las plataformas low-code.

Testim

Testim aparece entre las herramientas con funciones de inteligencia artificial revisadas por Garousi et al. y Ricca et al. Se presenta como una plataforma low-code que permite grabar flujos y utilizar localizadores construidos a partir de varios atributos, con capacidad de ajustarse entre ejecuciones (Garousi & Mantyla, 2016), (Garousi et al., 2024). Su principal propuesta es reducir la necesidad de programación y disminuir el mantenimiento cuando se producen cambios menores en el

DOM. El riesgo está en que el tester no siempre puede conocer con claridad por qué la herramienta seleccionó un determinado nodo, lo que puede generar falsos positivos o reparaciones automáticas incorrectas que afecten los resultados. Con esas limitaciones, Testim puede describirse mediante un perfil concreto: creación rápida de pruebas, uso de localizadores construidos con varios atributos, capacidad parcial de self-healing y posibilidad de generar falsos positivos cuando el DOM cambia durante la ejecución. En MediAgenda, este comportamiento puede observarse en el selector de especialidad, que vuelve a cargar parte de la interfaz. Ricca et al. encontraron que la literatura gris presenta más demostraciones de estas herramientas que comparaciones realizadas bajo condiciones equivalentes (Ricca et al., 2024). Para evitar atribuir capacidades no comprobadas, el modelo utilizado en este estudio no toma porcentajes de autocuración publicados con fines comerciales. En su lugar, se definen tres casos que utilizan el botón Confirmar, de los cuales dos se consideran autocurados y uno queda marcado para revisión, siguiendo comportamientos ya reportados por Garousi et al. en herramientas de características similares (Garousi et al., 2024).

Mabl

Plataforma que cuenta con integración de pipeline y el uso de una plataforma que no requiere de código. La prueba visual añade un oráculo distinto al del DOM: se compara la pantalla, no solo el id. Eso puede cazar un código de confirmación mal pintado que un assert flojo no ve, y también puede disparar alertas cuando cambia un margen de CSS. Pysmennyi et al. y Garousi et al. coinciden en que hace falta supervisión (Garousi et al., 2024), (Leotta et al., 2013). Misma condición que Testim: sin licencia, métricas modeladas, misma matriz, origen declarado en cada tabla.

Las características de cada una de las herramientas propuestas en dicha investigación se pueden establecer de mejor manera como se muestra en la Tabla 1.

Tabla 1

Rasgos de las tres herramientas según la literatura revisada.

Criterio	SeleniumBase	Testim	Mabl
Base técnica	Capa Python sobre Selenium WebDriver	Plataforma low-code con localizadores de varios atributos	Plataforma low-code/codeless con autocuración y prueba visual
Cómo se crea el caso	Script (pytest / API de SeleniumBase)	Grabación y ajuste en la herramienta	Grabación y flujo en la nube
IA	No es el foco; la robustez depende de quien programa	Localizadores que aprenden de ejecuciones	Multiples atributos, corrección grafica
Licencia	Código abierto	SaaS de pago	SaaS de pago

Documentación y evidencia

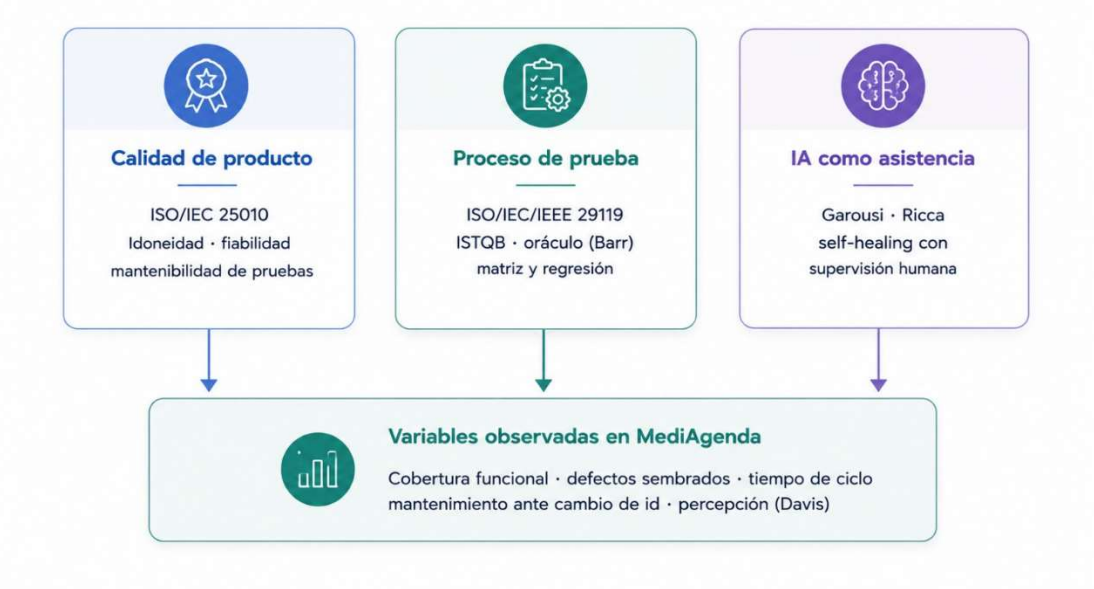
En este estudio, las actividades propuestas por ISTQB se aplican de la siguiente manera. El análisis y el diseño dieron lugar a los dieciocho casos de prueba y a los seis defectos sembrados. La implementación comprende los scripts desarrollados con SeleniumBase y las filas modeladas para Testim y Mabl. La ejecución queda registrada en las cuatro hojas de Excel, mientras que el cierre reúne las figuras del Capítulo IV y las capturas obtenidas del prototipo. Crispin y Gregory plantean que la documentación debe ser útil dentro del trabajo de cada sprint y no convertirse en información que el equipo termina sin consultar (Crispin & Gregory, 2009). Serna también señala la necesidad de incorporar las pruebas desde los requisitos (Serna et al., 2017). En MediAgenda, estos requisitos se representan mediante criterios de aceptación como el inicio de sesión, la disponibilidad de cupos, el código de confirmación y la liberación del horario después de cancelar una cita, los cuales se utilizan como oráculos de prueba.

Leotta et al. (2013) compararon captura-reproducción con pruebas programables mientras la

aplicación evolucionaba (Leotta et al., 2016). La grabación ganaba en la primera autoría. Lo programable ganaba cuando se movía la interfaz. En el 2016 se mostró que un locator se puede endurecer sin comprar una nube: menos fragilidad que el XPath absoluto y que los localizadores de Selenium IDE (Leotta et al., 2013). Se midió Selenium en un producto ecuatoriano y aun así vieron que reejecutar salía más barato que repetir a mano (Ponce Guerron & Naranjo Erazo, 2021). Lo que sigue faltando es poner esos dos precios sobre el mismo flujo de citas, con una herramienta de IA en la misma tabla.ISO/IEC 25023 baja el 25010 a medidas: completitud funcional, corrección y pertinencia funcionales (ISO/IEC, 2016) como se ve en la figura 1.

Figura 1

Postura teórica del estudio.



Capítulo III

Metodología

Este capítulo explica cómo se dará respuesta a las tres preguntas planteadas en el plan. No se realiza trabajo de campo dentro de la empresa patrocinadora, ya que el objeto de estudio es el prototipo MediAgenda, perteneciente a la misma clase de sistema, es decir, un sistema de agendamiento web. Para la evaluación se utiliza una matriz de casos común aplicada bajo cuatro condiciones de ejecución. El componente cuantitativo mide tiempos, veredictos y defectos detectados, mientras que el componente cualitativo tiene un enfoque principalmente documental y se utiliza para caracterizar las herramientas analizadas. La percepción se recoge mediante un cuestionario con escala Likert construido a partir de Davis y aplicado a perfiles vinculados con el prototipo, no a empleados de una aseguradora.

Tipo y diseño de investigación.

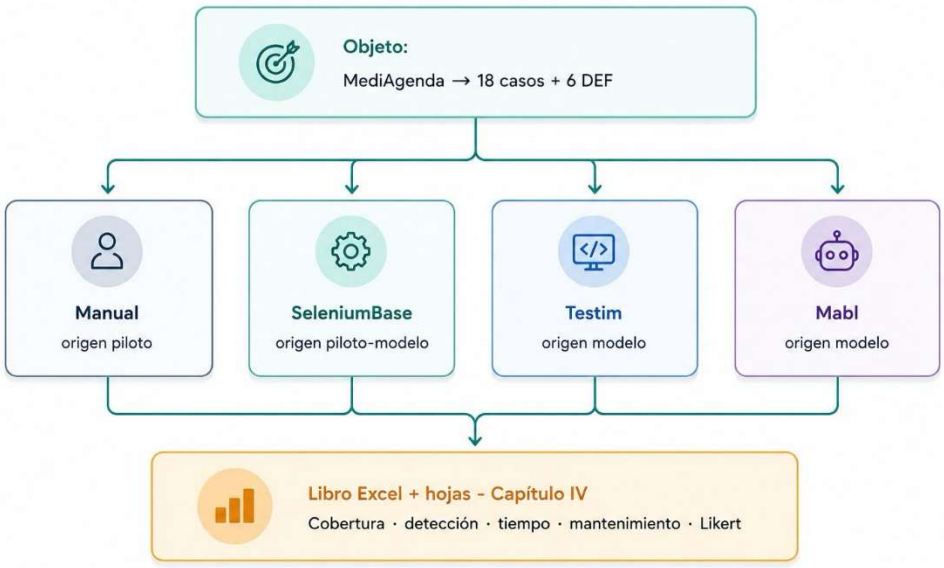
Según su alcance, el trabajo es descriptivo y comparativo. Se describe el comportamiento de cada condición durante las pruebas del agendamiento y luego se comparan los resultados obtenidos. También incluye un componente explicativo limitado, ya que se modifica la herramienta utilizada como variable independiente y se observan cambios en la cobertura, los defectos detectados y el tiempo como variables dependientes. Hernández Sampieri et al. señalan que existe experimentación cuando el investigador interviene sobre las condiciones estudiadas (Hernandez Sampieri et al., 2014). En este trabajo, la intervención se realiza sobre el prototipo mediante la activación de defectos sembrados, la ejecución de la matriz y el cambio del identificador del botón Confirmar. No existe asignación aleatoria de personas a grupos, puesto que el objeto sometido a prueba es el sistema. El diseño se considera experimental sobre el artefacto y de caso único.

Se define el estudio de caso como una investigación empírica orientada al análisis de un fenómeno contemporáneo cuando sus límites con las condiciones que lo rodean no están

completamente definidos (Yin, 2018). En esta investigación, el fenómeno estudiado corresponde al impacto de las tres herramientas de automatización sobre el portal de citas MediAgend. El caso único incorpora varias unidades de análisis, representadas por los dieciocho casos de prueba, los seis defectos sembrados y las cuatro condiciones evaluadas (Yin, 2018) como se muestra en la figura 2.

Figura 2

Diseño experimental del caso.



Según el tiempo de evaluación, la ejecución de la matriz es transversal porque se realiza durante un ciclo definido. La evaluación del mantenimiento incorpora un segundo momento, en el cual se repite el mismo conjunto de casos después de modificar un identificador. Esto no constituye un seguimiento longitudinal durante varios meses. Según la fuente de información, el objetivo 1 corresponde a la revisión documental, mientras que los objetivos 2 y 3 se desarrollan mediante el trabajo realizado con el prototipo.

Unidad de análisis y elementos evaluados

La unidad de análisis corresponde al prototipo MediAgend, específicamente al flujo funcional relacionado con autenticación, reserva, confirmación, consulta y cancelación de citas. La evaluación se

desarrolla sobre este entorno de prueba y no sobre un sistema en producción de la empresa patrocinadora.

Dentro del prototipo se definieron dieciocho casos de prueba que representan los principales criterios de aceptación del proceso de agendamiento. Estos casos incluyen escenarios positivos y negativos y constituyen la base utilizada para comparar las diferentes condiciones consideradas en el estudio.

Figura 3

Flujo funcional del afiliado.



Se seleccionaron escenarios orientados a cubrir los criterios de aceptación del proceso de agendamiento, incluyendo casos positivos y negativos hasta completar dieciocho ítems. Este criterio se relaciona con técnicas de diseño basadas en particiones de equivalencia y análisis de valores frontera (Ammann & Offutt, 2016). De manera similar, los seis defectos incorporados al prototipo fueron definidos de forma controlada para evaluar la capacidad de detección de cada condición analizada (Yin, 2018).

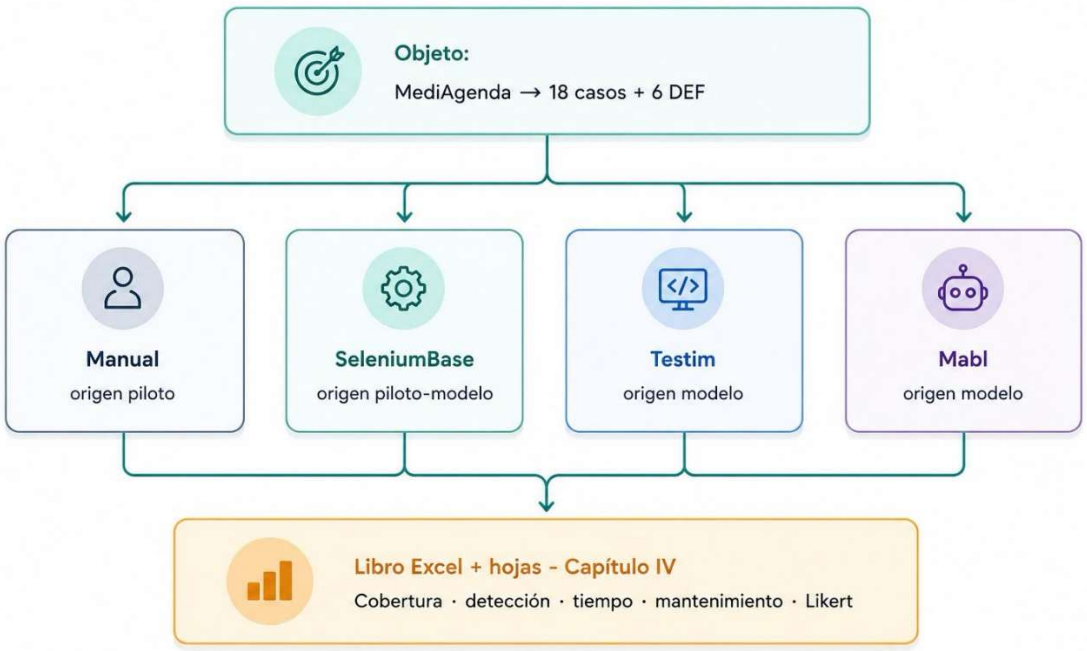
Para el componente de percepción, se trabajó con ocho participantes pertenecientes a la

empresa y vinculados con las actividades de desarrollo y aseguramiento de calidad. El grupo estuvo conformado por tres desarrolladores, tres profesionales de QA y dos responsables de liderazgo. Debido a que estos participantes corresponden al personal disponible dentro del área relacionada con el estudio, se consideró la totalidad de los integrantes identificados para esta evaluación.

