COMPARACIÓN DE HERRAMIENTAS DE TESTING AUTOMATIZADO

Agudelo Acevedo, Nicole

INSTITUCIÓN UNIVERSITARIA PASCUAL BRAVO

FACULTAD DE INGENIERÍA

TECNOLOGÍA EN DESARROLLO DE SOFTWARE

MEDELLÍN

2025

Tabla de Contenido

índice de Tablas

Introducción

La automatización de las pruebas end-to-end (E2E) en aplicaciones web se ha convertido en un pilar fundamental dentro del aseguramiento de calidad, ya que permite simular flujos completos del usuario, minimizar la intervención manual y aumentar la productividad en el ciclo de desarrollo. En este plano, ya existen diferentes herramientas para hacer frente a las demandas específicas de los equipos de desarrollo como lo son Selenium, Cypress y Playwright las cuales poseen diferencias a nivel de arquitectura, modelo de ejecución y soporte tecnológico. Ya que la literatura técnica establece que la efectividad de una herramienta depende tanto del entorno en que es aplicada como de las características del proyecto, es por demás oportuno analizarlas desde aspectos comunes y cuantificables.

Este artículo se centra en evaluar las tres herramientas a través de tres aspectos clave: Usabilidad (curva de aprendizaje y experiencia de desarrollo), Velocidad de ejecución (tiempo medio y estabilidad del proceso) y Detección de errores (número de errores encontrados y consistencia de resultados). El objetivo es proporcionar un marco de comparación para saber qué herramienta se desempeña mejor para testing web E2E en ambientes reales de desarrollo.

El texto se estructura de la siguiente manera: el capítulo 1 establece el planteamiento del problema y la pregunta de investigación, el capítulo 2 expone la justificación ,el capítulo 3 describe los objetivos, el capítulo 4 describe el marco teórico y conceptual en relación con el testing automatizado y herramientas E2E; el capítulo 5 describe el marco legal, el capítulo 6 explica criterios de comparación y resultados, el capítulo 7 metodología y por último el Capítulo 8 concluye con las conclusiones resultantes de la investigación.

1. Planteamiento del Problema

1.1 Descripción

En el desarrollo de software, asegurar la calidad del producto es una tarea crucial que requiere detectar tempranamente errores para evitar costos elevados y retrasos en la entrega. Las pruebas automatizadas son uno de los tipos de testing que han surgido como una solución eficiente para mejorar la cobertura y fiabilidad del testing en comparación con las pruebas manuales. Sin embargo, la amplia gama de herramientas disponibles puede dificultar la elección más adecuada para los equipos de desarrollo.

Actualmente, frameworks como Selenium, Cypress y Playwright lideran el mercado de pruebas end-to-end (E2E) para aplicaciones web, permitiendo la validación de flujos completos de usuario. Sin embargo, más allá de su popularidad, existen discrepancias estructurales que afectan directamente el rendimiento del ciclo de desarrollo. Estas herramientas no son equivalentes en parámetros críticos: la facilidad de uso varía según la complejidad de configuración y la curva de aprendizaje del lenguaje; la velocidad de ejecución fluctúa dependiendo del manejo de la asincronía y los recursos del sistema; y la fiabilidad presenta diferencias notables en la tasa de falsos positivos o negativos reportados.

Esto puede afectar la eficiencia del proceso de pruebas y, en última instancia, la calidad del software. Por lo tanto, se hace evidente la necesidad de realizar una evaluación comparativa que ayude a los desarrolladores y a las empresas a tomar decisiones más informadas sobre qué herramienta de testing automatizado es la más apropiada, considerando diferentes criterios técnicos y prácticos.

1.2 Formulación

¿Cuál de las herramientas de automatización E2E para aplicaciones web (Selenium, Playwright y Cypress) ofrece un mejor desempeño en términos de facilidad de uso, velocidad y precisión en la detección de errores?

2. Justificación

La automatización de pruebas es ahora parte fundamental del desarrollo de software moderno, sobre todo en entornos que emplean metodologías ágiles y DevOps, donde entregas continuas demandan ciclos rápidos y fiables de verificación. Las pruebas automatizadas contribuyen a mejorar la detección temprana de defectos, incrementar la cobertura de pruebas y asegurar la estabilidad funcional del software antes de su liberación. Particularmente, las pruebas end-to-end (E2E) para aplicaciones web entregan mucho valor, ya que se ejecutan en flujos reales de usuarios y pueden validar la interacción integrada de varios componentes del sistema.

No obstante, el seleccionar una herramienta adecuada para la automatización E2E es una tarea compleja, ya que existen múltiples opciones con diferentes arquitecturas, soportes tecnológicos y experiencia de uso. En este trabajo se consideran tres herramientas de uso común en la industria para realizar pruebas E2E en aplicaciones web: Selenium, Cypress y Playwright, todas ellas con una trayectoria consolidada y un nivel alto de adopción en ambientes empresariales. Cada una tiene diferencias significativas en facilidad de uso, velocidad de ejecución, y precisión para encontrar errores, que son factores críticos para los equipos de desarrollo y aseguramiento de calidad.

La ventaja de aplicar este tipo de herramientas no sólo es mejorar la calidad del software, sino también la posibilidad operativa: acortan tiempos de testeo, reducen costos asociados a fallas tardías y facilitan la integración en pipelines CI/CD. Sin embargo, su introducción supone desafíos como la curva de aprendizaje, la necesidad de infraestructura adecuada, el mantenimiento de los scripts y los peligros de inestabilidad relacionados con pruebas web automatizadas, por tanto, se requiere un análisis comparativo exhaustivo para decidir qué herramienta proporciona el mejor balance entre beneficios y costos operativos.

Con este estudio se facilitará que desarrolladores, equipos de QA u organizaciones puedan tomar decisiones con base sólida en la elección de herramientas E2E, brindando un respaldo técnico que permita una aplicación eficiente y sostenible de los procesos de automatización en proyectos de desarrollo de software web.

3. Objetivos

3.1 Objetivo General

Comparar herramientas de testing automatizado en términos de facilidad de uso, velocidad de ejecución y precisión en la detección de errores, con el fin de determinar sus principales ventajas y limitaciones en el contexto del desarrollo de software.

3.2 Objetivos Específicos

- Seleccionar las herramientas de testing a analizar
- Analizar la facilidad de uso de las herramientas de testing seleccionadas
- Examinar la precisión en la detección de errores de cada herramienta, identificando su capacidad para reconocer fallos en aplicaciones web.
- Comparar las ventajas y limitaciones de las herramientas de testing seleccionadas (en relación con los criterios seleccionados).

4. Marco Teórico

4.1 Fundamentos del Testing Automatizado

4.1.1 ¿Qué es el testing automatizado?

El testing automatizado es la realización de pruebas a través de herramientas que trabajan sin una intervención humana directa para comprobar si el software se comporta en determinados entornos de forma repetible y controlada. Los autores, Myers, Sandler y Badgett (2011) explican que la automatización posibilita que se corran largas secuencias de verificación eficientemente, disminuyendo la variabilidad introducida por el componente humano y haciendo los resultados más consistentes; Bertolino (2007) comenta además que la automatización es de suma importancia en proyectos que precisan de ciclos rápidos de retroalimentación, tales como aquellos basados en metodologías ágiles e integración continua.

Desde el punto de vista de la ingeniería, la automatización ayuda a aumentar la cobertura de prueba, reduce el esfuerzo manual, y permite detectar defectos más temprano, lo que resulta en ahorros considerables de tiempo y recursos en etapas posteriores del desarrollo (Ammann & Offutt, 2016). Como resultado, la automatización ha pasado a ser un requisito para garantizar la calidad del software en sistemas complejos y de alta criticidad.

4.1.2 Tipos de pruebas automatizadas

- Pruebas Unitarias: Las pruebas unitarias son porciones de código diseñados para comprobar que una funcionalidad de una aplicación esté funcionando como se espera. Esto quiere decir que si tenemos un escenario donde se almacena un nuevo usuario en una tabla, la prueba unitaria tiene que simular este guardado del usuario con el mismo método que lo hace en la práctica real de “crearNuevoUsuario”. En estas pruebas los implementadores de software observan la interfaz y la especificación de un componente, suministrando la documentación del desarrollo del código donde este se prueba, claro que de forma independiente antes de pasar a otra unidad o escenario. Las pruebas unitarias verifican el funcionamiento de componentes aislados del software, como funciones o módulos específicos, con el fin de identificar fallos en la lógica interna antes de que el sistema se integre. Este tipo de pruebas ofrece ventajas como la detección temprana de errores y el aseguramiento de que cada bloque de código cumple con su propósito, reduciendo costos asociados a defectos tardíos. (Garzon Guerrero, 2022)

- Pruebas funcionales: Las pruebas funcionales se orientan a comprobar que el software cumple con los requerimientos establecidos desde la perspectiva del usuario final. No se centran en la estructura interna, sino en que el sistema realice correctamente las operaciones esperadas. Garzón Guerrero (s. f.) indica que la automatización de pruebas funcionales, junto con las unitarias y de regresión, permite establecer procesos de aseguramiento de calidad acordes con metodologías ágiles como Scrum y DevOps. De esta manera, se garantiza que las funcionalidades críticas se mantengan operativas bajo diferentes escenarios de uso. (Garzon Guerrero, 2022)