La Figura 4 fija cómo está armado MediAgend, qué recorre el afiliado y cómo se parte ese objeto en cuatro condiciones que alimentan un solo libro Excel.

Figura 4

Flujo funcional del afiliado.



Operacionalización de variables

La variable independiente es la condición de ejecución: manual, SeleniumBase, Testim, Mabl. Manual es la línea base. SeleniumBase se ejecuta contra el prototipo en local. Testim y Mabl, sin licencia SaaS, se modelan sobre la misma matriz; cada fila del Excel lleva el campo origen (piloto o modelo) como se ve en la tabla 2.

Tabla 2

Operacionalización de las variables dependientes.

Variable	Indicador	Instrumento	Escala
Cobertura funcional	Casos con veredicto / 18	Matriz de ejecución	Porcentaje
Detección de defectos	DEF-xx denunciados / 6 sembrados	Bitácora de defectos	Conteo y porcentaje
Tiempo de ciclo	Suma de duraciones de los 18 casos	Cronómetro / log de la herramienta	Minutos
Mantenimiento	Casos rotos tras cambiar *btn-confirmar*	Hoja de mantenimiento	Conteo

Técnicas e instrumentos

Revisión documental

La revisión documental se utilizó principalmente para caracterizar las herramientas SeleniumBase, Testim y Mabl, así como para establecer los criterios técnicos empleados durante su comparación. Para ello se consultaron normas relacionadas con calidad y pruebas de software, entre ellas ISO/IEC 25010, ISO/IEC 25023 e ISO/IEC/IEEE 29119, además de bibliografía especializada y artículos científicos vinculados con automatización de pruebas, mantenibilidad de scripts y uso de inteligencia artificial en actividades de testing (ISO/IEC/IEEE, 2013).

Observación sistemática de la ejecución

El instrumento es la matriz de dieciocho casos (identificador, módulo, tipo, resultado esperado, veredicto, tiempo, nota). IEEE 829 e ISTQB piden que el caso deje rastro (IEEE, 2008), (ISTQB, 2023). La ejecución manual se cronometra sobre el prototipo con SEED_DEFECTS=1.

Siembra de defectos y cambio de interfaz

Para evaluar la capacidad de detección de las distintas condiciones de prueba, se incorporaron seis defectos controlados dentro del prototipo MediAgend, como se muestra en la figura 5. Estos defectos estuvieron relacionados con:

- Autenticación
- Horarios vencidos
- Duplicidad de citas
- Códigos de confirmación incorrectos

Cancelaciones

Figura 5

Inserción de casos.

```
def main():
    threading.Thread(target=start_server, daemon=True).start()
    time.sleep(1.6)

    from seleniumbase import SB

    with SB(headless=True, guest=True) as sb:
        sb.set_window_size(1280, 840)

        sb.open(f"{URL}/login")
        shot(sb, "mod_01_login")

        sb.type("#email", "paciente1@mediagenda.test")
        sb.type("#password", "mala.clave")
        sb.click("#btn-ingresar")
        shot(sb, "pru_02_login_clave_incorrecta")

        sb.open(f"{URL}/login")
        sb.type("#email", "paciente1@mediagenda.test")
        sb.click("#btn-ingresar")
        shot(sb, "pru_04_def01_clave_vacia")

        sb.open(f"{URL}/salir")
        sb.open(f"{URL}/login")
        sb.type("#email", "paciente1@mediagenda.test")
        sb.type("#password", "Prueba.123")
        sb.click("#btn-ingresar")
        sb.wait_for_text("Hola, Ana Pérez")
        shot(sb, "mod_02_panel")
        shot(sb, "pru_01_login_valido")
```

Cuestionario de percepción

Técnica: encuesta estructurada. Instrumento: nueve ítems Likert (1 = muy en desacuerdo, 5 = muy de acuerdo), tres por herramienta (facilidad, efectividad, beneficio), más el rol. Davis suministra los dos primeros constructos (Davis, 1989); el tercero (beneficio) se añade para cubrir el enunciado del

objetivo 3. No hay ítems de marketing. El cuestionario no se aplicó a una empresa. Los ítems son el protocolo de percepción del prototipo (Ver tabla 3).

Tabla 3

Ítems del cuestionario de percepción.

Código	Herramienta	Ítem	Código
I-01	SeleniumBase	Es fácil escribir y mantener los casos en esta herramienta	I-01
I-02	SeleniumBase	Resulta efectiva para detectar fallos del agendamiento	I-02
I-03	SeleniumBase	Su uso beneficia el ciclo de pruebas del equipo	I-03
I-04	Testim	Es fácil crear y mantener los casos en esta herramienta	I-04
I-05	Testim	Resulta efectiva para detectar fallos del agendamiento	I-05
I-06	Testim	Su uso beneficia el ciclo de pruebas del equipo	I-06
I-07	Mabl	Es fácil crear y mantener los casos en esta herramienta	I-07
I-08	Mabl	Resulta efectiva para detectar fallos del agendamiento	I-08
I-09	Mabl	Su uso beneficia el ciclo de pruebas del equipo	I-09

Se pudo añadió una seria de defectos que permitan comprobar de mejor manera la gestión del módulo con cada una de las herramientas, los mismos detallados en la tabla 4.

Tabla 4

Defectos sembrados en MediaAgend.

ID	Fallo	Caso detector previsto	ID
DEF-01	Acepta clave vacía si el correo existe	CP-04	DEF-01
DEF-02	Permite hoy con hora ya transcurrida	CP-08	DEF-02
DEF-03	Doble cita del paciente a la misma hora	CP-14	DEF-03
DEF-04	Código en pantalla distinto al de la lista	CP-10	DEF-04

Adicional, se realizó un cuestionario de percepción a fin de definir como recibe el usuario el uso de la herramienta, siendo 1 = muy en desacuerdo y 5 = muy de acuerdo.

Procedimiento

Se desarrolló de manera secuencial con el propósito de mantener las mismas condiciones durante la evaluación. En una primera etapa se preparó el prototipo MediAgenda y se definieron los datos necesarios para la ejecución de las pruebas. Posteriormente se elaboró la matriz de dieciocho casos de prueba y se establecieron los resultados esperados para cada uno de ellos.

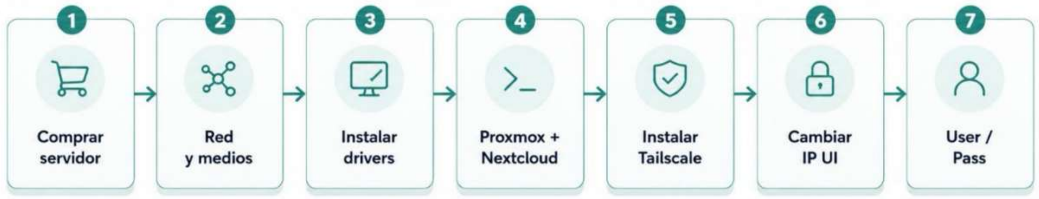
Una vez definida la matriz, se incorporaron los seis defectos controlados y se realizó la ejecución manual de los casos de prueba, registrando el tiempo empleado, el veredicto y las observaciones correspondientes. Después se ejecutaron los mismos casos mediante SeleniumBase, Testim y Mabl, conservando los criterios de evaluación establecidos previamente.

Finalizadas las ejecuciones iniciales, se realizó una modificación controlada en el identificador del botón Confirmar. A partir de este cambio se repitieron los casos afectados para observar el comportamiento de las herramientas frente a modificaciones en la interfaz y registrar los ajustes requeridos durante el mantenimiento de las pruebas.

Como parte del componente de percepción, se aplicó el cuestionario a ocho integrantes de la empresa vinculados con actividades de desarrollo, aseguramiento de calidad y liderazgo. Finalmente, la información obtenida mediante las ejecuciones y el cuestionario fue organizada para calcular los indicadores establecidos y elaborar las tablas y figuras utilizadas en el análisis de resultados, siguiendo el flujo de ejecución que muestra la figura 6.

Figura 6

Procedimiento de laboratorio en siete bloques.

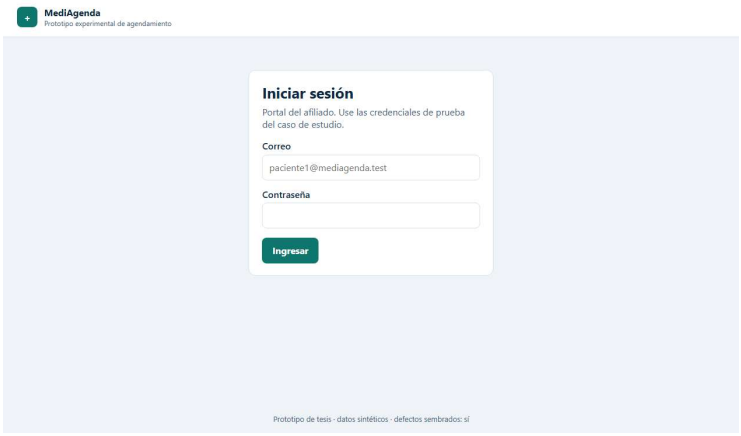


El sistema bajo prueba

IEEE 829 pide evidencia de qué se ejecutó (IEEE, 2008). Kaner et al. piden un oráculo que se pueda ver (Kaner et al., 2002). Las Figuras 7 a 11 son el módulo MediAgend con SEED_DEFECTS=1, datos sintéticos, pie de página que lo declara. No son pantallas de la empresa patrocinadora.

Figura 7

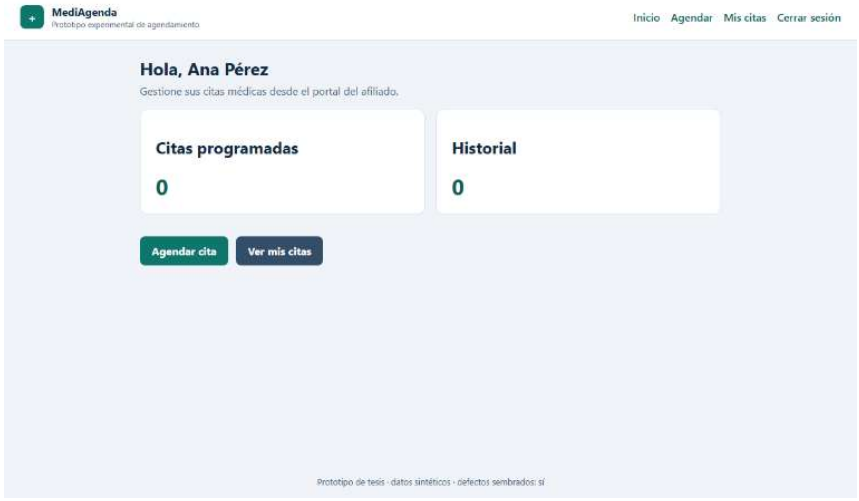
Inicio de sesión.

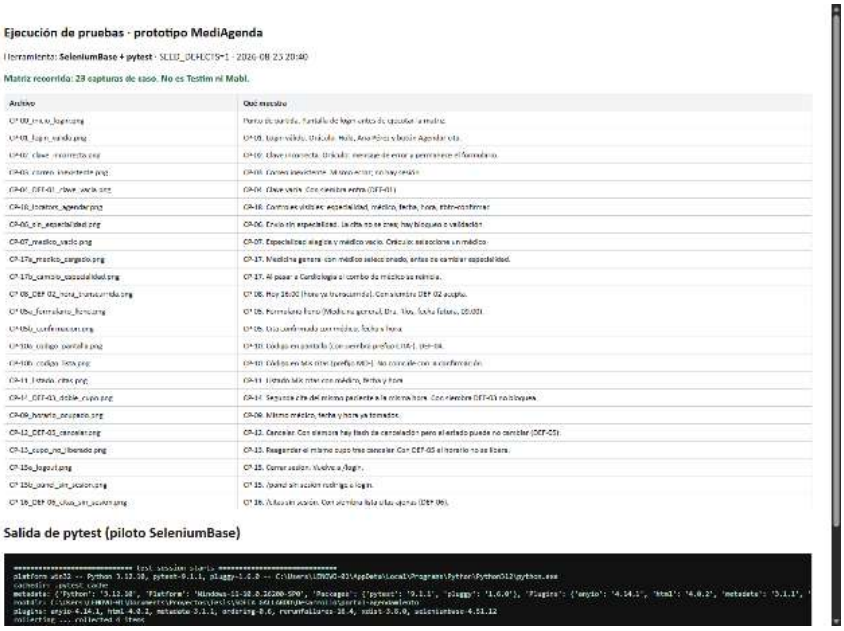


Se creo un usuario de prueba para poder acceder a las funciones del sistema y agendar una cita como se ve en la figura 8.

Figura 8

Login Valido y ejecución de prueba.