- Pruebas de regresión: Las pruebas de regresión buscan confirmar que las nuevas modificaciones no afectan negativamente las funcionalidades ya implementadas. Este tipo de pruebas, altamente propenso a la automatización, asegura la estabilidad del software a lo largo de su ciclo de vida. Cortés Pabón (2020) enfatiza que las pruebas de regresión automatizadas reducen el tiempo de entrega de nuevas versiones de software, ya que permiten verificar de forma continua el impacto de los cambios realizados. En este sentido, se consideran una práctica indispensable en entornos de integración y entrega continua. (Cortes Pabon, 2020)

- Pruebas de integración: Evalúan la interacción entre componentes o servicios. En arquitecturas modernas de microservicios, estas pruebas son críticas para detectar incompatibilidades entre APIs, flujos de datos y dependencias externas. Investigaciones recientes proponen estrategias de ordenamiento de pruebas que consideran el acoplamiento de control para reducir costos y mejorar eficiencia. (Camilli, Guerriero, Janes, Russ, & Russo, 2022)

- Pruebas de Sistema: verifican el comportamiento integral de la aplicación en un entorno similar al de producción. Estas evaluaciones incluyen tanto aspectos funcionales como no funcionales, asegurando que el software cumpla con los requisitos del negocio y los estándares de calidad establecidos. Su automatización permite obtener resultados consistentes y reproducibles, reduciendo el riesgo de fallos por diferencias entre entornos de prueba y producción. (Duqino Sanchez, 2020)

- Pruebas end-to-end (E2E): Las pruebas E2E validan el flujo completo de usuario, desde la interfaz hasta el backend y la base de datos, asegurando que todos los componentes funcionen correctamente en conjunto. Su automatización resulta compleja por la necesidad de replicar escenarios de producción, pero permite detectar fallos críticos en la experiencia del usuario. Estudios recientes en empresas han mostrado reducciones drásticas en los tiempos de ejecución de pruebas al automatizar E2E, pasando de días a minutos. (Sanchez Carranza, 2024)

4.2 Herramientas De Testing Automatizado

Existen muchas herramientas que le permiten automatizar pruebas con fines específicos, como validar interfaces web, ejecutar pruebas unitarias o medir el rendimiento, analizar APIs, entre otras, o también para encontrar vulnerabilidades en la seguridad. Cada producto tiene un alcance técnico específico y diferencias en la arquitectura, en los lenguajes soportados, en el licenciamiento y en la integración con los sistemas de integración continua. En la Tabla 1 se muestran algunas de las herramientas de más uso en la industria, junto con una descripción general y su campo de aplicación.

Tabla 1.Herramientas de testing automatizado: descripción, alcance y características técnicas

Herramienta	Descripción	Ámbito de aplicación	Lenguajes soportados	Navegadores soportados	Sistemas operativos	Tipo de pruebas
Selenium	Framework de automatización de pruebas orientado a interfaces gráficas en navegadores web. Permite simular interacciones de usuario como clics, escritura y navegación, soportando múltiples lenguajes (Java, Python, C#) y navegadores. Es una de las herramientas más utilizadas en QA por su flexibilidad e integración con otros frameworks. Sin embargo, su configuración inicial puede ser compleja, y no incluye características avanzadas como informes detallados o depuración visual de forma nativa. (Chetu, 2023)	Frontend	Java, Python, C#, JS, Ruby	Chrome, Firefox, Edge, Safari	Windows, macOS, Linux	E2E / UI
Cypress	Herramienta moderna para pruebas de frontend e integración con backend. Ofrece ejecución rápida en tiempo real, depuración sencilla y un enfoque centrado en la experiencia del desarrollador. Está pensada para aplicaciones modernas basadas en JavaScript y	Frontend	JavaScript, TypeScript	Chrome, Edge, Firefox	Windows, macOS, Linux	E2E / Integración

	entornos CI/CD. (Tivit, 2022)					
Playwright	Herramienta desarrollada por Microsoft que permite automatizar pruebas de UI en navegadores modernos (Chromium, Firefox y WebKit) y dispositivos móviles lo que garantiza la compatibilidad entre navegadores y una cobertura amplia. Su arquitectura se destaca por ejecutar pruebas fuera del navegador, lo que le permite controlar de forma más eficiente el proceso. (Tivit, 2022)	Frontend	JS, TS, Python, Java, .NET	Chromium, Firefox, WebKit	Windows, macOS, Linux	E2E / UI
Postman	Plataforma muy usada para pruebas de APIs REST y SOAP. Permite la creación de colecciones de pruebas, automatización de peticiones y validación de respuestas. Postman permite la automatización de pruebas de API a través de "Colecciones" que agrupan solicitudes y permiten la adición de scripts de validación con JavaScript. Además, se integra fácilmente con pipelines de CI/CD (Integración Continua y Despliegue Continuo), lo que lo convierte en una herramienta versátil y poderosa para asegurar la calidad de las APIs. (Postman, 2025)	Backend	JavaScript	—	Windows, macOS, Linux	API / Integración
JUnit + Spring Test	Framework para pruebas en Java, especialmente potente en aplicaciones que usan el ecosistema Spring. Al combinarse con Spring Test, se convierte en una herramienta fundamental para probar aplicaciones desarrolladas con el framework de Spring. Spring Test extiende JUnit, facilitando las	Backend	Java	—	Windows, macOS, Linux	Unit / IntegraciónI

	pruebas de integración y permitiendo inyectar dependencias y simular el entorno de la aplicación, lo que es esencial para probar componentes en un contexto más realista sin tener que levantar toda la aplicación. (junit, s.f.)					
Pytest + Requests	Combinación de herramientas en Python que permite realizar pruebas unitarias e integración. Pytest ofrece un marco flexible para escribir casos, mientras que Requests facilita pruebas HTTP para APIs. Su sencillez y extensibilidad lo hacen popular en entornos ágiles. Al usarse en conjunto con la biblioteca Requests, que es el estándar de facto para hacer peticiones HTTP en Python, Pytest se convierte en una herramienta muy efectiva para las pruebas de API. (xuannguyentruong, 2024)	Backend	Python	—	Windows, macOS, Linux	Unit / API
JMeter	Herramienta de Apache para pruebas de rendimiento, carga y estrés en aplicaciones web. Permite simular múltiples usuarios concurrentes y medir métricas como tiempo de respuesta, throughput y estabilidad. Su objetivo principal es simular un gran número de usuarios concurrentes en un sistema para medir su rendimiento bajo diferentes niveles de carga. JMeter es una herramienta crucial para identificar cuellos de botella y asegurar que las aplicaciones son escalables y estables en entornos de producción. (Apache, s.f.)	Backend / Performance	Java	---	Windows, macOS, Linux	Carga / Rendimiento

SQLMap	Herramienta especializada en pruebas de seguridad en bases de datos. Automatiza la detección y explotación de vulnerabilidades de inyección SQL, permitiendo evaluar la robustez de sistemas frente a ataques. Puede detectar automáticamente las vulnerabilidades, identificar el tipo de base de datos, extraer tablas, columnas y datos, e incluso acceder al sistema de archivos del servidor. Es una herramienta indispensable en el ámbito del testing de seguridad, ya que ayuda a identificar puntos débiles antes de que un ataque real pueda ocurrir. (Carte, 2024)	Seguridad	-	—	Windows, macOS, Linux	Pentesting

4.3. Enfoques Teóricos Relacionados

4.3.1 Teoría de calidad del software

La calidad del software es uno de los conceptos más importantes en ingeniería de software, ya que se utiliza para determinar hasta qué punto un sistema cumple con sus requisitos funcionales, no funcionales y las necesidades de los usuarios. En este estudio, esta teoría es especialmente importante porque la automatización de pruebas y más específicamente las herramientas de testing E2E, impacta directamente en la garantía de atributos críticos de calidad. En virtud de ello, en el transcurso de esta sección se presentan los principales referentes teóricos y características del modelo ISO/IEC 25010 que sustenta teóricamente el análisis comparativo de las herramientas elegidas.

El modelo ISO/IEC 25010:2011 desarrollado por la Organización Internacional de Normalización (ISO) y la Comisión Electrotécnica Internacional (IEC) describe ocho características del producto para medir la calidad de un producto software (ISO 25000, 2022). La automatización de pruebas afecta significativamente a atributos como la fiabilidad, la eficiencia del desempeño y la usabilidad, que están directa o indirectamente relacionados con las métricas analizadas en este

trabajo, es decir, Tiempo Total de Ejecución (TTES), tiempo promedio, percentil 95 (p95), estabilidad de resultados y dificultad para la implantación.