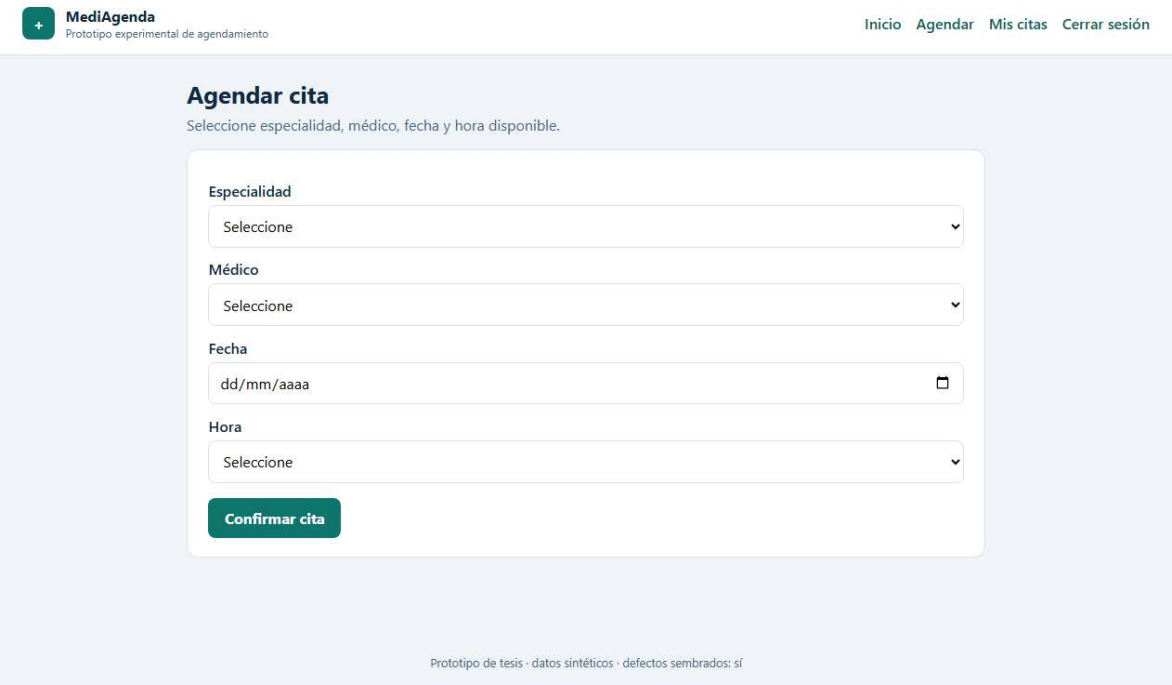


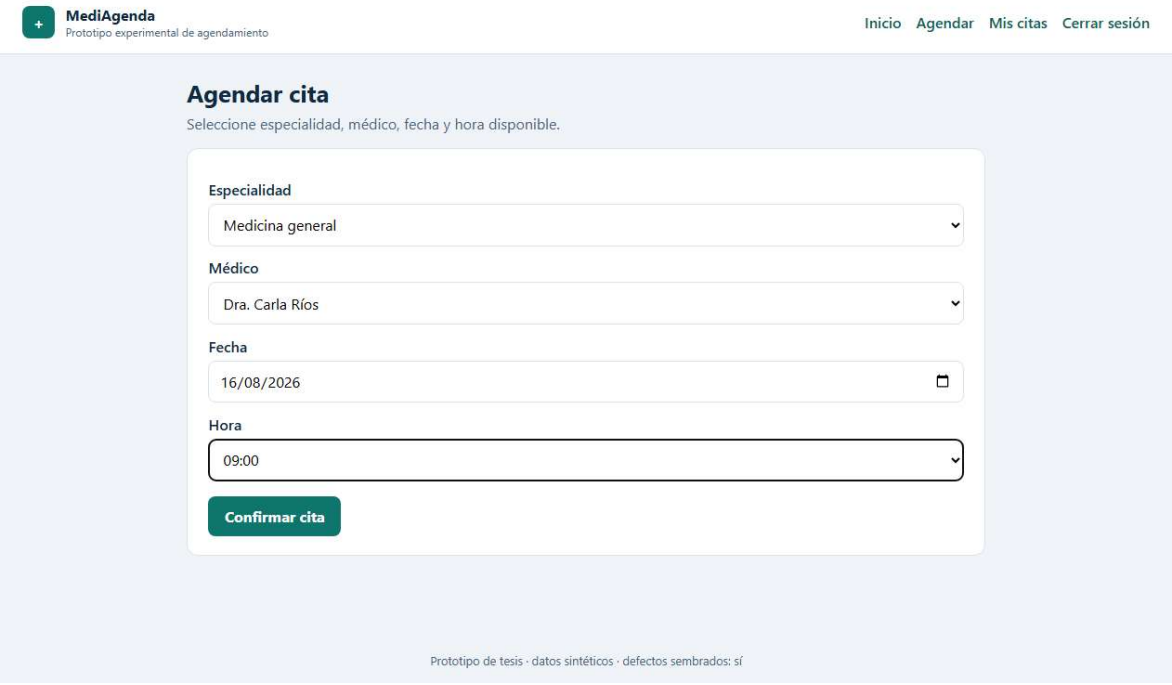


El botón Confirmar lleva el id btn-confirmar, el mismo que se renombra en el escenario de mantenimiento, lo que permite el proceso de agendamiento como se ve en la figura 9.

Figura 9

Formulario de agendamiento.

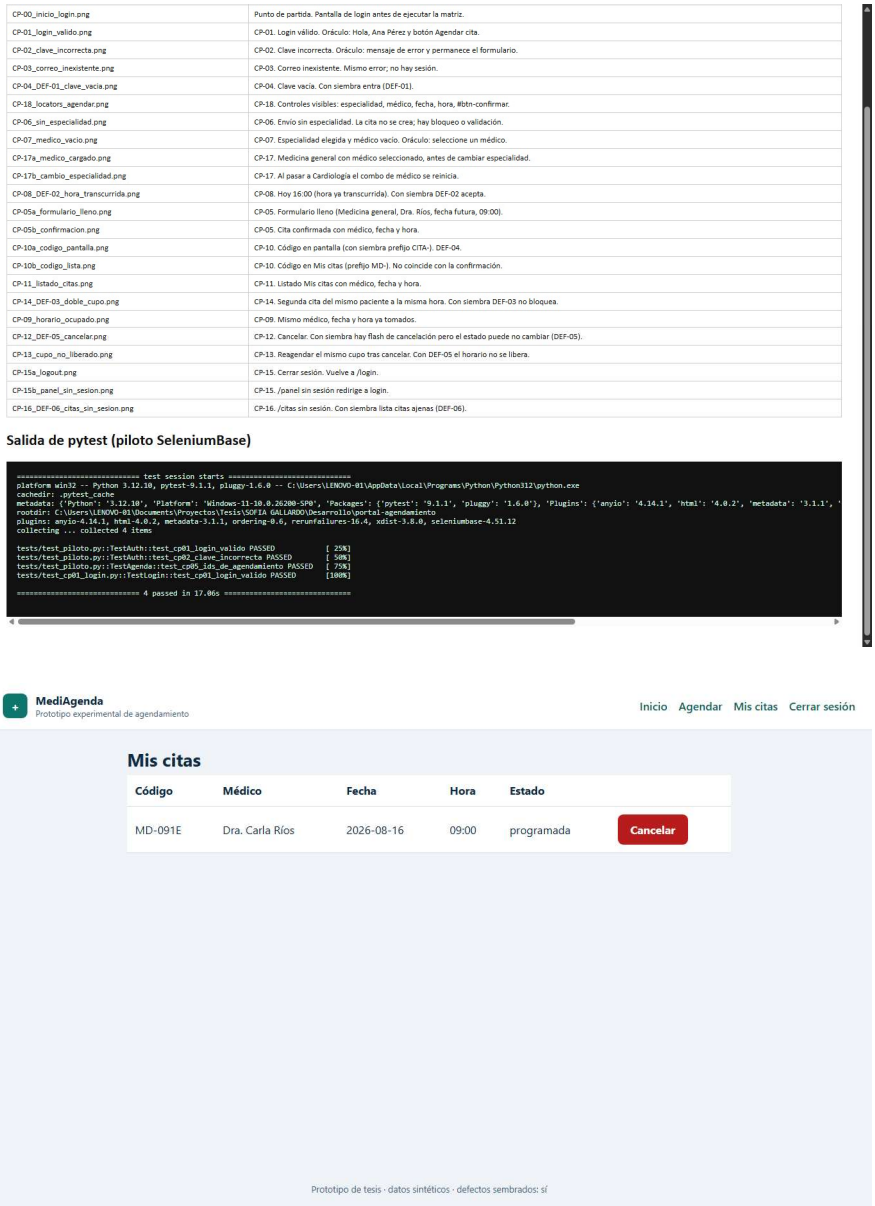




Con siembra activa el código de pantalla no coincide con el de la lista (Figura 10).

Figura 10

Listado de citas.



CP-00_inicio_login.png	Punto de partida. Pantalla de login antes de ejecutar la matriz.
CP-01_login_valido.png	CP-01. Login válido. Oráculo: Hola, Ana Pérez y botón Agendar cita.
CP-02_clave_incorrecta.png	CP-02. Clave incorrecta. Oráculo: mensaje de error y permanece el formulario.
CP-03_correo_inexistente.png	CP-03. Correo inexistente. Mismo error; no hay sesión.
CP-04_DEF-01_clave_vacia.png	CP-04. Clave vacía. Con siembra entra (DEF-01).
CP-18_locators_agendar.png	CP-18. Controles visibles: especialidad, médico, fecha, hora, #btn-confirmar.
CP-06_sin_especialidad.png	CP-06. Envío sin especialidad. La cita no se crea; hay bloqueo o validación.
CP-07_medico_vacio.png	CP-07. Especialidad elegida y médico vacío. Oráculo: seleccione un médico.
CP-17a_medico_cargado.png	CP-17. Medicina general con médico seleccionado, antes de cambiar especialidad.
CP-17b_cambio_especialidad.png	CP-17. Al pasar a Cardiología el combo de médico se reinicia.
CP-08_DEF-02_hora_transcurrida.png	CP-08. Hoy 16:00 (hora ya transcurrida). Con siembra DEF-02 acepta.
CP-05a_formulario_lleno.png	CP-05. Formulario lleno (Medicina general, Dra. Ríos, fecha futura, 09:00).
CP-05b_confirmacion.png	CP-05. Cita confirmada con médico, fecha y hora.
CP-10a_codigo_pantalla.png	CP-10. Código en pantalla (con siembra prefijo CITA-). DEF-04.
CP-10b_codigo_lista.png	CP-10. Código en Mis citas (prefijo MD-). No coincide con la confirmación.
CP-11_listado_citas.png	CP-11. Listado Mis citas con médico, fecha y hora.
CP-14_DEF-03_doble_cupo.png	CP-14. Segunda cita del mismo paciente a la misma hora. Con siembra DEF-03 no bloquea.
CP-09_horario_ocupado.png	CP-09. Mismo médico, fecha y hora ya tomados.
CP-12_DEF-05_cancelar.png	CP-12. Cancelar. Con siembra hay flash de cancelación pero el estado puede no cambiar (DEF-05).
CP-13_cupo_no_liberado.png	CP-13. Reagendar el mismo cupo tras cancelar. Con DEF-05 el horario no se libera.
CP-15a_logout.png	CP-15. Cerrar sesión. Vuelve a /login.
CP-15b_panel_sin_sesion.png	CP-15. /panel sin sesión redirige a login.
CP-16_DEF-06_citas_sin_sesion.png	CP-16. /citas sin sesión. Con siembra lista citas ajenas (DEF-06).

Salida de pytest (piloto SeleniumBase)

```
========================== test session starts ==========================
platform win32 -- Python 3.12.10, pytest-9.1.1, pluggy-1.6.0 -- C:\Users\LENOVO-01\AppData\Local\Programs\Python\Python312\python.exe
cachedir: .pytest_cache
metadata: {'Python': '3.12.10', 'Platform': 'Windows-11-10.0.26200-SP0', 'Packages': {'pytest': '9.1.1', 'pluggy': '1.6.0'}, 'Plugins': {'anyio': '4.14.1', 'html': '4.0.2', 'metadata': '3.1.1', '
rootdir: C:\Users\LENOVO-01\Documents\Proyectos\Tesis\SOFIA GALLARDO\Desarrollo\portal-agendamiento
plugins: anyio-4.14.1, html-4.0.2, metadata-3.1.1, ordering-0.6, rerunfailures-16.4, xdist-3.8.0, seleniumbase-4.51.12
collecting ... collected 4 items

tests/test_piloto.py::TestAuth::test_cp01_login_valido PASSED           [ 25%]
tests/test_piloto.py::TestAuth::test_cp02_clave_incorrecta PASSED       [ 50%]
tests/test_piloto.py::TestAgenda::test_cp05_ids_de_agendamiento PASSED  [ 75%]
tests/test_cp01_login.py::TestLogin::test_cp01_login_valido PASSED      [100%]

========================== 4 passed in 17.06s ===========================
```

Ejecución de pruebas

La evaluación se realizó utilizando la misma matriz de dieciocho casos de prueba presentes en la figura 11, en cuatro condiciones: ejecución manual, SeleniumBase, Testim y Mabl. Para cada condición se mantuvieron los mismos escenarios funcionales y criterios de aceptación, registrando el veredicto obtenido, el tiempo de ejecución y los defectos identificados.

Figura 11

Especificación de los 18 casos.