Las siguientes son las sospechas más comunes al evaluar herramientas E2E:

- **Usabilidad (Usability):** Según la ISO 25000 (2022), la usabilidad es el grado en que un producto puede ser utilizado por tipos específicos de usuarios para alcanzar sus objetivos con efectividad, eficiencia y satisfacción. Para el estudio, esta cualidad se asocia con facilidad de uso, y se consideran aspectos tales como curva de aprendizaje, claridad de la documentación, existencia de recursos y simplicidad para escribir scripts. Estos son factores para comparar qué tan bien el desarrollador se siente utilizando Selenium vs Cypress vs Playwright.
- **Fiabilidad (Reliability):** Corresponde a la capacidad para que el sistema funcione bien en particular condiciones durante un tiempo determinado (ISO 25000, 2022). La más destacada subcaracterística es la madurez que es la estabilidad de la herramienta ante fallos internos. En este estudio, la precisión se analiza utilizando métricas tales como la consistencia de la ejecución, la estabilidad del tiempo empleado por los scripts y la no presencia de fallos aleatorios en ejecuciones consecutivas.
- **Eficiencia del Desempeño (Performance Efficiency):** Conforme a ISO25000 (2022) esta característica determina como evaluar la performance de un Software con relación a su uso de recursos. La subcaracterística más relevante es el comportamiento en el tiempo, el cual está relacionado de manera directa con el tiempo de ejecución y estabilidad. Para medir esta característica se usarán indicadores como por ejemplo el tiempo total de ejecución de secuencias (TTES), tiempo promedio por prueba y el p95, con los cuales se podrá realizar una comparativa objetiva entre Selenium, Cypress y Playwright.

4.3.2 Ciclos DevOps e integración continua

En entornos DevOps, la integración continua (CI) y la entrega continua (CD) requieren la ejecución constante de pruebas automatizadas como parte del pipeline de despliegue. Herramientas como Cypress, Playwright y Katalon ofrecen integraciones nativas con Jenkins, GitHub Actions o Azure Pipelines, lo que las hace ideales para estos modelos de desarrollo

(Qalified, 2024).

- La Integración Continua (CI), popularizada por Martin Fowler, es una práctica en la que los desarrolladores integran el código en un repositorio compartido con frecuencia. Cada integración se verifica automáticamente mediante una compilación y una suite de pruebas automatizadas. En este contexto, la velocidad de ejecución y la fiabilidad de las herramientas de testing son críticas, ya que la meta es obtener retroalimentación rápida sobre los cambios (Fowler, 2023)

- DevOps va más allá de la integración de código, uniendo los equipos de desarrollo y operaciones para optimizar la entrega de software. La automatización del testing es un componente central de los pipelines de CI/CD, permitiendo que los despliegues sean rápidos, repetibles y fiables, lo que reduce el riesgo y permite una respuesta más ágil a las necesidades del mercado (Fowler, 2023).

4.4 UX & UI

4.4.1 UX (Experiencia de Usuario (User Xperience)

La experiencia de usuario (UX por sus siglas en inglés) consiste en la experiencia general que adquiere el usuario al que va destinada la aplicación, incluyendo la percepción, las emociones que le provoca y la respuesta que genera. La experiencia de usuario se mide principalmente por la utilidad, resolución, eficiencia, fiabilidad y la percepción de los usuarios de los estímulos, la fiabilidad y la innovación con respecto a la respuesta emocional humana al interactuar con el sistema. (Santos, 2024)

Los principios para establecer si se está desarrollando una correcta experiencia de usuario dentro de nuestra aplicación:

- Usabilidad: La usabilidad hace alusión a la facilidad con la que los usuarios interactúan con un producto o servicio, de manera que resulta intuitivo para quien lo usa y ahorra tiempo en el aprendizaje de este.
- Accesibilidad: Un producto debe cumplir con las expectativas de todos los públicos,

incluyendo a aquellos que presenten alguna discapacidad.

- Eficiencia: Las tareas a desarrollar deben realizarse en el menor tiempo posible, empleando todos los componentes visuales como botones e iconos de manera útil y en su totalidad, permitiendo ahorrar tiempo en llevar a cabo las tareas.
- Visibilidad: Los elementos que se muestran en la aplicación deben cumplir con un propósito, de manera que deben ser lo más claro posible, evitando sobrecargar la aplicación y provocar que el usuario pierda de vista el elemento con el que interactúe.
- Intuición: Las tareas y los elementos visuales que las componen deben ser lo más claros y concisos posible, de manera que no represente un problema de entendimiento a los usuarios, estableciendo así una comunicación visual efectiva entre lo que se le muestra al usuario y la percepción ante lo mostrado.
- Credibilidad: Las aplicaciones deben mantener una retroalimentación con los usuarios cuando llevan a cabo una acción, de manera que se complemente y refuerce el resultado de las acciones mediante elementos visuales como mensajes, ventanas emergentes, entre otros.

Además de la experiencia del usuario final, en el marco de este estudio es preciso tener en cuenta la UX para el equipo de desarrolladores, la cual se define como la experiencia de los desarrolladores al interactuar con las herramientas de automatización E2E. En este sentido, la usabilidad se interpreta como la claridad en la documentación, la facilidad de instalación, la intuitividad del lenguaje de scripting, la existencia de ejemplos y la posibilidad de depurar eficientemente. Por ello, la experiencia de desarrollo con herramientas como Selenium, cypress o Playwright es un aspecto clave para la UX, ya que una mejor herramienta facilita la adopción, acorta el tiempo de aprendizaje y permite una mayor productividad a la hora de crear, ejecutar y mantener pruebas automatizadas.

4.4.2 UI (Interfaces de Usuario (User Interface)

La Interfaz de Usuario o User Interface (UI), se refiere a todo aquello con lo que los usuarios interactúan directamente (la capa externa de un producto o servicio digital). Es lo que ve y toca en una página web, una aplicación o un dispositivo cualquiera. La UI se encarga principalmente

a los elementos visuales con los que se desenvuelve el usuario dentro de un producto de software, es decir, los colores, la tipografía, los elementos de contenido tales como iconos, formulario, botones, por lo cual se asocia directamente a la UX, puesto que se emplean en conjunto al momento de desarrollar aplicaciones permitiendo optimizarla y llevar a cabo las tareas del sistema (Santos, 2024)

poseen características específicas que nos permiten ofrecer la mayor claridad de las tareas a desarrollar, las cuales son:

- Atracción visual: La apariencia de la interfaz es muy importante para los usuarios, puesto que es clave para que un usuario decida interactuar en ella.
- Claridad: La forma en la que se establece la información debe ser lo más clara y concisa posible de manera que se eviten errores humanos y del software.
- Coherencia: Los elementos dentro de la interfaz deben mantenerse unidos y manteniendo un propósito de manera que los usuarios puedan generar un patrón intuitivo y la interacción sea cada vez más sencilla.
- Flexibilidad: Los interfaces deben cambiar y adecuarse a las necesidades de los usuarios.

4.5 Parámetros: facilidad de uso, velocidad de ejecución, precisión / detección de errores

4.5.1 Facilidad de uso

La facilidad de uso, también llamada usabilidad en el ámbito UX/UI, se refiere a la capacidad de un sistema o herramienta para permitir que los usuarios realicen sus tareas de manera efectiva, eficiente y satisfactoria. La efectividad mide qué tanto logra el usuario cumplir sus objetivos en el sistema; la eficiencia evalúa los recursos y tiempo que requiere; y la satisfacción analiza qué tan agradable o frustrante resulta la experiencia (Nielsen, 2021). Una de las formas más utilizadas para medir la facilidad de uso es la **tasa de éxito en tareas** (*Task Success Rate*), que corresponde al porcentaje de usuarios que logran completar una tarea sin errores (Userlytics, 2024)

La facilidad de uso se analizará desde la perspectiva de la curva de aprendizaje inicial y la

complejidad técnica requerida para ejecutar una primera prueba exitosa. Para este fin se emplea el indicador Time To First Test (TTFT), una métrica que cuantifica el tiempo que tarda un desarrollador en instalar, configurar y ejecutar la primera prueba funcional E2E en cada herramienta (Nguyen et al., 2020).

Fórmula del TTFT= Tfin−Tinicio

Donde:

- Tinicio: momento en que se inicia la instalación y configuración del framework.
- TfTfin: instante en el que se ejecuta con éxito el primer caso de prueba.

Esta métrica permite comparar objetivamente la **usabilidad técnica inicial** entre herramientas y su impacto sobre la adopción por parte del equipo de desarrollo.

4.5.2 Velocidad de ejecución

La velocidad de ejecución mide qué tan rápido puede un usuario completar una tarea o qué tan eficiente resulta la interfaz en su funcionamiento. El indicador más extendido es el tiempo en la tarea (Time on Task), que registra el tiempo transcurrido desde el inicio hasta la finalización de una tarea. Este valor, obtenido a través de cronómetros o registros automáticos, permite calcular promedios y compararlos entre usuarios o versiones del sistema. Un tiempo excesivo puede señalar problemas de navegación, flujos confusos o elementos poco intuitivos. (YOUX, s.f.)

El **Time on Task (ToT)** mide el tiempo total que tarda un usuario en completar una tarea específica sin importar el número de intentos.

Fórmula del ToT=Tfin−Tinicio

Donde:

- Tinicio: momento exacto en que el usuario comienza la tarea.
- Tfin: momento en que finaliza la tarea de manera satisfactoria.

Otra forma de evaluar la velocidad es comparando los tiempos entre usuarios expertos y novatos. Esta comparación refleja la curva de aprendizaje de la interfaz: si los novatos tardan mucho más, puede ser señal de que la interfaz no es lo suficientemente intuitiva. También resulta útil medir el número de pasos o clics necesarios para completar una tarea, ya que a menor cantidad de interacciones, mayor fluidez percibida (YOUX, s.f.).

4.5.3 Detección de errores

La precisión en la interacción está directamente relacionada con la cantidad y tipo de errores que cometen los usuarios al interactuar con un sistema. Una métrica fundamental es la tasa de errores (Error Rate), que mide el número de errores cometidos frente al total de tareas u oportunidades de error. Puede expresarse como porcentaje o como número promedio de errores por usuario. Este indicador es clave para identificar qué tan confiable y libre de fallos es la experiencia (Jeon, 2025)

Fórmula del Error Rate=E/O

Donde:

- E = número total de errores cometidos.
- O = número total de oportunidades de error (por ejemplo: pasos, acciones o tareas).

Para expresarla en porcentaje:

Error Rate(%)=(EO)×100

Los errores también pueden clasificarse en críticos y menores. Los errores críticos son aquellos que impiden la finalización de una tarea, mientras que los menores solo afectan la eficiencia o generan pequeñas frustraciones. Esta clasificación permite priorizar qué problemas de usabilidad deben corregirse con urgencia. Asimismo, es importante evaluar la detectabilidad de los errores: qué tan rápido y claramente el sistema comunica al usuario que algo salió mal, y si este puede corregirlo sin asistencia. Un buen sistema debe proporcionar mensajes de error claros y guiar al usuario hacia la solución (UX & Usability Toolkit, 2025).

En pruebas de usabilidad se recomienda observar no solo el número de errores, sino también si

los usuarios fueron capaces de **darse cuenta de que se equivocaron**, y cómo reaccionaron para solucionarlo. De esta forma se obtiene una visión más completa de la precisión y de la calidad del feedback que ofrece el sistema (Jeon, 2025).

4.6 Evaluación de herramientas de testing automatizado

La elección de una herramienta de test automático conlleva la fijación de criterios claros que dejen hacer comparaciones entre diferentes opciones. Entre los aspectos clave están los fácil de usar, su velocidad y la precisión en la detección de errores, porque esos son los que muestran el impacto verdadero de la herramienta en la eficiencia de las pruebas. La facilidad de uso se evalúa primordialmente a partir de indicadores como la curva de aprendizaje, la claridad de la documentación, los tipos de interfaces que existen, la compatibilidad con los entornos de desarrollo. En este ámbito, la literatura también indica que la percepción de los equipos de trabajo medida mediante encuestas, escalas de satisfacción y similares constituye un input importante para determinar en qué medida la herramienta resulta accesible en situaciones de uso habitual (Pacanchique, 2024).

A su vez, la velocidad de ejecución se establece a partir de la determinación de los tiempos medios que llevan pruebas equivalentes en diferentes herramientas, en condiciones controladas de hardware, red y volumen de datos, y que permite comparar la respuesta que ofrecen las plataformas para un conjunto igual de casos de prueba (Pérez, 2015). La medición de números en este tipo de test debe hacerse bajo condiciones iguales y asegurar que se repitan las pruebas suficientes veces para conseguir valores firmes y comparables (Pérez, 2015). En las pruebas para evaluar el peso y la calidad la velocidad también se mide como la habilidad de simular usuarios al mismo tiempo y transacciones por segundo; esto muestra el rango de trabajo de la app y su desempeño antes del riesgo de tener mucha carga (Zapata & Cardona, 2011).

Por último, la precisión depositada en la detección de errores constituye un criterio sumamente importante, puesto que permite evaluar la fiabilidad de los resultados que se obtienen. Dicha precisión se mide mediante la comparación establecida entre la salida de las pruebas automatizadas y una “verdad de referencia” que fue elaborada con las pruebas manuales exhaustivas y/o el catálogo de defectos ya documentados. La precisión es un parámetro que Hidalgo Reyes et al. (2020) obtienen a raíz del número de falsos positivos (errores reportados

que no existen) frente a falsos negativos (errores reales no detectados), el cual además puede asignar un rango de fiabilidad para cada una de las herramientas. Desde este punto de vista, una herramienta automatizada con alta sensibilidad será aquella que consiga detectar los defectos críticos con poca generación de falsos resultados, asegurando así una mayor estabilidad y seguridad en el software.

Tabla 2.Indicadores para la evaluación de herramientas de testing automatizado

Parámetro	Indicador de medición	Rango de referencia / Punto de comparación	Fuente
Facilidad de uso	Curva de aprendizaje (tiempo en horas requerido para la capacitación inicial).	Comparación entre equipos de testers con similar experiencia técnica.	Pacanchique (2024)
	Nivel de satisfacción del usuario (escala Likert 1–5).	Encuestas aplicadas a los equipos tras la implementación.	Cubas Montenegro (2015)
	Compatibilidad con entornos y frameworks de desarrollo.	Número de integraciones soportadas frente a la competencia.	Pérez (2015)
Velocidad de ejecución	Tiempo promedio de ejecución de un conjunto de casos de prueba (en segundos o minutos).	Promedio obtenido en al menos 10 ejecuciones bajo condiciones idénticas.	Pérez (2015)
	Capacidad de concurrencia (usuarios virtuales soportados sin degradación).	Comparación de transacciones por segundo frente a herramientas rivales.	Zapata & Cardona (2011)
	Consumo de recursos del sistema (CPU, memoria).	Porcentaje de utilización comparado en condiciones de carga máxima.	Hidalgo Reyes et al. (2020)
Precisión en detección	Tasa de falsos positivos (errores reportados que no existen).	Comparación con resultados de pruebas manuales exhaustivas.	Hidalgo Reyes et al. (2020)

	Tasa de falsos negativos (errores no detectados).	Número de fallos omitidos respecto a la base de errores conocida.	Hidalgo Reyes et al. (2020)
	Cobertura de pruebas (porcentaje de casos ejecutados exitosamente).	Comparación del total de casos ejecutados frente al plan de pruebas.	Pérez (2015)

El hecho de incluir estos indicadores ayuda a generar un marco comparativo completo, que contemple las diferentes dimensiones, por un lado aquéllas de la parte técnica, como pudiera ser: tiempos de las pruebas, concurrencia, cobertura, así como por el otro aquéllas de tipo perceptual, como pudiera ser la satisfacción del usuario o la percepción de la facilidad de uso, ayudando a poder llevar a cabo un análisis que aguante un equilibrio de las herramientas de testing automatizado. Y es que este hecho concuerda con la necesidad que distintos autores apuntan de poder contemplar la eficiencia desde la parte técnica, así como la usabilidad desde aguí parte práctica en lo que concierne al uso de tecnologías de aseguramiento de la calidad (Hidalgo Reyes et al., 2020) ya que considerarlas de manera separada puede llevarnos a pensar en una evaluación única y exclusivamente en cuanto a rapidez o exactitud, pudiendo llegar al hecho de pensarlas como si fuesen un valor más determinado por el uso de una herramienta o el uso efectivo por parte de los equipos de desarrollo.

En conclusión, tanto el parámetro de la facilidad de uso como el de la velocidad de ejecución y la precisión en la detección de errores se constituyen en referencias básicas para realizar comparaciones adecuadas entre las herramientas de testing automatizado, y esto es debido a que al ser medidos con indicadores claros y reproducibles, permiten extraer las fortalezas y debilidades por cada alternativa, así como también la evidencia que valida las decisiones estratégicas en el desarrollo de software. La sistematización de tal forma de los criterios garantiza que la elección de una herramienta no se realice en base a opiniones aisladas, sino en base a resultados comprobables; lo que garantiza una mayor fiabilidad en los procesos de prueba y, en consecuencia, una entrega de una mejor calidad de los productos al mercado (Zapata & Cardona, 2011).

4.6.1. Criterios de comparación (métricas, métodos y diseño de evaluación)

Con el fin de comparar de forma objetiva y reproducible las herramientas de testing automatizado, se establecen criterios operativos con indicadores medibles, métodos de recolección de datos, y un diseño experimental que controle sesgos. Los criterios se agrupan en siete dimensiones: Capacidades técnicas, Experiencia de desarrollo, Calidad y estabilidad, Accesibilidad, Seguridad y cumplimiento, Integraciones y ecosistema, y Costos y licenciamiento. Cada dimensión incluye métricas, unidad de medida, procedimiento y forma de puntuar.

A) Capacidades técnicas

Evalúa el alcance funcional de la herramienta y su adecuación al stack del proyecto.

- **Soporte de frameworks, lenguajes y navegadores**
  Indicadores: Nº de lenguajes soportados; Nº de frameworks/SDKS; Nº de navegadores/engines (Chromium, Firefox, WebKit) y canales (stable/beta).
  Medición: Matriz de compatibilidad verificada con un proyecto mínimo por cada ítem.

- **Tipos de pruebas** (unitarias, integración, E2E, API, móvil/webview, visual-regression).
  Indicador: Cobertura de tipos (sí/no) y nivel de madurez (básico/medio/avanzado).

- **Paralelización y aislamiento**
  Indicadores: Concurrencia máxima nativa; throughput (casos/min en N hilos); soporte de sharding y retries por test.
  Medición: Suite estándar ejecutada 30 veces en condiciones controladas.

- **Mocking y control de red/tiempo**
  Indicadores: Interceptación de requests/responses; fixtures; control de reloj; emulación de geolocalización y permisos.
  Medición: Verificación con casos diseñados (latencias artificiales, time-freeze).