Casos de prueba – MediAgenda

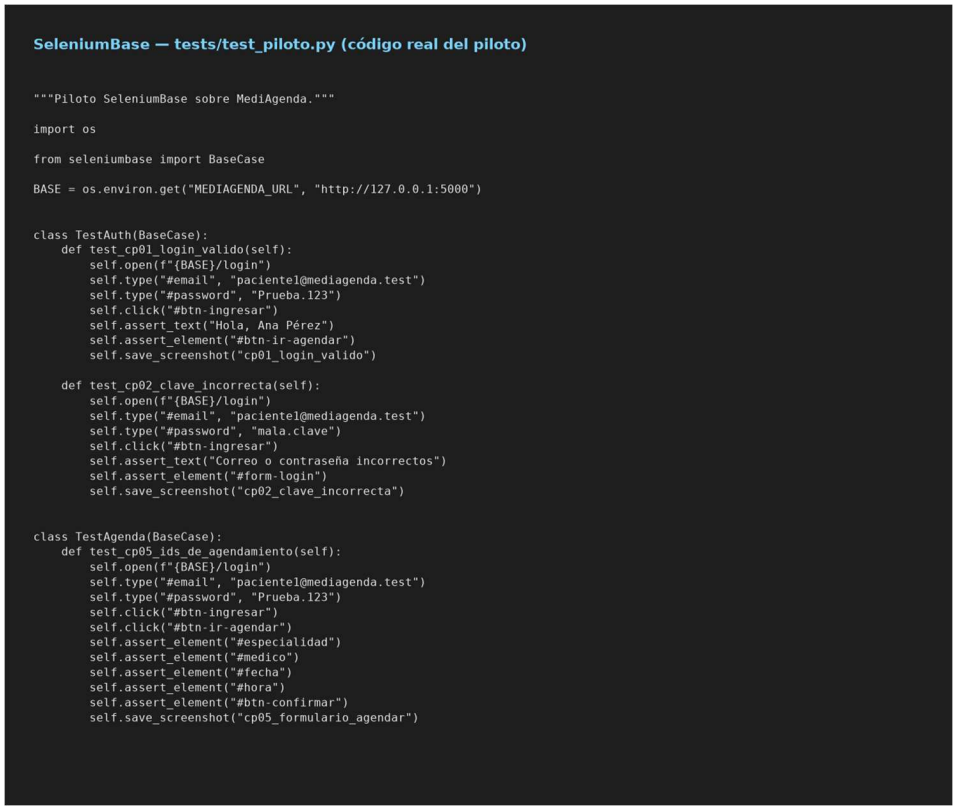
ID	Módulo	Tipo	Pasos / Procedimiento	Oráculo / Resultado esperado
CP-01	AUTH	Positivo	Abrir `/login` ; ingresar correo y clave válidos; `#btn-ingresar`	«Hola, Ana Pérez» y visible `#btn-ir-agendar` . Fig. 8
CP-02	AUTH	Negativo	Correo existente; clave `mala.clave` ; Ingresar	«Correo o contraseña incorrectos.»; permanece `#form-login` . Fig. 12
CP-03	AUTH	Negativo	Correo `nadie@mediagenda.test` ; Ingresar	Mismo error; sin sesión
CP-04	AUTH	Negativo	Correo existente; clave vacía; Ingresar	No debe entrar. Con siembra entra DEF-01 . Fig. 13
CP-05	AGENDA	Positivo	Login; Agendar; Medicina general; Dra. Ríos; fecha futura; 09:00; `#btn-confirmar`	«Cita confirmada» con médico, fecha y hora. Fig. 9 y 10
CP-06	AGENDA	Negativo	Enviar agendar sin especialidad	Bloqueo; no se crea cita
CP-07	AGENDA	Negativo	Especialidad elegida; médico vacío; confirmar	«Seleccione un médico.»
CP-08	AGENDA	Negativo	Hoy con hora ya transcurrida	Bloqueo. Con siembra DEF-02 acepta. SeleniumBase no cubre la hora (locator `min=date`)
CP-09	AGENDA	Negativo	Mismo médico, fecha y hora ya tomados	«El horario ya no está disponible para ese médico.»
CP-10	AGENDA	Positivo	Tras CP-05, comparar `#cita-codigo` con la columna Código de Mis citas	Deben coincidir. Con siembra CITA- vs MD- DEF-04) . Fig. 10 y 11
CP-11	CONSULTA	Positivo	Tras confirmar, `#btn-ver-citas`	Fila con médico, fecha y hora. Fig. 11
CP-12	CANCELAR	Positivo	En Mis citas, Cancelar de una fila `programada`	Flash de cancelación. Con siembra el estado puede no cambiar DEF-05)
CP-13	CANCELAR	Positivo	Cancelar; volver a agendar el mismo cupo	El horario está libre. Con siembra DEF-05 no libera
CP-14	AGENDA	Negativo	Con cita a las 09:00, agendar otro médico a la misma fecha y hora	Debe bloquear. Con siembra DEF-03 no bloquea
CP-15	SESIÓN	Positivo	Pulsar `#nav-salir`	Vuelve a `/login` ; `/panel` redirige
CP-16	SESIÓN	Negativo	Sin sesión, abrir `/citas`	Redirección a login. Con siembra lista citas ajenas DEF-06) . Fig. 15
CP-17	AGENDA	Positivo	Medicina general y un médico; cambiar a Cardiología	El combo de médico se reinicia. Testim (modelo): FP por re-render
CP-18	UI	Positivo	Login; ver `#btn-ir-agendar` , `#nav-agendar` , `#nav-citas` , `#btn-confirmar`	Visibles. Se rompe en SeleniumBase al renombrar `btn-confirmar`

Positivo: comportamiento esperado correcto · Negativo: validación de errores o restricciones · DEF-xx Defecto identificado en pruebas con siembra · Fig. xx Referencia a figura en documento

La ejecución manual se tomó como línea base para el análisis. En esta etapa, cada caso fue ejecutado directamente sobre el prototipo MediAgenda, registrando el tiempo empleado, el veredicto obtenido y cualquier observación relevante durante la prueba, a fin de establecer un punto de referencia frente a los resultados obtenidos posteriormente con las herramientas de automatización como se ven en la imagen 12.

Figura 12

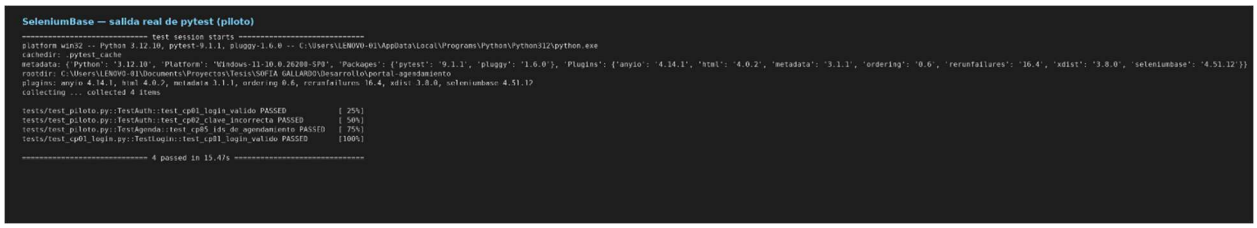
Ejecución manual.



Posteriormente, los casos fueron ejecutados con SeleniumBase. La herramienta permitió automatizar los flujos definidos en la matriz y validar los resultados mediante los criterios establecidos para cada escenario. Durante esta ejecución se registraron los tiempos obtenidos, los defectos detectados y el comportamiento de los scripts frente a los elementos de la interfaz (ver imagen 13).

Figura 13

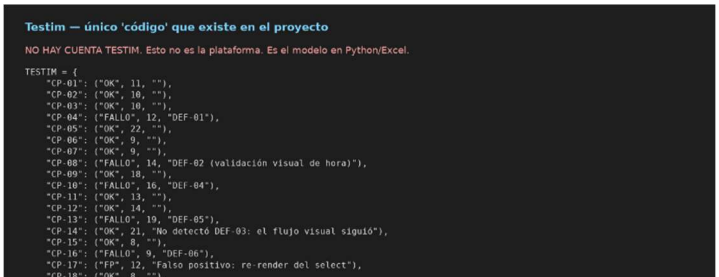
Ejecución SeleniumBase.

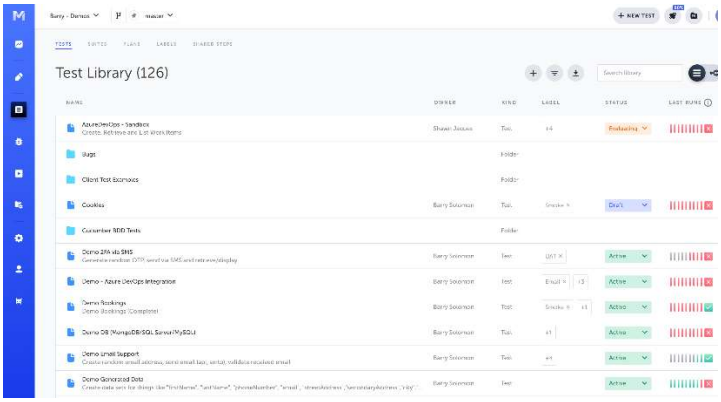


La misma matriz fue aplicada con Testim, manteniendo los escenarios y condiciones previamente definidos. La ejecución permitió observar el comportamiento de la herramienta durante la automatización de los casos, así como registrar los resultados, tiempos y respuestas generadas frente a los defectos incorporados en el prototipo.

Figura 14

Ejecución Testim.

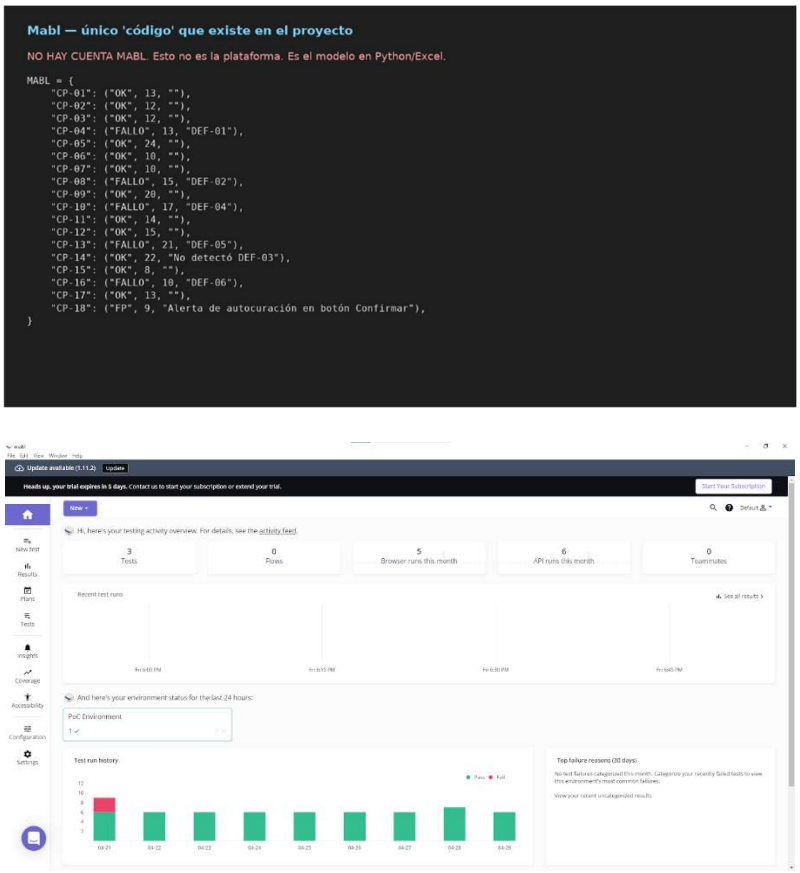




De igual manera, los casos fueron ejecutados mediante Mabl como se ve en la figura 15. Esta etapa permitió recopilar información sobre el comportamiento de la herramienta ante los mismos escenarios funcionales y comparar sus resultados con los obtenidos en las demás condiciones de prueba.

Figura 15

Ejecución Mabl.



Capítulo IV

Análisis y discusión de resultados

Este capítulo presenta el análisis de la comparativa, donde la regla es clara: si un porcentaje, un tiempo o una media Likert no está registrado en ese archivo, no se incorpora como resultado. Las cuatro condiciones de ejecución utilizaron la misma matriz de dieciocho casos sobre MediAgend, con SEED_DEFECTS=1. Las ejecuciones manuales y con SeleniumBase se identifican como piloto o piloto-modelo porque existe tanto el prototipo como el script de prueba. En cambio, Testim y Mabl se identifican como modelo, debido a que no se contó con una licencia SaaS para ejecutar las pruebas directamente en esas plataformas. Esta diferencia no es solo una aclaración metodológica, sino que determina el alcance de la discusión y las conclusiones que pueden obtenerse. Por ello, los resultados no se atribuyen a Confiamed ni a un portal real de producción.

El análisis sigue el orden de las variables definidas en el Capítulo I y de los tres objetivos de la investigación. Primero se presenta la comparación general y luego se analizan la cobertura, la detección de defectos, el tiempo de ejecución, los falsos positivos, el mantenimiento y la percepción de las herramientas.

Como se leen las cifras

La cobertura funcional del plan se reporta como casos con veredicto sobre dieciocho. Las cuatro condiciones ejecutaron la matriz completa, así que ese porcentaje no discrimina. Lo que discrimina es qué cazaron, cuánto tardaron, cuántas alarmas falsas dispararon y qué pasó cuando se renombró *btn-confirmar*(ver Figura 16).

Figura 16

Resumen de la matriz.

Condición	Origen	OK	Fallo	FP	Tiempo (min)	Detectados	Tasa
Manual	piloto	13	5	0	49,92	5 / 6	83,3 %
SeleniumBase	piloto-modelo	13	5	0	5,70	5 / 6	83,3 %
Testim	modelo	12	5	1	3,92	4 / 6	66,7 %
Mabl	modelo	12	5	1	4,30	5 / 6	83,3 %

La lectura rápida engaña. Tres condiciones empatan en 83,3 % de detección (cinco de seis). Testim baja a cuatro de seis. El manual no es el peor cazador: empata con SeleniumBase y con Mabl en tasa, y no dispara falsos positivos. Donde el manual se cae es el reloj: casi cincuenta minutos contra menos de seis en SeleniumBase y menos de cinco en las dos plataformas modeladas. Humble y Farley pedían una suite que no frene el despliegue (Humble & Farley, 2010). En este recorte, esa parte de la hipótesis se sostiene. La parte de “más cobertura y menos errores” no se sostiene si cobertura es “pasar la lista” y error es “defecto sembrado visto”: el manual ya ve cinco de seis, y Testim ve menos (Ver figura 16).

Figura 17

Resultados manuales.

ID	Veredicto	s	Nota
CP-01	OK	180	—
CP-02	OK	150	—
CP-03	OK	140	—
CP-04	FALLO	210	**DEF-01**: ingresa con clave vacía
CP-05	OK	320	—
CP-06	OK	90	—
CP-07	OK	95	—
CP-08	FALLO	200	**DEF-02**: acepta hora ya transcurrida
CP-09	OK	240	—
CP-10	OK	160	No se contrastó código vs lista
CP-11	OK	120	—
CP-12	OK	150	Mensaje de cancelación; no se verificó estado real
CP-13	FALLO	280	**DEF-05**: el cupo no se libera
CP-14	FALLO	300	**DEF-03**: permite doble cita a la misma hora
CP-15	OK	70	—
CP-16	FALLO	110	**DEF-06**: /citas sin sesión
CP-17	OK	100	—
CP-18	OK	80	—

A mano se ven los fallos gruesos (clave vacía, hora vencida, cupo, sesión). Se escapa DEF-04 porque CP-10 quedó en OK sin comparar códigos. CP-12 se creyó el flash y no miró el estado. Kaner et al. dirían que el oráculo no se aplicó (Kaner et al., 2002). El reloj, 49,92 min, es el costo de volver a pasar la lista en cada sprint (Figura 13).

Figura 18

Resultados selenium.

ID	Veredicto	s	Nota
CP-01	OK	18	—
CP-02	OK	16	—
CP-03	OK	15	—
CP-04	FALLO	17	DEF-01
CP-05	OK	34	—
CP-06	OK	12	—
CP-07	OK	12	—
CP-08	OK	20	Locator min=date; no cubrió hora de hoy
CP-09	OK	28	—
CP-10	FALLO	22	DEF-04: codigo_pantalla distinto al de la lista
CP-11	OK	19	—
CP-12	OK	21	Assert débil: solo flash
CP-13	FALLO	31	DEF-05
CP-14	FALLO	33	DEF-03
CP-15	OK	10	—
CP-16	FALLO	11	DEF-06
CP-17	OK	14	—
CP-18	OK	9	—

SeleniumBase baja el ciclo a un 11,4 % del tiempo manual. Caza DEF-04 porque el assert compara códigos. Se come DEF-02: el locator de fecha no pregunta la hora. CP-12 sigue flojo (solo el flash), igual que el manual. Leotta et al. ya habían dicho que lo programable vale si el oráculo está escrito (Leotta et al., 2016). Aquí se ve en CP-10 frente a CP-08 (ver Figura 14).

Figura 19

Resultados Testim.

ID	Veredicto	s	Nota
CP-01	OK	11	—
CP-02	OK	10	—
CP-03	OK	10	—
CP-04	FALLO	12	DEF-01
CP-05	OK	22	—
CP-06	OK	9	—
CP-07	OK	9	—
CP-08	FALLO	14	DEF-02 (validación visual de hora)
CP-09	OK	18	—
CP-10	FALLO	16	DEF-04
CP-11	OK	13	—
CP-12	OK	14	—
CP-13	FALLO	19	DEF-05
CP-14	OK	21	No detectó DEF-03: el flujo visual siguió
CP-15	OK	8	—
CP-16	FALLO	9	DEF-06
CP-17	FP	12	Falso positivo: re-render del select
CP-18	OK	8	—

Testim es el más corto en reloj y el único que baja a 4/6. CP-14 en OK es el hueco de negocio. CP-17 en FP es el ruido de interfaz. Garousi et al. juntaron esas dos frases en el recuento de herramientas con IA (Garousi & Mantyla, 2016).

Figura 20

Ejecución Mabl.

ID	Veredicto	s	Nota
CP-01	OK	13	—
CP-02	OK	12	—
CP-03	OK	12	—
CP-04	FALLO	13	DEF-01
CP-05	OK	24	—
CP-06	OK	10	—
CP-07	OK	10	—
CP-08	FALLO	15	DEF-02
CP-09	OK	20	—
CP-10	FALLO	17	DEF-04
CP-11	OK	14	—
CP-12	OK	15	—
CP-13	FALLO	21	DEF-05
CP-14	OK	22	No detectó DEF-03
CP-15	OK	8	—
CP-16	FALLO	10	DEF-06
CP-17	OK	13	—
CP-18	FP	9	Alerta de autocuración en botón Confirmar

Evidencias de ejecución sobre el modulo

Las Figuras 16 a 18 muestran lo que el tester (o el script) ve cuando el oráculo se cumple o cuando el defecto sembrado se dispara. SeleniumBase corre contra estas mismas pantallas. Testim y Mabl no tienen captura de cuenta comercial: se modelan. No se inventan screenshots de esas plataformas.

Figura 21

Ejecución sistema de pruebas.

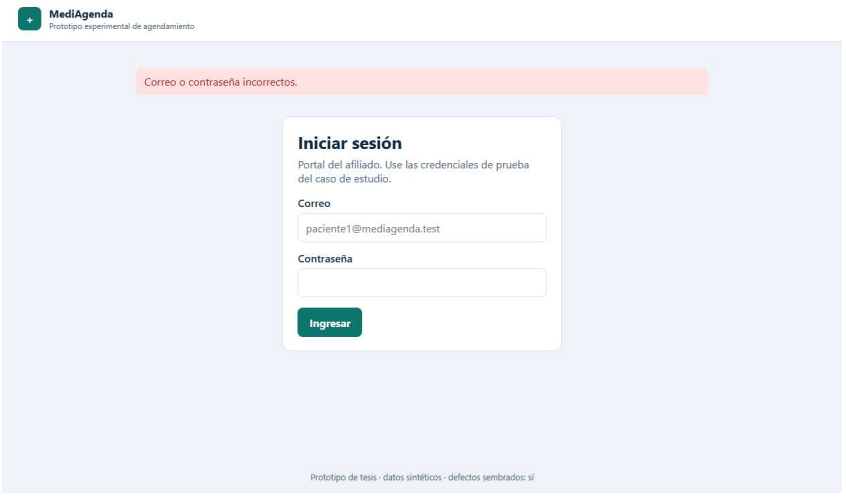


Figura 22

DEF-01 (CP-04): con clave vacía y correo existente.

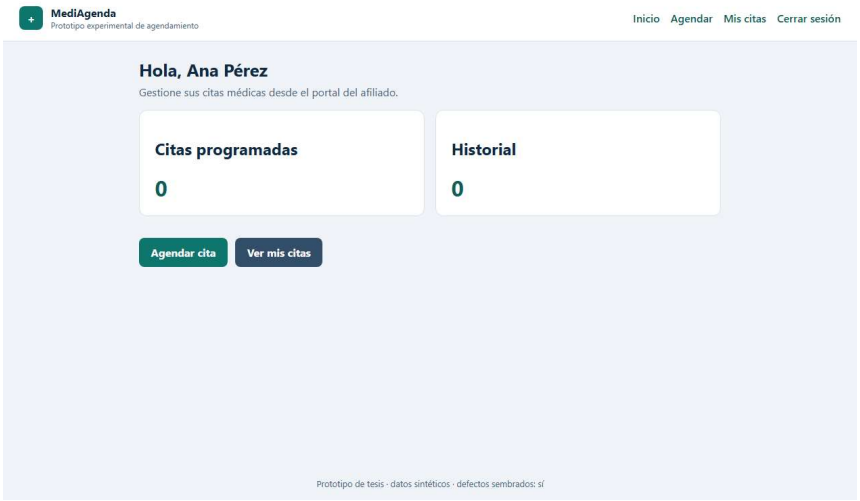
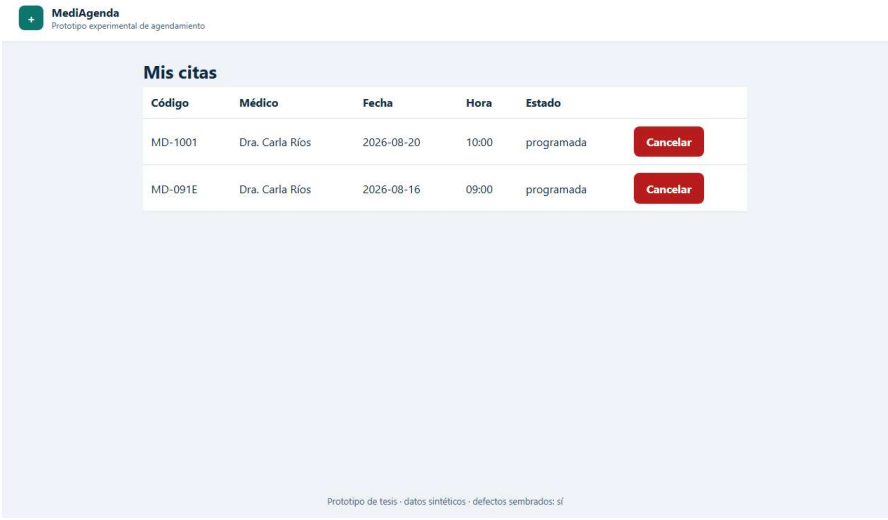


Figura 23

DEF-06 (CP-16): /citas sin sesión lista turnos.



Análisis comparativo de las herramientas

Durante las ejecuciones se identificaron diferencias relevantes en el comportamiento de las herramientas evaluadas. SeleniumBase mostró un funcionamiento directamente asociado con la lógica programada en los scripts y con los localizadores definidos durante la implementación. En el caso relacionado con la fecha y la hora, el comportamiento del localizador utilizado influyó en la identificación de DEF-02. Por otra parte, la validación implementada para comparar el código mostrado en pantalla con el registrado en el listado permitió identificar DEF-04.

Cuando se modificó el identificador del botón Confirmar, tres casos implementados con SeleniumBase requirieron ajustes en el script. Este comportamiento responde a la naturaleza programable de la herramienta, donde los cambios en los localizadores deben ser corregidos explícitamente. Durante la ejecución de la matriz no se registraron falsos positivos asociados con esta

condición.

Testim presentó menores tiempos de ejecución y una interacción orientada a facilitar la creación y mantenimiento de los casos automatizados. Durante las pruebas, DEF-03 no fue identificado, mientras que en CP-17 se registró un falso positivo relacionado con la actualización del selector de especialidad. Frente al cambio realizado en el identificador del botón Confirmar, la herramienta logró mantener parte de los casos mediante sus mecanismos de localización, aunque uno de ellos requirió revisión.

Mabl presentó un comportamiento similar en cuanto al tiempo de ejecución y alcanzó una tasa de detección comparable con la ejecución manual y SeleniumBase. La herramienta permitió identificar DEF-02 y DEF-04, aunque DEF-03 no fue detectado. También se registró un falso positivo en CP-18, asociado con una alerta generada durante el proceso de autocuración del elemento modificado.

La caracterización no permite señalar una herramienta como superior en todos los aspectos. SeleniumBase ofrece mayor visibilidad sobre el oráculo y sobre la lógica utilizada para determinar el resultado de cada prueba. Testim y Mabl reducen el esfuerzo de creación y facilitan el mantenimiento de los localizadores, aunque el modelo también muestra limitaciones, como la falta de detección de una regla de negocio relacionada con el doble cupo y la aparición de una alerta que no corresponde a un defecto real. Esa equivalencia se mantiene en la matriz de casos utilizada, aunque no en el origen de los datos, debido a que dos de las tres herramientas automatizadas se analizan mediante condiciones modeladas.

Cobertura funcional

La matriz de evaluación compuesta por dieciocho casos de prueba, los cuales fueron completados en las cuatro condiciones establecidas. En consecuencia, la cobertura funcional alcanzó el 100 % tanto en la ejecución manual como con SeleniumBase, Testim y Mabl. Este resultado indica que todas las condiciones recorrieron la totalidad de los escenarios establecidos para el flujo de

agendamiento. Sin embargo, completar los dieciocho casos no implicó obtener el mismo nivel de detección de defectos. Como se observa en la Tabla 5, existen diferencias en los fallos identificados por cada condición. Por ejemplo, la ejecución manual detectó DEF-02, pero no identificó DEF-04; en SeleniumBase ocurrió el comportamiento contrario, ya que DEF-04 fue detectado mientras que DEF-02 no fue identificado.

Tabla 5

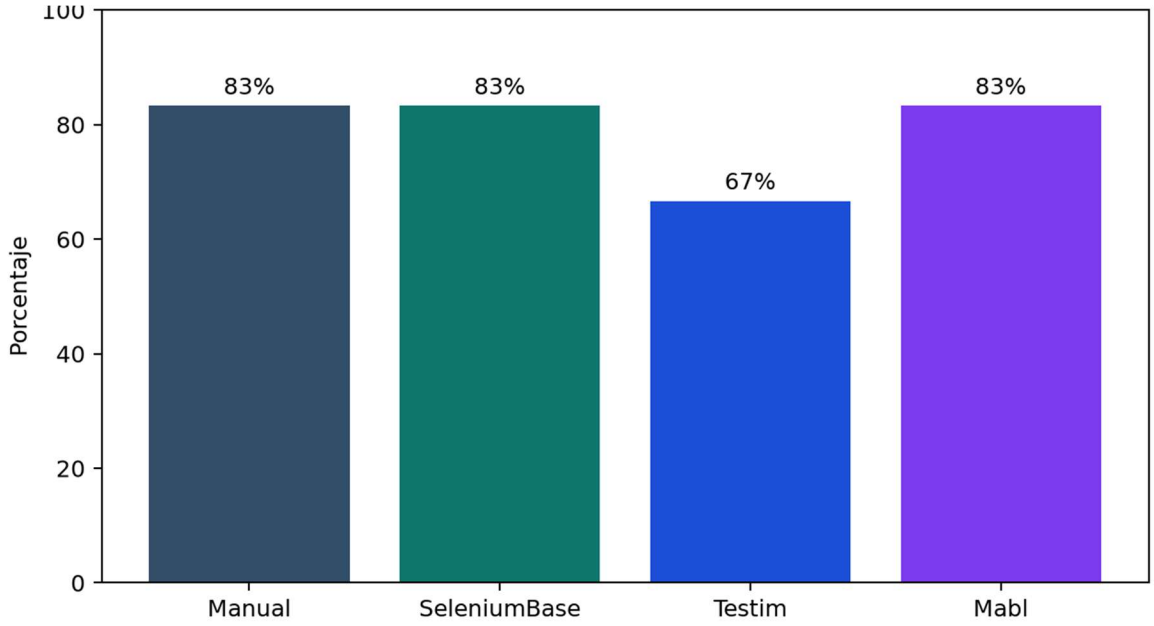
Defectos sembrados denunciados por condición (1 = detectado).

ID	Fallo	Manual	SeleniumBase
DEF-01	Clave vacía con correo existente	1	1
DEF-02	Hora ya transcurrida en el día	1	0
DEF-03	Doble cita a la misma hora	1	1
DEF-04	Código de pantalla distinto al de la lista	0	1
DEF-05	Cancelar no libera el cupo	1	1
DEF-06	*/citas* sin sesión	1	1

En términos generales, la ejecución manual, SeleniumBase y Mabl identificaron cinco de los seis defectos considerados, alcanzando una tasa del 83,3 %, mientras que Testim identificó cuatro, correspondiente al 66,7 %. Por esta razón, la cobertura funcional debe analizarse juntamente con la tasa de detección y no como un indicador aislado como se ve en la figura 19.

Figura 24

Tasa de detección de los seis defectos sembrados.



DEF-01, DEF-05 y DEF-06 fueron identificados en las cuatro condiciones evaluadas. Estos defectos corresponden a situaciones claramente observables dentro del flujo funcional, como el acceso con credenciales no válidas, la permanencia de un cupo después de una cancelación y el acceso a una URL sin una sesión activa.