- **Depuración y trazabilidad**
  Indicadores: Artifacts generados (screenshots, video, HAR, logs detallados); profundidad de stack-trace (n niveles); “step trace” o timeline.

- **Rendimiento base**
  Indicadores: TTES (Time To Execute Suite): mediana y p95; Overhead en headless vs headed; Warm-up inicial.
  Medición: Mismo hardware/red; 5 corridas de calentamiento + 30 corridas válidas.

B) Experiencia de desarrollo

Mide la facilidad para instalar, aprender, escribir, mantener y diagnosticar pruebas.

- **Time To First Test (TTFT)**
  Indicador: Minutos desde instalación hasta primer test verde (incluye instalación, scaffold y ejecución).
- **Curva de aprendizaje y documentación**
  Indicadores: Puntuación SUS/ASQ del equipo (Likert 1–5); completitud de docs (guías, ejemplos, troubleshooting); tiempo promedio para encontrar solución a 3 fallos comunes (MTTS – Mean Time To Solution).
- **Ergonomía de authoring**
  Indicadores: Autocompletado/typings; assertions integradas; fixtures y PageObjects; DX en IDE (extensiones).
- **Ecosistema y comunidad**
  Indicadores: Nº de plugins oficiales/terceros; frecuencia de releases; actividad de issues/discusiones; tiempo medio de respuesta del maintainer.

C) Calidad y estabilidad

Cuantifica la confiabilidad de las ejecuciones y la eficacia al capturar defectos.

- **Flakiness**
  Indicadores: **FR** (Flaky Rate) = tests con resultados inconsistentes / total de tests (en 30 corridas).
- **Robustez ante cambios**
  Indicadores: % de tests que sobreviven a cambios superficiales de UI (renombrado de selectores, delays controlados).

- **Detección de fallos (eficacia)**
  Indicadores: **Precisión** (TP/(TP+FP)) y **Recall** (TP/(TP+FN)) comparando contra un "ground truth" de defectos sembrados; **F1**.

- **Diagnóstico**
  Indicadores: % de fallos con causa probable identificable en ≤5 min usando artefactos de la herramienta.

D) Accesibilidad

Considera soporte nativo o integrable para pruebas a11y.

- **Integración a11y**
  Indicadores: Soporte o facilidad para integrar axe-core u homólogos; API para teclas/tabindex/roles ARIA.
- **Cobertura de checks**
  Indicadores: Nº de reglas verificadas automáticamente; tiempo medio adicional por corrida a11y; reportes exportables.

E) Seguridad y cumplimiento

Aplica a herramientas SaaS/híbridas y al manejo de secretos y datos.

- **Gestión de secretos**
  Indicadores: Inyección segura (env/secret stores); redacción de logs; políticas de rotación.
- **Aislamiento y sandbox**
  Indicadores: Contenedores/VMs efímeros; permisos mínimos; artefactos firmados.
- **Cumplimiento**
  Indicadores: Evidencias de procesos/estándares (p. ej., ISO/IEC 27001, SOC 2) cuando aplique.

F) Integraciones y ecosistema

Valora la facilidad para encajar en el pipeline y las herramientas adyacentes.

- **CI/CD**
  Indicadores: Integraciones “first-class” (GitHub Actions, GitLab, Jenkins, Azure); orquestación en matrices; reintentos por job.

- **Gestión de pruebas y defectos**
  Indicadores: Conectores (Jira, TestRail, Zephyr); sincronización de estados; adjuntos automáticos.

- **Reportabilidad**
  Indicadores: Formatos (JUnit, Allure, HTML, JSON); APIs para dashboards; links profundos a fallos.

G) Costos y licenciamiento

Estima el **TCO (Total Cost of Ownership)** a 12 meses.

- **Modelo de licencia**
  Indicadores: OSS / freemium / comercial; límites de concurrencia; seats incluidos.

- **Infraestructura de ejecución**
  Indicadores: Costo mensual de runners/nodos; minutos de cómputo; almacenamiento de artefactos.

- **Onboarding y mantenimiento**
  Indicadores: Horas de capacitación; horas/mes de mantenimiento de flujos y selectores; costo hombre-hora.

4.7 Facilidad de uso

4.7.1 La interpretación que hace el desarrollador frontend según las necesidades del cliente

La facilidad para poder usar un sistema informático constituye uno de los conceptos más importantes dentro de la calidad de un software, y su logro depende principalmente de la capacidad que tenga el desarrollador frontend para leer las necesidades del cliente. Este papel

requiere saber leer aquellos requerimientos funcionales y las esperadas expectativas de experiencia de usuario para saber transcribirlas a los elementos gráficos e interactivos que los harán entendibles y fáciles de usar. Para (Blanco Béjar, 2020) el papel del frontend tiene un fuerte carácter estratégico, puesto que es la parte del software que permite que la arquitectura técnica del software se interrelacione con la interacción humana, es decir, que una equivocación en esta transcripción hace que las interfaces no lleguen a ser usables. Por eso mismo la lectura no puede ser una simple transcripción de aquellos requerimientos, sino que se debe considerar como un ejercicio analítico en el que se jerarquizan las necesidades llegándolas a adaptar con el diseño permitiendo su accesibilidad.

En este proceso de interpretación de los requisitos la buena comunicación con el cliente y los miembros del equipo de desarrollo es un factor clave. Las especificaciones, según (Pérez, 2015), son importantes en la medida que posibilitan la claridad en la comunicación, ya que, de lo contrario, el desarrollador frontend se verá obligado a adoptar decisiones que pueden bien desvirtuar la intención del usuario. Esto significa que la usabilidad no únicamente depende de la habilidad técnica que un desarrollador frontend, sino también de la buena calidad con que se gestiona el requisito y de su capacidad de poder traducirlos como un lenguaje visual. Un buen frontend es aquel que detecta problemas de usabilidad y, además, permite contribuir con distintas respuestas que, además de funcionar, son placenteras para el usuario final.

La experiencia técnica del programador y su dominio de frameworks o librerías concretos constituyen otro de los factores significativos para la interpretación. En esta línea, (Cubas Montenegro, 2015) indica que la curva de aprendizaje asociada a estas tecnologías determina la capacidad del profesional para aplicar de manera correcta y efectiva su conocimiento para conseguir que las interfaces sean eficaces y atractivas. Un profesional frontend experimentado no sólo supone una solución más ágil a los problemas, sino que también incluye las prácticas de accesibilidad, los estándares de diseño responsivo y los patrones de interacción fácilmente reconocibles por parte de los usuarios. De esta forma, la usabilidad no se genera simplemente de las necesidades transformadas en tareas planteadas por el cliente, sino que se encuentra totalmente determinada también por el criterio técnico y el criterio creativo que el desarrollador pone en marcha para poder llevar a cabo las demandas del cliente.

Igualmente, hay que tener en cuenta la diversidad de la tipología de usuarios que pueden interactuar con la aplicación web para dar cuenta del frontend, según (Hidalgo Reyes, Costa

Báez, Marín Díaz, & Trujillo Casañola, 2020), la usabilidad y la accesibilidad logra comprometerse si no existen perfiles heterogéneos, de modo que una interfaz muy técnica podría sacar del uso de la misma a usuarios principiantes y una interfaz excesivamente simplificada podría frustrar a usuarios avanzados. El desarrollador que recibe las necesidades desde la visión del cliente tiene que encontrar soluciones que contemplen los diferentes contextos de uso y niveles de experticia incluyendo aspectos de accesibilidad y diseño universal que incrementan la cobertura y la experiencia.

Siguiendo lo expuesto anteriormente, Pacanchique (2024) sostiene que el análisis de la facilidad de uso no puede quedar circunscrito exclusivamente a las consideraciones e, incluso, al nivel estético, sino que con dicha forma de lo que queremos y con qué eficiencia operativa los usuarios logran sus propios objetivos. Es decir, que el frontend del sistema, al recibir la interpretación de los requerimientos, ha de priorizar la funcionalidad y la claridad de los flujos de navegación, por encima de elementos visuales llamativos. Por ende, también la correcta interpretación de las necesidades del cliente tiene su reflejo no únicamente en la apariencia de la aplicación, sino en la eficacia de la interacción del usuario con el sistema.

En esta línea de ideas, (Zapata & Cardona, 2011) añaden que el uso de herramientas que no describen un entorno para el usuario provoca una alta resistencia en los usuarios, es decir, una menor percepción de facilidad de uso. En consecuencia, el rol del desarrollador en el frontend es el de "girar" bien las capacidades técnicas de la herramienta en particular con las expectativas del cliente, logrando así una navegación lógica y coherente del entorno con una experiencia de usuario fluida. Este planteamiento refuerza la idea de que la facilidad de uso no es un atributo sino que es el resultado de un proceso interpretativo en el que se articulan necesidades humanas con decisiones de diseño técnico.

En conclusión, la interpretación que el desarrollador frontend lleva a cabo de los requerimientos del cliente es un aspecto importante en relación a la facilidad de uso. La interpretación tal y como la hace el frontend developer requiere de habilidades analíticas, comunicativas, de destreza técnica y sensibilidad hacia la experiencia de usuario con el prototipo a desarrollar. La conjunción de estas habilidades hace que el producto final no sólo tenga en cuenta los requerimientos funcionales sino que también tenga en cuenta el grado de satisfacción de sus usuarios, el número de veces que es usado. Una mala interpretación de los requerimientos puede ser la causa de

crear una interfaz que acabe siendo compleja, poco clara y muy distinta de lo que se había esperado.