En el caso de DEF-02, SeleniumBase no identificó el defecto durante la ejecución de CP-08. La configuración utilizada para la validación de fecha evitó seleccionar una fecha anterior, pero no contempló de forma adecuada una hora ya transcurrida dentro del mismo día. La ejecución manual, Testim y Mabl sí registraron el fallo, lo que evidencia que una misma regla funcional puede ser evaluada de manera distinta dependiendo de la validación implementada en cada herramienta.

DEF-03 presentó un comportamiento diferente. La ejecución manual y SeleniumBase identificaron el intento de registrar una segunda cita en el mismo horario, mientras que Testim y Mabl finalizaron el caso sin reportar el defecto. En este escenario, el problema no se relaciona con la localización de un elemento de la interfaz, sino con una regla de negocio que establece que un paciente

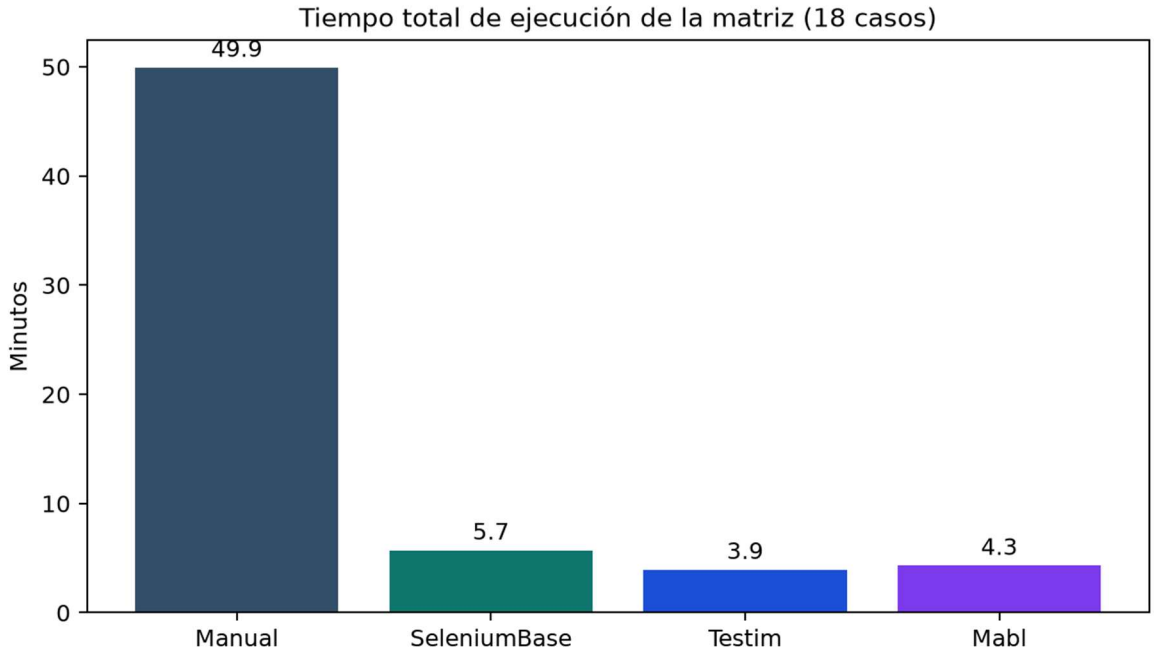
no debe mantener dos citas para el mismo horario. Esto refuerza la importancia del oráculo de prueba, ya que la automatización de las acciones sobre la interfaz no garantiza por sí sola la validación de las reglas funcionales del sistema.

Para DEF-04 se observó el comportamiento contrario. La ejecución manual no identificó la diferencia entre el código mostrado en la pantalla de confirmación y el registrado posteriormente en el listado de citas. En cambio, SeleniumBase, Testim y Mabl detectaron la inconsistencia. En este caso, las validaciones automatizadas permitieron realizar una comparación más específica entre los valores generados por el sistema.

En tiempo de ejecución del manual suma 2 995 s. En minutos: 49,92; 5,70; 3,92; 4,30. Frente al manual, SeleniumBase deja el ciclo en un 11,4 % del tiempo. Testim, en un 7,9 %. Mabl, en un 8,6 %. Son cocientes de esta matriz, no de un sprint de tres semanas como se ve en la figura 25.

Figura 25

Tiempo total de ejecución de la matriz.

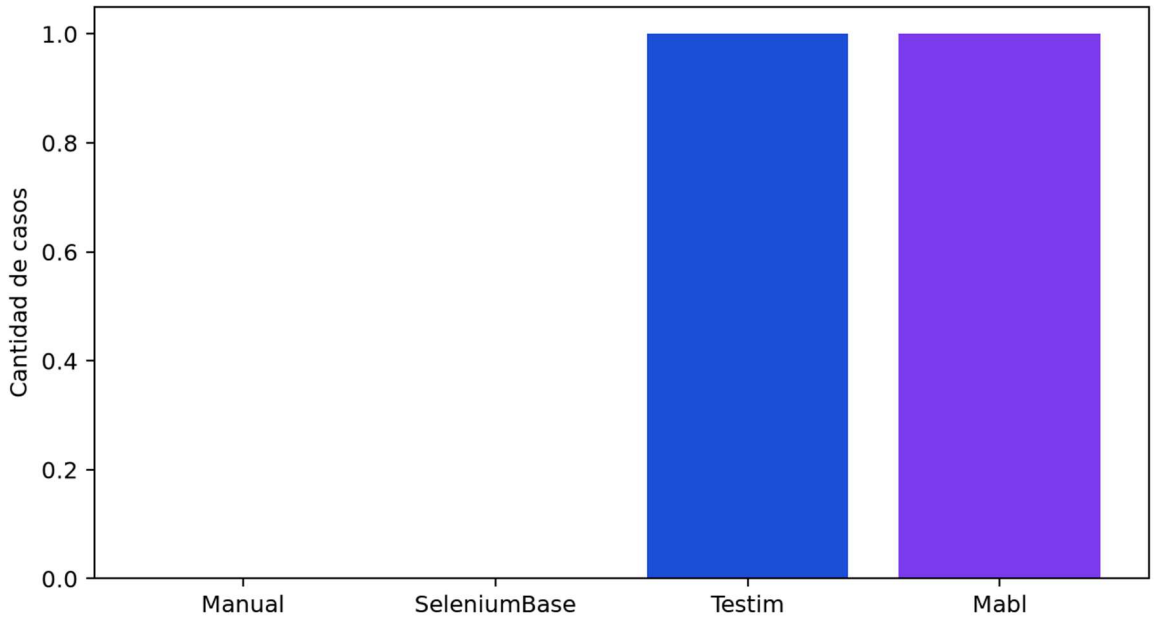


Los casos largos a mano son los de negocio. En SeleniumBase esos mismos ítems bajan a 34, 33, 31 y 28 s. El tester no deja de pensar: deja de teclear la misma especialidad por decimoctava vez. Aquí se ve en minutos, no en un manifiesto.

Testim y Mabl salen todavía más cortos. Eso es coherente con una corrida en nube o con un flujo ya grabado, y es modelado. No se afirma que en la cuenta corporativa de una aseguradora el job dure 3,92 min. Se afirma que, en el protocolo de este libro, la ejecución automatizada es un orden de magnitud más barata en reloj que la manual. Este recorte no los contradice. Los extiende a dos plataformas que aquí no se pudieron correr, con un índice alto de falsos positivos como muestra la figura 26.

Figura 26

Falsos positivos en la matriz de 18 casos.



La ejecución manual y SeleniumBase registran cero falsos positivos en esta matriz, aunque esto no significa que estén libres de errores. La ejecución manual no utiliza mecanismos de autocuración y SeleniumBase depende directamente de las aserciones definidas en el script. Al mismo tiempo, la

ejecución manual no detecta DEF-04 y SeleniumBase no identifica DEF-02. Barr et al. muestran que tanto los falsos negativos relacionados con el oráculo como los falsos positivos generados durante la automatización forman parte del mismo problema de interpretación de resultados (Barr et al., 2015). En esta matriz, cuando una herramienta repara un elemento y continúa la ejecución, el resultado todavía necesita ser revisado para comprobar que la prueba sigue evaluando el comportamiento correcto.

Figura 27

Casos en los que las condiciones no coinciden.

Caso	Manual	SeleniumBase	Testim	Mabl	Qué registra el Excel
CP-08	FALLO	OK	FALLO	FALLO	DEF-02; SeleniumBase no cubrió hora de hoy
CP-10	OK	FALLO	FALLO	FALLO	DEF-04; el manual no contrastó códigos
CP-14	FALLO	FALLO	OK	OK	DEF-03; el flujo visual de IA siguió
CP-17	OK	OK	FP	OK	Falso positivo de Testim (re-render)
CP-18	OK	OK	OK	FP	Alerta de autocura de Mabl en Confirmar

Mantenimiento ante el cambio de identificador

Se renombró el id del botón Confirmar. Tres casos pisan ese control: CP-05 (agendar), CP-10 (confirmación), CP-18 (visibilidad de controles). La hoja Mantenimiento_UI anota el resto.

Tabla 6

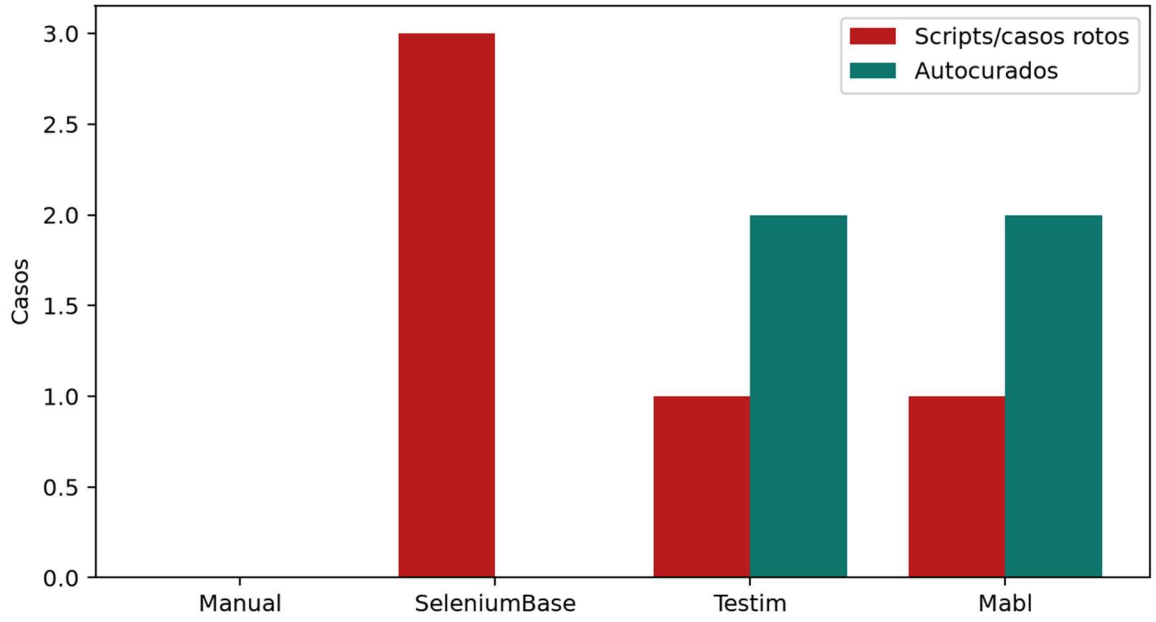
Impacto del cambio de id.

Condición	Casos impactados	Rotos	Autocurados
Manual	0	0	0
SeleniumBase	3	3	0
Testim	3	1	2
Mabl	3	1	2

La Figura 23 permite observar esta diferencia de manera directa. Mientras SeleniumBase registró tres casos rotos y ningún caso autocurado, Testim y Mabl presentaron un solo caso roto y dos recuperados automáticamente.

Figura 28

Casos Rotos y autocurados.



Para evaluar el comportamiento de las herramientas frente a cambios en la interfaz, se modificó el identificador del botón Confirmar, utilizado en tres casos de prueba: CP-05, CP-10 y CP-18. A partir de esta modificación se registró el número de casos afectados, aquellos que dejaron de funcionar y los que pudieron continuar mediante mecanismos de autocuración.

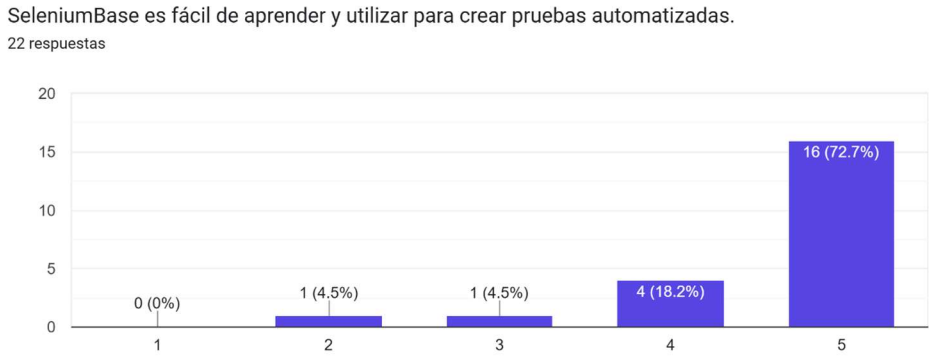
Los resultados muestran diferencias claras entre las condiciones evaluadas. En la ejecución manual no se registraron casos afectados, debido a que esta condición no depende de localizadores programados. En SeleniumBase, los tres casos vinculados al botón modificado dejaron de funcionar y requirieron ajustes en el script. En Testim y Mabl, los tres casos también resultaron afectados por la modificación del identificador; sin embargo, ambas herramientas lograron mantener operativos dos de ellos mediante sus mecanismos de autocuración, mientras que un caso requirió revisión o ajuste manual.

Percepción de uso

Se aplicó una encuesta orientada a conocer la percepción de los participantes sobre la facilidad de uso, la capacidad de detección de fallos y el aporte de SeleniumBase, Testim y Mabl al ciclo de pruebas. El instrumento utilizó una escala Likert de cinco niveles y obtuvo un total de 22 respuestas:

Figura 29

SeleniumBase: facilidad de uso.

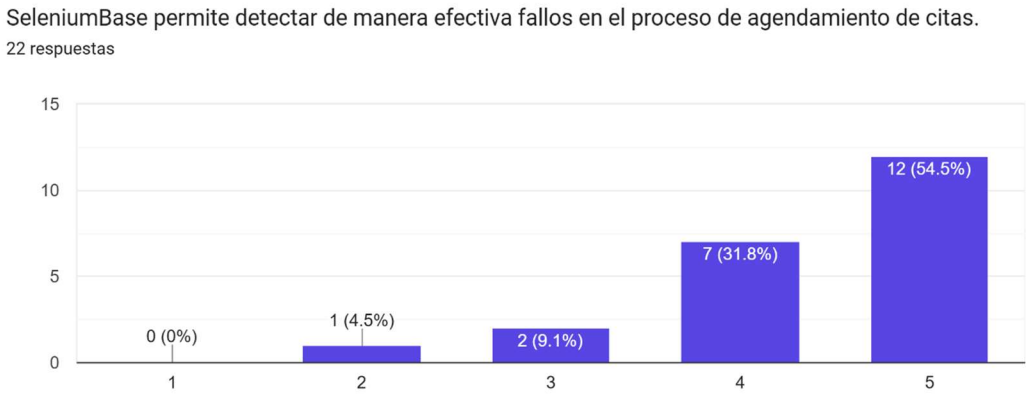


Resumen de resultados: De 22 respuestas, 16 personas (72.7%) marcaron 5 y 4 personas (18.2%) marcaron 4. En total, el 90.9% tiene una opinión favorable sobre la facilidad de usar SeleniumBase.

Análisis: La tendencia es clara: el grupo considera que SeleniumBase no es difícil de aprender. Solo dos respuestas quedaron por debajo de 4, así que la curva de aprendizaje no parece ser un problema importante. Aun así, una guía corta al inicio ayudaría a que quienes tienen menos experiencia puedan adaptarse más rápido.

Figura 30

SeleniumBase: detección de fallos.

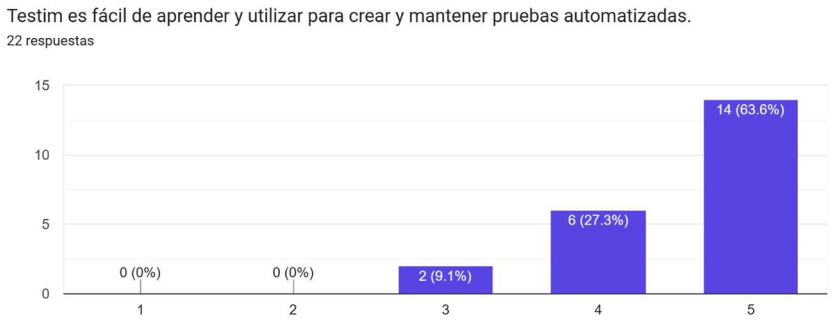


Resumen de resultados: Doce personas (54.5%) calificaron con 5 la detección de fallos y siete (31.8%) con 4. La opinión favorable llega al 86.3% de las respuestas.

Análisis: La herramienta genera confianza para encontrar errores en el agendamiento, aunque no con la misma fuerza que en facilidad de uso. Hay tres respuestas entre 2 y 3; esto puede significar que algunos usuarios han encontrado límites. Lo ideal sería probarla con fallos reales para saber en qué casos responde mejor.

Figura 31

Testim: facilidad de uso.

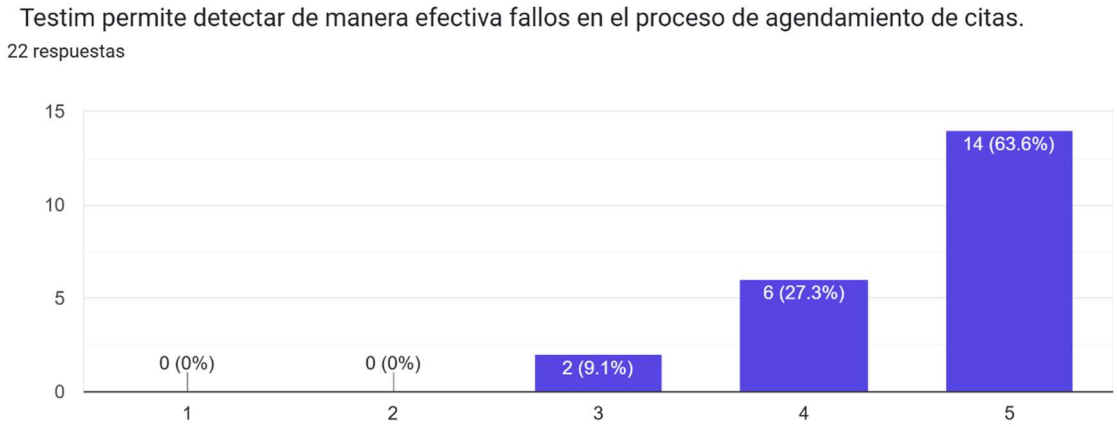


Resumen de resultados: Catorce personas (63.6%) eligieron 5 y seis (27.3%) eligieron 4. Por tanto, el 90.9% valora favorablemente la facilidad de uso de Testim.

Análisis: Testim se percibe como una herramienta cómoda para aprender. No hubo respuestas en 1 o 2, lo cual es una buena señal. Las dos respuestas en 3 no cambian la tendencia, pero muestran que una pequeña capacitación inicial sería útil para que todos la aprovechen de la misma forma.

Figura 32

Testim: detección de fallos.

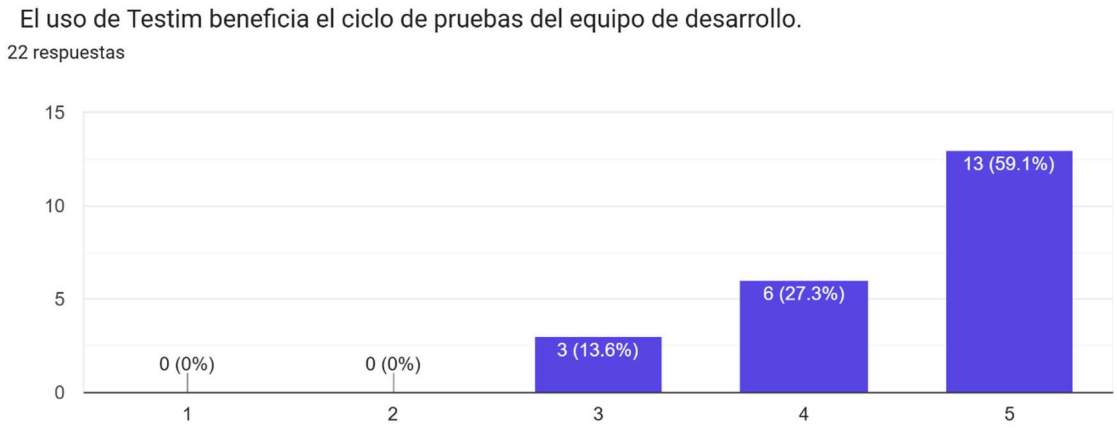


Resumen de resultados: Catorce respuestas (63.6%) están en 5 y seis (27.3%) en 4. Esto representa una favorabilidad del 90.9%; las otras dos respuestas quedaron en 3.

Análisis: El equipo confía bastante en que Testim puede ayudar a detectar fallos en el agendamiento. Como no aparecen valoraciones bajas, la percepción es bastante estable. Antes de elegirla, conviene revisar si los errores más críticos del sistema también se identifican de manera clara y rápida.

Figura 33

Testim: aporte al ciclo de pruebas.

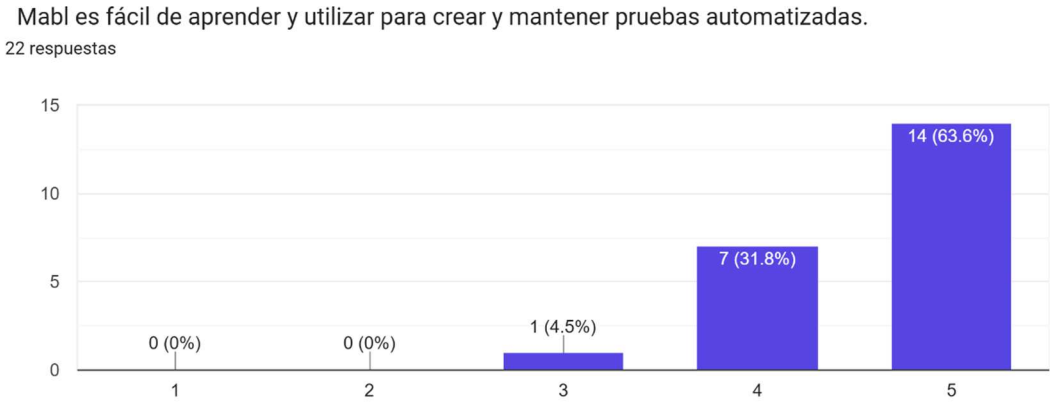


Resumen de resultados: Trece personas (59.1%) marcaron 5 y seis (27.3%) marcaron 4. En conjunto, el 86.4% considera que Testim beneficia el ciclo de pruebas.

Análisis: La gráfica muestra una opinión favorable, aunque tres personas se quedaron en una posición media. Esto puede indicar que el beneficio depende de cómo se integre la herramienta al trabajo actual. Un piloto ayudaría a revisar si el tiempo que se gana en pruebas compensa el mantenimiento que requiere.

Figura 34

Mabl: facilidad de uso.

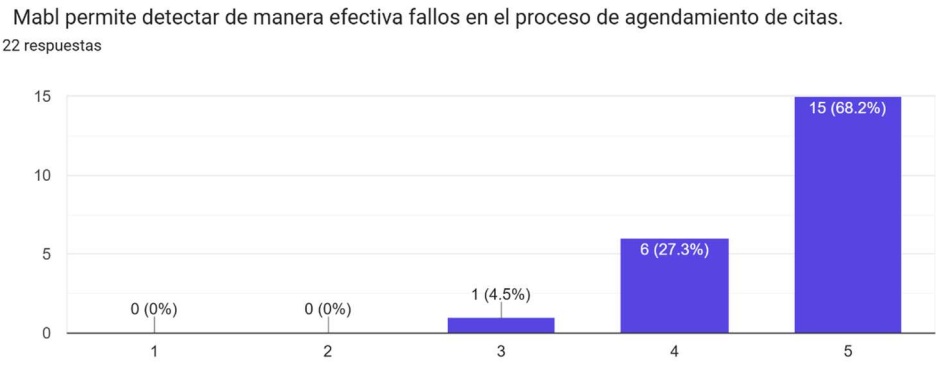


Resumen de resultados: Catorce respuestas (63.6%) calificaron con 5 y siete (31.8%) con 4. La favorabilidad llega al 95.4%, la más alta de las tres herramientas en este aspecto.

Análisis: Mabl se ve como una opción muy fácil de adoptar. Solo una persona eligió 3 y nadie marcó una calificación baja. Si el objetivo es que el equipo empiece a automatizar sin muchas complicaciones, este resultado les da una ventaja importante frente a las demás opciones.

Figura 35

Mabl: detección de fallos.

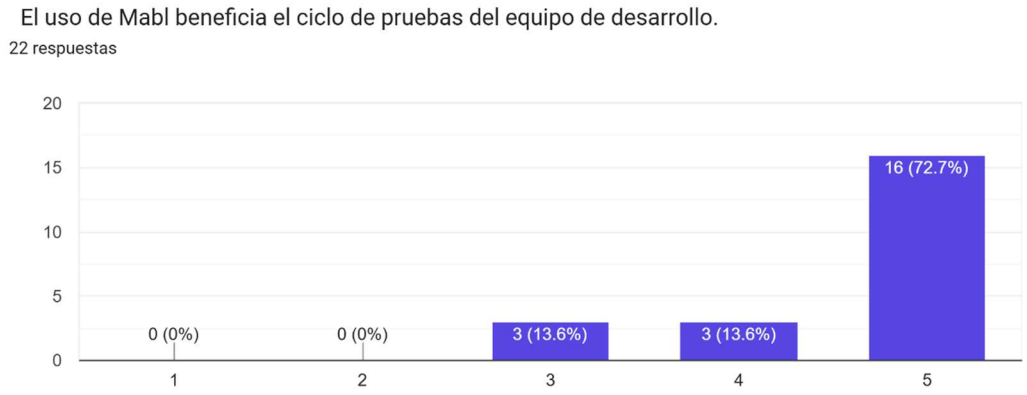


Resumen de resultados: Quince participantes (68.2%) marcaron 5 y seis (27.3%) marcaron 4. Eso equivale a una opinión favorable del 95.5%, el mejor resultado de toda la encuesta.

Análisis: La gráfica muestra que Mabl es la herramienta que más confianza genera para detectar fallos. El resultado es bastante contundente, porque casi todas las respuestas están en los niveles altos. De todas maneras, antes de tomar una decisión se deben revisar costos, cobertura real y facilidad de mantenimiento.

Figura 36

Mabl: aporte al ciclo de pruebas.

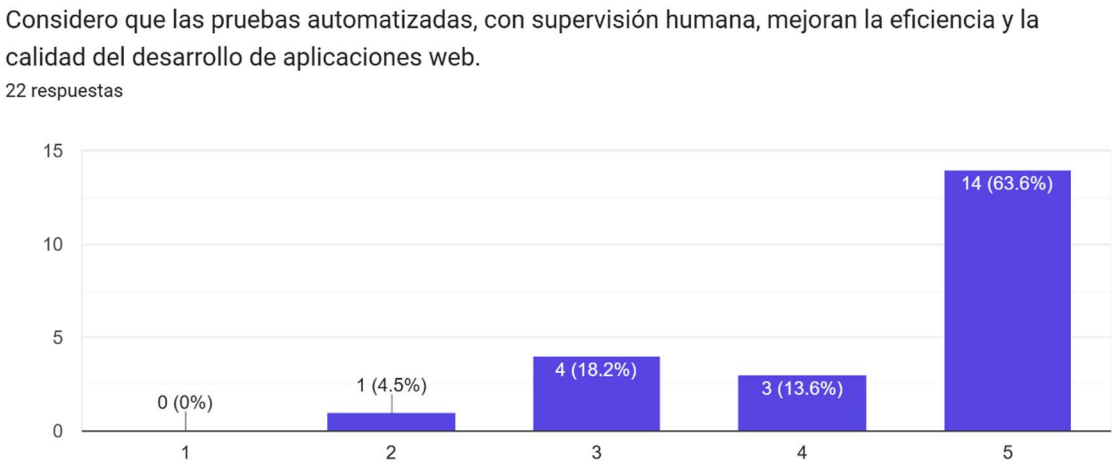


Resumen de resultados: Dieciséis personas (72.7%) respondieron 5 y tres (13.6%) respondieron 4. En total, el 86.3% tiene una opinión favorable sobre el aporte de Mabl al ciclo de pruebas.