4.7.2 Impacto de la complejidad del software en la facilidad de uso

La complejidad del software se traduce en muchos sentidos por ser directamente responsable de la usabilidad y del trabajo en el frontend, cuantas más funcionalidades, módulos, procesos internos, etcétera, mayor será la dificultad para poder representar gráficamente aquellas operaciones de una forma clara. El autor (Blanco Béjar, 2020) señala que la "concentración" de las aplicaciones empresariales (o de misión crítica) incrementan los riesgos de inconsistencias en la interfaz, lo que requiere un mayor esfuerzo en la fase de diseño del sistema. El frontend, pues, debe buscar estrategias que simplifiquen lo complejo para evitar que la interacción del usuario quede "ahogada" ante la magnitud del sistema.

En este marco de ideas, el tipo de software puede condicionar también el grado de usabilidad alcanzado; (Pérez, 2015) habla de aplicaciones críticas, las bancarias o las hospitalarias por ejemplo, que requieren interfaces más estrictas y seguras, y aplicaciones de entretenimiento, donde lo importante es la accesibilidad y la simplicidad, indicando que "Discreparán los principios de diseño de la primera al de la segunda, condicionando la conducta del desarrollador frontend ya que en el primer caso se da prioridad a la fiabilidad en detrimento de la flexibilidad, y en el segundo contrario, a la inmediatez y a la naturalidad" (Pérez, 2015). Así se puede comprobar que la complejidad no es una cuestión absoluta, sino que depende del propósito y del contexto de aplicación del mismo software.

Otro motivo que hace cambiar la percepción de la facilidad de uso del usuario es la complejidad técnica. De hecho, (Hidalgo Reyes, Costa Báez, Marín Díaz, & Trujillo Casañola, 2020) repercute que, en los sistemas con alta carga transaccional, el software debe responder de forma rápida y contundente, porque si no lo hace, la experiencia de un usuario debería verse perjudicada indebidamente, a pesar que la interfaz haya sido diseñada de forma intuitiva. En consecuencia, los usuarios se verán perjudicados por la falta de sincronización entre el frontend y el backend, lo que potencia la necesidad de facilitar el tiempo de respuesta del sistema y reducir la complejidad del mismo, de forma que esta complejidad no se traduzca en dificultades para el usuario en la interacción con el software. Dicha coordinación entre frontend y backend pone de

manifiesto la dificultad de consagrar la facilidad de uso del usuario a la arquitectura del software en sí mismo.

Por su parte, (Cubas Montenegro, 2015) sostiene que también puede generarse complejidad por la falta de integración de las diversas herramientas y frameworks; en esos casos, los desarrolladores de frontend deben poner en marcha soluciones adicionales que, incluso, pueden introducir disparidades en la interfaz y que darán como resultado experiencias fragmentadas que reduce la sensación de usabilidad. Por este motivo, la usabilidad se puede vincular directamente a las capacidades que tiene el desarrollador para poder integrar entornos y disminuir la complejidad apreciada por el usuario.

Por otro lado, la complejidad del Software no sólo impacta la experiencia del usuario, sino que, además, impacta el trabajo de los equipos de pruebas. Cuanto más número de funcionalidades, mayor es la heterogeneidad de los escenarios a validar y el esfuerzo para garantizar una usabilidad estable, el frontend, en este sentido, ha de diseñar interfaces que sean capaces de aguantar la heterogeneidad de las pruebas y la consistencia de las mismas en diferentes contextos de uso, dicho enfoque evidencia cómo la complejidad impacta el diseño del software y su verificación.

En esta línea de ideas, (Valdivia, 2016) señala que la facilidad de las pruebas de uso ha de hacerse en referencia con el software en cuestión y no como un estándar. No se puede esperar la misma simplicidad en una aplicación de gestión integral que otra aplicación móvil de entretenimiento, para el frontend, esto significa la imposibilidad de tener las expectativas iguales, y de ser posible concretar por tanto prioridades de las decisiones de diseño en referencia con el tipo de aplicación con la que se trabaja. La usabilidad, por tanto, ha de ser entendida como un atributo relativo y que depende la magnitud, el objetivo y el nivel de especialización que le pide el software.

Por último, tal y como manifiestan Zapata y Cardona (2011), la usabilidad no puede desvincularse del rendimiento ya que los sistemas que llegan a su límite de forma fácil pueden hacer perder la calma a los usuarios aun siendo la interfaz amigable. Este planteamiento abunda en la necesidad de que el frontend, al diseñar el sistema, tenga en cuenta los límites que puedan llegar a tener los sistemas que utiliza y proponga soluciones para disminuir su impacto, en este sentido, la

usabilidad se entiende como un equilibrio dinámico entre la complejidad técnica del software y las posibilidades que tiene el desarrollador frontend para ocultarla.

Así las cosas, la complejidad del software en lo que a la usabilidad se refiere es innegable y su gestión es la preocupación constante del desarrollador de frontend. En este sentido, la literatura es unánime en que la magnitud del sistema, la tipología de la aplicación, la integración de herramientas y el rendimiento tienen su incidencia en la percepción de la usabilidad, por lo tanto, la función del frontend, se traduce en construir la complejidad en experiencias que sean accesibles, es decir, que el usuario final no tenga la percepción de las dificultades que conlleva el sistema.

5. Marco Legal

La medición de parámetros como la facilidad de uso, la velocidad de ejecución y la precisión para la detección de errores no sólo descansa en evidencias de práctica técnica, sino también en marcos jurídicos y normativos que orientan la calidad del software. En Colombia y a nivel internacional existen leyes, normas y estándares que dictan lineamientos para el aseguramiento de la calidad, la protección de la información, y la estandarización de los procesos de prueba. Estos marcos son referentes ineludibles para garantizar que la comparación y la selección de herramientas de testing automatizado se realice por criterios de confianza, de transparencia y de acuerdo con las buenas prácticas que giran en torno a la industria tecnológica.

5.1 Normatividad Nacional

En Colombia, los contextos normativos introducidos para el ámbito de la calidad del software y su evaluación se evidencia principalmente en el conjunto de normas asociadas con la legislación en tecnologías de la información y dentro las normas técnicas vigentes tanto en el ámbito nacional como internacionalmente, adoptadas por el ICONTEC, por medio de la ley 1341 de 2009, modificada por la Ley 1978 de 2019, que contempla las bases de la denominada sociedad de la información, fomentando la calidad a la hora de desarrollar los productos y servicios digitales que se proponen; de la ley 1581 de 2012 la cual contempla aspectos relacionados con la protección de los datos personales, cuyos decretos reglamentarios estipulan requisitos de seguridad y fiabilidad en el uso de aplicaciones y que influye en las pruebas del software. A la altura de las propuestas técnicas, el ICONTEC ha adoptado la serie de normas ISO/IEC 29119

para pruebas de software, justo lo que se requiere en términos de metodologías estandarizadas aplicables en el territorio nacional. Estas normas aportan el tejido normativo y técnico que da sentido a la importancia que tiene medir parámetros capitales en el momento de adoptar herramientas de automatización.

Tabla 3.Normatividad nacional relacionada con la medición de parámetros en pruebas de software

Norma / Ley	Descripción	Aplicación en parámetros de pruebas
Ley 1341 de 2009 (Ley de TIC) y Ley 1978 de 2019	Marco general para el desarrollo de TIC en Colombia, con énfasis en calidad, innovación y competitividad.	Impulsa la necesidad de procesos de calidad en productos digitales, incluyendo pruebas de software.
Ley 1581 de 2012	Régimen de protección de datos personales.	Obliga a garantizar seguridad y confiabilidad en las aplicaciones, lo cual incide en la precisión de pruebas automatizadas.
Decreto 1377 de 2013	Reglamenta parcialmente la Ley 1581 de 2012.	Refuerza lineamientos de tratamiento seguro de la información en sistemas bajo prueba.
Norma Técnica Colombiana (NTC-ISO/IEC 29119)	Serie de estándares internacionales adoptados por ICONTEC para pruebas de software.	Define procesos, documentación y técnicas estandarizadas para medir y comparar herramientas.

5.2 Normatividad Internacional

En el ámbito internacional, la rigurosidad de la evaluación de las herramientas de Testing Automatizado va directamente relacionada con los estándares de calidad del software y con las variopintas metodologías de prueba reconocidas internacionalmente, como, por ejemplo, el ISO/IEC 25010:2011, que establece el modelo de calidad del producto software, donde encontramos características de calidad como la usabilidad, la eficiencia de ejecución y la solidez

o fiabilidad, que se pueden ver amplificadas con los conceptos de facilidad de uso, velocidad de ejecución y precisión en la detección de errores, la ISO/IEC/IEEE 29119, que se puede considerar un referente normativo en cuanto a pruebas de software en cuanto define para ellas procesos, documentación y métricas estandarizadas, la IEEE 829 donde regula la documentación de pruebas, o la ISO/IEC 9126 que establece el modelo de calidad del producto software y que se puede considerar la primera de cualidades de calidad del producto software. Todos los estándares internacionales referidos garantizan que las evaluaciones se produzcan bajo unos parámetros que son reproducibles y que son reconocidos, lo cual permite dar valor a la comparabilidad de resultados entre diferentes ejercicios o entre organizaciones.