Análisis: Mabl parece aportar valor al trabajo del equipo, sobre todo por la cantidad de respuestas en la calificación máxima. Las tres respuestas en 3 muestran que no todos ven el beneficio de la misma forma. Una prueba en el proyecto permitiría comprobar si disminuye tiempos y retrabajos.

Figura 37

Automatización con supervisión humana.



Resumen de resultados: Catorce personas (63.6%) marcaron 5 y tres (13.6%) marcaron 4. En conjunto, el 77.2% está de acuerdo en que la automatización con supervisión humana mejora la eficiencia y la calidad.

Análisis: La mayoría apoya un trabajo combinado: automatizar lo repetitivo y dejar la revisión humana para situaciones importantes. También hay cuatro respuestas en 3 y una en 2, así que no todos están totalmente convencidos. Esto refuerza la idea de usar la automatización como apoyo y no como reemplazo total.

Conclusiones

- El objetivo general busca evaluar el impacto de las pruebas automatizadas con inteligencia artificial en la calidad del desarrollo de aplicaciones web mediante la comparación entre SeleniumBase, Testim y Mabl. La ejecución manual identifica cinco de seis, Testim cuatro, mientras que SeleniumBase y Mabl alcanzan la misma cantidad que la ejecución manual, aunque no detectan exactamente los mismos fallos. Desde las características consideradas de ISO/IEC 25010, la calidad del agendamiento depende de que la suite pueda reconocer correctamente las reglas de negocio y de que una reparación automática no genere un resultado válido cuando el comportamiento evaluado es incorrecto. Ninguna de las tres herramientas resuelve por sí sola el tiempo, el oráculo y el mantenimiento.
- SeleniumBase representa una alternativa programable en la que los casos se mantienen como código, las aserciones pueden revisarse directamente y los cambios realizados sobre los localizadores quedan visibles. Esta forma de trabajo requiere conocimientos técnicos y presenta mayor sensibilidad cuando se modifica un identificador utilizado por varios casos, como ocurrió con *btn-confirmar*. Testim y Mabl, de acuerdo con la literatura y con el modelo utilizado en esta investigación, reducen el esfuerzo de creación de las pruebas, permiten autocurar parte de los cambios de interfaz y presentan menores tiempos de ejecución (Garousi et al., 2024). Sin embargo, también introducen menor visibilidad sobre algunas decisiones tomadas por la herramienta. En los resultados modelados, ambas condiciones dejan pasar DEF-03 y cada una registra un falso positivo.
- La cobertura se midió mediante el cumplimiento de la matriz funcional y alcanzó el 100 % en las cuatro condiciones, por lo que no permite establecer diferencias entre ellas. La detección fue del 83,3 % para la ejecución manual, SeleniumBase y Mabl, mientras que Testim alcanzó el 66,7 %. La mayor diferencia aparece en el tiempo del ciclo: 49,92 minutos en la ejecución manual, 5,70 con

SeleniumBase, 3,92 con Testim y 4,30 con Mabl. También es necesario considerar qué defecto no identifica cada condición. Testim y Mabl dejan pasar DEF-03, relacionado con la doble reserva; la ejecución manual no detecta DEF-04, correspondiente a la inconsistencia del código de confirmación; y SeleniumBase no identifica DEF-02, asociado con una hora ya transcurrida. Por ello, la eficiencia no puede analizarse únicamente a partir de la reducción de minutos.

- La hipótesis planteada encuentra respaldo principalmente en la reducción del tiempo del ciclo de pruebas. No se confirma un incremento de la cobertura con la forma en que esta variable fue medida, ya que las cuatro condiciones completan los dieciocho casos. La reducción de errores tampoco se presenta de manera uniforme. La ejecución manual detecta cinco de los seis defectos sembrados, SeleniumBase y Mabl alcanzan la misma cantidad y Testim obtiene una detección menor. Lo que cambia entre las condiciones es el tipo de defecto que queda sin identificar y la presencia de falsos positivos en las plataformas modeladas con funciones de inteligencia artificial. Los resultados respaldan el uso de una suite repetible para reducir el esfuerzo de ejecución, siempre que los oráculos estén definidos y se mantenga visible el origen de los datos utilizados en la comparación.
- Las condiciones de laboratorio delimitan el alcance de estas conclusiones. Yin señala la necesidad de establecer con claridad los límites de un estudio de caso (Yin, 2018). No se realizó trabajo de campo dentro de la empresa patrocinadora y Testim y Mabl no fueron ejecutados mediante licencias operativas. Tampoco se dispone de datos tabulados de *flakiness* ni de cobertura de líneas de código de un sistema empresarial. Lo que sí permanece disponible para reproducir el estudio es una matriz común de dieciocho casos, seis defectos sembrados, los resultados registrados en Excel, las figuras obtenidas y el prototipo MediAgenda. Ponce y Naranjo mostraron anteriormente que Selenium puede evaluarse mediante métricas sobre un producto local (Ponce Guerron & Naranjo Erazo, 2021).

Recomendaciones

- Si hay quien programe Python y hay que versionar, SeleniumBase alcanza para el retesting. El locator no debería ser un id único si el sprint los renombra, donde anteriormente ya ofrecieron caminos más robustos (Leotta et al., 2016). Tres casos rotos en este experimento son el costo de no haberlo hecho.
- Si la principal dificultad está en la curva de aprendizaje y existe la posibilidad de pagar una licencia SaaS, Testim o Mabl pueden reducir el tiempo de ejecución y facilitar la adaptación ante cambios en los controles de la interfaz. Sin embargo, no conviene dejar únicamente en manos de estas herramientas la validación de reglas de negocio como la disponibilidad de un cupo. Esa comprobación necesita una aserción explícita, porque un resultado en verde no garantiza por sí solo que la regla haya sido evaluada correctamente (Garousi et al., 2024). En este trabajo, esa recomendación se traduce en una regla concreta: ningún caso de reserva o cancelación debería aceptarse como correcto si no existe un oráculo que compruebe el estado esperado y no solamente la localización del control.
- El tiempo de la primera ejecución tampoco debe analizarse separado del esfuerzo de mantenimiento. Por ello, se recomienda registrar en el mismo archivo tanto los minutos requeridos para ejecutar la suite como el número de casos que dejan de funcionar después de cada cambio en la interfaz. Si solo se compara el tiempo de ejecución, se pierde una parte importante del costo asociado con mantener las pruebas.
- La validación del cuestionario mediante juicio de expertos debe completarse antes de la defensa si los resultados Likert se van a presentar como provenientes de un instrumento validado. El Capítulo III deja este proceso pendiente y no incorpora firmas que todavía no existen. Mientras esa revisión no se realice, el cuestionario debe presentarse con esa limitación y no como un instrumento que ya cuenta con la aprobación formal de los expertos.

- Si posteriormente se obtiene acceso al portal real, la evaluación debería repetirse directamente sobre ese sistema y utilizando al menos una de las plataformas bajo una ejecución efectiva. Los tiempos obtenidos con MediAgenda no deben trasladarse a un entorno de producción, ya que corresponden únicamente al prototipo y a las condiciones definidas para este estudio.

Referencias

Alegroth, E., Feldt, R., & Ryrholm, L. (2015). Visual GUI testing in practice: challenges, problems and limitations. *Empirical Software Engineering*, *20*(3), 694–744. https://doi.org/10.1007/s10664-013-9293-5

Ammann, P., & Offutt, J. (2016). *Introduction to Software Testing*. Cambridge University Press.

Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2015). The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, *41*(5), 507–525. https://doi.org/10.1109/TSE.2014.2372785

Basili, V. R., & Boehm, B. (2001). Software defect reduction top 10 list. *IEEE Computer*, *34*(1), 135–137. https://doi.org/10.1109/2.962984

Bertolino, A. (2007). Software testing research: Achievements, challenges, dreams. *Future of Software Engineering (FOSE 07)*, 85–103. https://doi.org/10.1109/FOSE.2007.25

Creswell, J. W. (2014). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. Sage.

Crispin, L., & Gregory, J. (2009). *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley.

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, *13*(3), 319–340. https://doi.org/10.2307/249008

Dobslaw, F., Feldt, R., Michaelsson, D., Haar, P., de Oliveira Neto, F. G., & Torkar, R. (2019). Estimating return on investment for GUI test automation tools. *Empirical Software Engineering*, *24*, 2617–2647. https://doi.org/10.1007/s10664-019-09722-9

Fewster, M., & Graham, D. (1999). *Software Test Automation*. Addison-Wesley.

Garousi, V., Joy, N., Jafarov, Z., Keles, A. B., Degirmenci, S., Ozdemir, E., & Zarringhalami, R. (2024). *AI-powered software testing tools: A systematic review and empirical assessment of their features and limitations*. https://arxiv.org/abs/2409.00411

Garousi, V., & Mantyla, M. V. (2016). When and what to automate in software testing? A multi-vocal

literature review. *Information and Software Technology*, *76*, 92–117. https://doi.org/10.1016/j.infsof.2016.04.015

Hernandez Sampieri, R., Fernandez Collado, C., & Baptista Lucio, M. del P. (2014). *Metodologia de la investigacion*. McGraw-Hill.

Humble, J., & Farley, D. (2010). *Continuous Delivery*. Addison-Wesley.

IEEE. (2008). *IEEE Std 829-2008 IEEE Standard for Software and System Test Documentation*. IEEE.

ISO/IEC. (2001). *ISO/IEC 9126-1:2001 Software engineering. Product quality. Part 1: Quality model*. ISO.

ISO/IEC. (2011). *ISO/IEC 25010:2011 Systems and software Quality Requirements and Evaluation (SQuaRE). System and software quality models*. ISO.

ISO/IEC. (2016). *ISO/IEC 25023:2016 Measurement of system and software product quality*. ISO.

ISO/IEC/IEEE. (2013). *ISO/IEC/IEEE 29119 Software and systems engineering. Software testing*. ISO.

ISTQB. (2023). *Certified Tester Foundation Level Syllabus v4.0*. International Software Testing Qualifications Board.

Jia, Y., & Harman, M. (2011). An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, *37*(5), 649–678. https://doi.org/10.1109/TSE.2010.62

Kaner, C., Bach, J., & Pettichord, B. (2002). *Lessons Learned in Software Testing*. Wiley.

Leotta, M., Clerissi, D., Ricca, F., & Tonella, P. (2013). Capture-replay vs. programmable web testing: An empirical assessment during test case evolution. *Proceedings of the 20th Working Conference on Reverse Engineering (WCRE)*, 272–281. https://doi.org/10.1109/WCRE.2013.6671292

Leotta, M., Stocco, A., Ricca, F., & Tonella, P. (2016). ROBULA+: an algorithm for generating robust XPath locators for web testing. *Journal of Software: Evolution and Process*, *28*(3), 177–204. https://doi.org/10.1002/smr.1771

Mesbah, A., van Deursen, A., & Lenselink, S. (2012). Crawling Ajax-based web applications through dynamic analysis of user interface state changes. *ACM Transactions on the Web*, *6*(3), 1–30.

https://doi.org/10.1145/2109205.2109208

Mesbah, A., van Deursen, A., & Roest, D. (2012). Invariant-based automatic testing of modern web applications. *IEEE Transactions on Software Engineering*, *38*(1), 35–53. https://doi.org/10.1109/TSE.2011.28

Myers, G. J., Sandler, C., & Badgett, T. (2011). *The Art of Software Testing*. Wiley.

Nass, M., Alegroth, E., & Feldt, R. (2021). Why many challenges with GUI test automation (will) remain. *Information and Software Technology*, *138*, 106625. https://doi.org/10.1016/j.infsof.2021.106625

Ponce Guerron, A. D., & Naranjo Erazo, R. D. (2021). *Implementacion de un software para la ejecucion de pruebas en aplicaciones web extendiendo la herramienta Selenium. Caso de estudio en el producto Gestor G5 Trust*. Universidad de las Fuerzas Armadas ESPE.

Presler-Marshall, K., Horton, E., Heckman, S., & Stolee, K. T. (2019). Wait, wait. No, tell me. Analyzing Selenium configuration effects on test flakiness. *2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST)*, 7–13. https://doi.org/10.1109/AST.2019.000-1

Pressman, R. S. (2010). *Ingenieria del software: un enfoque practico*. McGraw-Hill.

Pysmennyi, I., Kyslyi, R., & Kleshch, K. (2025). AI-driven tools in modern software quality assurance: an assessment of benefits, challenges, and future directions. *Technology Audit and Production Reserves*, *3*(2), 44–54. https://doi.org/10.15587/2706-5448.2025.330595

Ricca, F., Marchetto, A., & Stocco, A. (2024). *A multi-year grey literature review on AI-assisted test automation*. https://arxiv.org/abs/2408.06224

Romano, A., Song, Z., Grandhi, S., Yang, W., & Wang, W. (2021). An empirical analysis of UI-based flaky tests. *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE)*, 1585–1597. https://doi.org/10.1109/ICSE43902.2021.00141

Runeson, P., & Host, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. https://doi.org/10.1007/s10664-008-

9102-8

Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide*. Scrum.org.

Serna, E., Martinez, R., & Tamayo, P. (2017). Un modelo para determinar la madurez de la automatizacion de las pruebas del software como area de investigacion y desarrollo. *Computacion y Sistemas*, *21*(2), 337–352. https://doi.org/10.13053/cys-21-2-2214

Serna Montoya, E. (2013). *Prueba funcional del software: un proceso de verificacion constante*. Instituto Tecnologico Metropolitano. https://doi.org/10.22430/9789588743417

Sommerville, I. (2011). *Ingenieria de software*. Pearson Educacion.

Wohlin, C., Runeson, P., Host, M., Ohlsson, M. C., Regnell, B., & Wesslen, A. (2012). *Experimentation in Software Engineering*. Springer.

Yin, R. K. (2018). *Case Study Research and Applications: Design and Methods*. Sage.

Apéndices