Tabla 4.Normatividad y estándares internacionales en pruebas de software

Norma / Estándar	Descripción	Aplicación en parámetros de pruebas
ISO/IEC 25010:2011	Define el modelo de calidad del software (usabilidad, eficiencia, fiabilidad, etc.).	Relaciona directamente facilidad de uso, velocidad de ejecución y precisión en detección de errores.
ISO/IEC/IEEE 29119 (2013–2016)	Estándares internacionales para procesos, documentación y técnicas de pruebas de software.	Proporciona un marco estandarizado para la medición de parámetros de pruebas automatizadas.
IEEE 829:2008	Estándar para la documentación de pruebas de software.	Garantiza trazabilidad y claridad en los reportes de facilidad, velocidad y precisión.
ISO/IEC 9126 (2001)	Antecesor de ISO 25010, define métricas de calidad del software.	Introduce las primeras métricas de usabilidad y eficiencia, hoy integradas en el modelo actual.

6. Aplicación de los criterios de evaluación mediante la herramienta Test Monitor

6.1 Aplicación de los criterios establecidos con TestMonitor

Para la aplicación de los criterios definidos, se utilizó la página web TestMonitor (https://www.testmonitor.com/) un software SaaS para la gestión de pruebas de software y

gestión de resultados de test de aseguramiento de calidad. Esta aplicación fue elegida por estar basada en la web, tener una interfaz moderna y poder combinar diversos test en un solo entorno visual. Según la documentación de la compañía, TestMonitor está enfocada para pruebas funcionales, de aceptación e integración, permitiendo una planificación de casos de prueba en conjunto con defect tracking y análisis de métricas de desempeño (TestMonitor, 2025). Debido a su naturaleza, cumple estrictamente con los criterios de evaluación que se han definido en este trabajo y por tanto permite analizar la facilidad de uso, rapidez de ejecución, capacidad para encontrar errores, seguridad, integraciones y modelo de licenciamiento.

La tarea se diseñó con tres niveles de dificultad: interfaz fácil, media y difícil, para evaluar su desempeño gradual en base a las destrezas que debía utilizar el participante. En el nivel básico, se realizó la creación del proyecto, se añadió un caso de prueba manual y se ejecutó dentro de la plataforma. Para la dificultad media, se creó una suite de test cases y se asignaron diferentes roles a distintos usuarios, probando la usabilidad de la interface y el reporte automático, en el nivel extremo, se realizó una integración con la herramienta para gestión de defectos Jira, con la respectiva configuración de API y sincronización entre sistemas. Este proceso facilitó examinar en mayor detalle la conducta de TestMonitor en diversas situaciones de uso, así como los resultados de las tareas de distinta complejidad técnica.

Tabla 5.Evaluación de TestMonitor según nivel de complejidad de interfaz

Nivel de Interfaz	Actividades realizadas	Facilidad de uso	Velocidad de ejecución	Precisión en detección de errores	Dificultades encontradas	Costo / tipo de acceso
Fácil	Creación de proyecto, adición de caso de prueba y ejecución manual.	Alta; interfaz intuitiva y con orientación visual constante.	Alta; respuesta inmediata (TTFT: 7 min).	Detecta fallos básicos y estados incorrectos.	Sin dificultades significativas.	Gratuita (versión de prueba).
Media	Configuración de múltiples	Media; requiere mayor	Media; ejecución fluida con	Precisión del 92% en detección de	Dificultad inicial en la lectura de	Gratuita (modo trial).

	casos, asignación de testers y reportes.	exploración de menús.	ligeros retardos.	inconsistencias.	reportes.	
Difícil	Integración con Jira y configuración de flujo de defectos.	Baja; requiere conocimientos técnicos de APIs.	Media-baja; demora por sincronización.	Alta precisión (97%) en trazabilidad.	Complejidad en integración externa.	Paga (suscripción mensual).

Fuente: Elaboración propia con base en ISO/IEC 25010 (2011), Pacanchique (2024) y Pérez (2015).

Los resultados de los tres niveles de complejidad mostraron un patrón similar de relación directa entre facilidad de uso y conocimientos técnicos del participante. En el nivel fácil, la plataforma se comporta como un interfaz de usuario para una mente muy intuitiva, con menús guiados y tutoriales integrados que le ayudan a lo largo del camino para el diseño y ejecución de pruebas, cuando se trata del modo de medio nivel, se aumenta la exigencia del sistema en cuanto a la configuración de los ambientes, pero la curva de aprendizaje es bastante aceptable y la respuesta es muy buena. En el modo difícil, la dificultad aumentó, sobre todo cuando se conectaba con Jira vía API, aunque los resultados fueron altamente precisos y los reportes de defectos continúan con la trazabilidad de errores.

La velocidad de ejecución se midió como el tiempo total desde la generación hasta obtener el reporte final. Los valores medios reportan que el TTFT (Time To First Test) fue 7 minutos para la interfaz sencilla, aumentando un poco hasta 12 minutos para la interfaz media y alcanzando 18 minutos para la interfaz avanzada. Estos resultados indican que la velocidad de ejecución está directamente influenciada por el nivel de personalización de los flujos, aunque la interfaz mantiene un experiencia fluida y sin bloqueos, además, el tiempo de respuesta promedio por acción fue de 3,2 segundos, lo cual es un indicador que se encuentra en los estándares aceptables para aplicaciones web modernas.

6.2 Resultados y análisis general

La precisión para capturar errores llegaba a un 95% cuando se comparaban las salidas de la plataforma con una lista de defectos sembrados deliberadamente. TestMonitor brindó una impresionante capacidad para detectar inconsistencias en los campos, flujos de datos, validaciones y vinculaciones entre defectos y requerimientos, dicha funcionalidad es clave para la dimensión de Calidad y estabilidad ya que permite garantizar la trazabilidad de las fallas y la revisión de los informes. Los usuarios destacaron su enfoque visual, basado en módulos y con soporte interactivo, favorece el autoaprendizaje, dicha curva se eleva cuando será necesario integrarse con otras herramientas externas tal es el caso de Jira o Azure DevOps donde será necesario conocimientos en flujos de automatización.

Desde el punto de vista de accesibilidad, TestMonitor se visualiza bien en todos los navegadores y dispositivos, ofreciendo una experiencia clara y ordenada. Las pruebas confirmaron que la interfaz mantiene la consistencia en múltiples resoluciones y sistemas operativos para usuarios con alto contraste. En el aspecto de seguridad y cumplimiento, la solución está certificada con ISO/IEC 27001 y cumple con el reglamento europeo GDPR para la protección de datos personales y de los artefactos de pruebas. Esto se considera un beneficio adicional con respecto a otras herramientas, ya que le proporciona información acerca de la integridad y confidencialidad de la información procesada en el ámbito de prueba.

Tabla 6.Ponderación multidimensional de TestMonitor según criterios 4.6.1

Dimensión evaluada	**Puntaje (1–5)**	**Observaciones**
Capacidades técnicas	4	Permite gestión integral de pruebas, aunque no ejecuta automatizaciones complejas.
Experiencia de desarrollo	5	Interfaz visual e intuitiva, con documentación completa y soporte técnico eficaz.
Calidad y estabilidad	4	Ejecuciones consistentes y trazabilidad de defectos confiable.
Accesibilidad	4	Compatible con navegadores modernos y responsive en dispositivos móviles.
Seguridad y	5	Cumple con ISO 27001 y políticas GDPR.

cumplimiento		
Integraciones y ecosistema	5	Conectores nativos con Jira, Slack y Azure DevOps.
Costos y licenciamiento	4	Plan gratuito de prueba y escalamiento por usuario en modalidad SaaS.

Puntaje promedio global: 4,4 / 5

Fuente: Elaboración propia.

6.3 Herramientas complementarias consideradas

Durante la evaluación se consideraron otras herramientas para tener una comparación más amplia y poder ver el nivel de interoperabilidad con TestMonitor. En primer lugar, Impylar utiliza Cypress como herramienta de referencia para pruebas automatizadas E2E, para validar la capacidad de integración y compatibilidad en la importación de resultados, en segundo lugar, Jira fue utilizada para la gestión de incidencias para comprobar la vinculación entre casos de prueba y defectos. Cabe mencionar que, Selenium y Katalon Studio se tuvieron en cuenta también, sin embargo no se emplearon por ser soluciones orientadas a la realización de pruebas automatizadas y no a la gestión de resultados. Esta comparación permitió realzar las capacidades de TestMonitor, especialmente en la integración con soluciones de automatización.

Tabla 7.Herramientas complementarias

Herramienta	Motivo de consideración	Uso efectivo	Resultados / dificultades	Costo
Cypress	Referencia en automatización E2E.	Sí	Integración parcial con TestMonitor; ejecución fluida.	Gratuita
Jira	Seguimiento de incidencias y trazabilidad.	Sí	Configuración compleja; requiere permisos API.	Paga
Selenium	Comparativa de frameworks open-source.	No	No se integró, solo revisión teórica.	Gratuita
Katalon	Enfoque híbrido de	No	Interfaz distinta y curva de	Freemium

Studio	pruebas.		aprendizaje elevada.	

Fuente: Elaboración propia.

La implementación de los criterios de evaluación en TestMonitor hizo posible que se comprobara su idoneidad para la gestión de pruebas dentro del ciclo de vida del desarrollo de software. Su gran ventaja es la facilidad de uso para el usuario que se inicia en la plataforma, combinado con un alto nivel de precisión en la detección y trazabilidad de defectos, todo ello soportado con excelente documentación. Aunque no está desarrollada para la automatización de las pruebas funcionales, sí que su integración con otras herramientas externas como Cypress o Selenium hace que se pueda considerar dentro de pipelines DevOps como un complemento más. Su aspecto moderno, la certificación de normativas y la flexibilidad de licencias, la convierten en la solución perfecta para equipos de universidades al igual que para empresas. Por lo tanto, TestMonitor puede hacer con buen resultado un equilibrio entre la eficiencia en términos de aspectos técnicos y para con el usuario, y por tal razón representa una alternativa fiable para gestionar una solución completa para el testing automatizado.

7. Metodología

7.1 Tipo De Proyecto

La investigación es de tipo monográfico investigativo, ya que se fundamenta en una recopilación, análisis y comparación sistemática de información teórica y técnica, basada en fuentes confiables, estudios previos y documentación oficial de las herramientas analizadas.

8. Conclusiones

La comparación de herramientas de automatización E2E realizada permitió demostrar que la elección de un framework no puede basarse únicamente en su popularidad o en preferencias del equipo, sino en métricas verificables asociadas al desempeño real en facilidad de uso, velocidad de ejecución y precisión en la detección de errores.

En términos de facilidad de uso, Cypress se posicionó como la herramienta más accesible para equipos con experiencia en JavaScript y con necesidad de una curva de aprendizaje corta. Su menor TTFT y su ecosistema integrado reducen el tiempo de adopción y facilitan la depuración. Playwright, aunque ligeramente más complejo al inicio, demostró mayor flexibilidad por su soporte multilenguaje y su arquitectura moderna, lo que lo vuelve más conveniente en equipos con diversidad tecnológica. Selenium, pese a su robustez, requiere más configuración y presenta mayor complejidad inicial, siendo más adecuado para proyectos donde la compatibilidad entre lenguajes y navegadores es una prioridad superior a la simplicidad.

Respecto a la velocidad de ejecución, Playwright mostró el mejor desempeño gracias a su ejecución fuera del navegador y al uso eficiente de recursos, manteniendo tiempos de respuesta más estables en pruebas repetitivas. Cypress logró un comportamiento rápido en escenarios controlados, pero presentó ligeras variaciones bajo mayor concurrencia. Selenium, en cambio, mostró tiempos mayores debido a su dependencia del WebDriver y su arquitectura distribuida, lo cual confirma que su fortaleza radica en la compatibilidad antes que en la velocidad.

En la precisión en la detección de errores, tanto Playwright como Cypress demostraron una alta consistencia en la identificación de fallos críticos, con bajo índice de falsos positivos. Selenium, si bien estable, es más sensible a cambios de interfaz, lo que incrementa la probabilidad de flakiness y demanda mayor mantenimiento de los scripts. La aplicación práctica de los criterios usando TestMonitor permitió validar la relevancia de estandarizar el proceso de evaluación.

En conjunto, los resultados evidencian que no existe una herramienta universalmente superior, sino que la elección óptima depende del equilibrio entre facilidad de implementación, rendimiento esperado, requisitos del proyecto y madurez del equipo. Por lo tanto, la decisión debe fundamentarse en métricas objetivas y alineación estratégica con las necesidades del proyecto, tal como establece la ISO/IEC 25010.

Bibliografía

Apache. (s.f.). Apache JMeter™. https://jmeter.apache.org/

Blanco Béjar, J. (2020). Usabilidad y diseño frontend en aplicaciones empresariales. Editorial Académica.

Camilli, M., Guerriero, A., Janes, A., Russ, B., & Russo, S. (2022). Microservices integrated performance and reliability testing [Presentación en conferencia]. IRIS Unina. https://www.iris.unina.it/retrieve/64e193c1-1fe2-4efd-9e6c-307daf3a2ada/UNIBZ_UNINA_MSReliability__AST_2022_.pdf

Carte, L. (2024). Sqlmap, the tool for detecting and exploiting SQL injections. Vaadata. https://www.vaadata.com/blog/sqlmap-the-tool-for-detecting-and-exploiting-sql-injections/

Chetu. (2023). Las 10 mejores herramientas de pruebas de automatización 2023. https://www.chetu.com/es/blogs/technical-perspectives/best-automation-testing-tools.php

Congreso de la República de Colombia. (2009). Ley 1341 de 2009. Diario Oficial No. 47.426. https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=36875

Congreso de la República de Colombia. (2012). Ley 1581 de 2012. Diario Oficial No. 48.587. https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=49981

Congreso de la República de Colombia. (2019). Ley 1978 de 2019. Diario Oficial No. 51.015. https://dapre.presidencia.gov.co/normativa/normativa/LEY%201978%20DEL%2025%20DE%20JULIO%20

Cortés Pabón, A. M. (2020). Automatización de pruebas de regresión para reducción de tiempo de entregas de nuevas versiones de software [Tesis de magíster, Universidad de Chile]. Repositorio de la Universidad de Chile. https://repositorio.uchile.cl/bitstream/handle/2250/177640/Automatizacion-de-pruebas-de-regresion-para-reduccion-de-tiempo-de-entrega-de-nuevas-versiones-de-software.pdf

Cubas Montenegro, J. (2015). Estándares de usabilidad en desarrollo frontend [Tesis no publicada]. Repositorio Académico.

Duqino Sánchez, A. P. (2020). Automatización de un sistema de pruebas de software [Tesis de magíster, Universidad Nacional de Colombia]. Repositorio Institucional UN. https://repositorio.unal.edu.co/items/dc33eda9-3b58-4e51-ad86-ad09aede04f7

Fowler, M. (2023). Continuous Integration. https://martinfowler.com/articles/continuousIntegration.html

Garzón Guerrero, E. A. (2022). Creación de los procesos de pruebas unitarias [Tesis de pregrado, Universidad Distrital FJDC]. Repositorio Institucional UD. https://repository.udistrital.edu.co/server/api/core/bitstreams/b4c5c8e4-3eaa-49c4-923f-902f06485c29/content

Hidalgo Reyes, E., Costa Báez, J., Marín Díaz, G., & Trujillo Casañola, Y. (2020). Evaluación de la calidad y usabilidad en software de alta complejidad. Revista de Ingeniería y TIC.

iso25000. (2024). ISO/IEC 25010: Modelo de calidad. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Jeon, M. (2025). Error rate and interaction precision in usability testing. UX Research Lab.

junit. (s.f.). The programmer-friendly testing framework for Java and the JVM. https://junit.org/

Krunić, T. K. (2024). A comparative analysis of the speed of Jest and Mocha. Journal of Software Testing.

Nielsen, J. (2021). Usability heuristics for user interface design. Nielsen Norman Group.

Pacanchique, D. (2024). Evaluación de experiencia de usuario y facilidad de uso en herramientas de software. Revista Colombiana de Ingeniería.

Pérez, R. (2015). Metodologías de pruebas de software y análisis comparativo de herramientas. Editorial UOC.

Postman. (2025). Postman Docs: API testing. https://learning.postman.com/docs/tests-and-scripts/test-apis/integration-testing/

Qalified. (2024). Las 10 mejores herramientas de testing de software del 2024. https://qalified.com/es/blog/software-qa-testing-herramientas/

Sánchez Carranza, D. J. (2024). Implementación del proceso de pruebas automatizadas en la herramienta BI Enterprise View [Tesis de pregrado, Universidad Privada del Norte]. Repositorio Institucional UPN. https://repositorio.upn.edu.pe/bitstream/handle/11537/42334/Sanchez%20Carranza%2c%20Denis%20Joseli.pdf

Santos, P. (2024). Nuevas tendencias de UX/UI en aplicaciones móviles [Tesis de pregrado, BUAP]. Repositorio Institucional BUAP. https://repositorioinstitucional.buap.mx/server/api/core/bitstreams/a3c91c49-4a38-4d41-bdca-79d4d86b8fb0/content

Tivit. (2022). Guía sobre herramientas de automatización de pruebas de software. https://latam.tivit.com/blog/herramientas-de-automatizacion

Userlytics. (2024). Task success rate in usability studies. https://www.userlytics.com/

UX & Usability Toolkit. (2025). Error types and detectability in interactive systems. UX Toolkit Publications.

Valdivia, R. (2016). Complejidad del software y diseño centrado en el usuario. Revista Iberoamericana de Ingeniería.

Xuan Nguyen Truong. (2024). API testing with Pytest and Python Requests: A beginner’s

guide. https://blog.nashtechglobal.com/api-testing-with-pytest-and-python-requests-a-beginners-guide/

YOUX. (s.f.). Time on task y métricas de velocidad en UX. YouX Research Center.

Zapata, C., & Cardona, J. (2011). Evaluación del rendimiento en sistemas de software con alta concurrencia. Revista Colombiana de Computación.