9. Plataformas para Pruebas

9.1. Introducción y objetivos

En este tema se estudian las principales plataformas que nos ayudan a automatizar las pruebas del software.

Existen diversas formas de poder clasificar estas herramientas al atender las etapas del ciclo de vida, el alcance o los lenguajes y tecnologías de programación a los que se orienta. En este tema, clasificaremos estas soluciones en las que se centran en las pruebas unitaria, las de integración, las de sistema, las de validación y, finalmente, presentaremos otras herramientas que pueden apoyar en varias de las anteriores.

Con el estudio de este tema pretendemos alcanzar los siguientes objetivos:

* Comprender el valor y la utilidad que tienen las plataformas para la automatización de las pruebas del software.
* Identificar los criterios principales que nos ayuden a seleccionar las plataformas en cada caso.
* Conocer las principales plataformas de pruebas unitarias.
* Conocer las principales plataformas de pruebas de integración.
* Conocer las principales plataformas de pruebas de sistema.
* Conocer las principales plataformas de pruebas de aceptación.
* Conocer algunas de las plataformas y herramientas de pruebas mejor valoradas.
* Saber utilizar algunas de las principales plataformas de pruebas.

9.2. Contexto de las plataformas para pruebas

El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) define las pruebas de software como el proceso para evaluar un sistema o sus componentes de forma manual o automática para determinar si cumple con los requisitos del usuario o para encontrar las diferencias entre los resultados reales y los resultados esperados (Zein, Salleh y Grund, 2016).

El desarrollo de pruebas de software tiene persigue estos objetivos (Naik y Tripathy, 2011):

* Asegurar que el software funciona: al implementar una unidad de programa, el programador puede asegurar si el código funciona o no en circunstancias normales. La misma idea se aplica también a un sistema completo: una vez que un sistema se ha integrado, los desarrolladores pueden querer probar si el sistema realiza las funciones de forma correcta en su conjunto.
* Buscar posibles fallos: una vez que el equipo de desarrollo asegura que una unidad o el sistema completo funciona hasta cierto punto, con las pruebas es posible reproducir escenarios que lleven al sistema a situaciones no previstas con el objetivo de encontrar posibles fallos.

A menudo se habla del testeo como la validación y verificación, si bien son conceptos que se aplican a un ámbito distinto (International Organization for Standardization, 2022):

* Verificación: este tipo de actividad nos ayuda a evaluar un sistema de software determinando si el producto de una fase de desarrollo dada satisface los requisitos establecidos antes del inicio de dicha fase. Estas actividades se pueden acotar a un producto intermedio, como la especificación de requisitos, especificación de diseño, código, manual de usuario o incluso el producto final.
* Validación: actividad orientada a confirmar que un producto cumple con su uso previsto, es decir, que un producto cumple con las expectativas de sus clientes. En otras palabras, las actividades de validación se centran en el producto final, que se prueba exhaustivamente desde el punto de vista del cliente. La validación establece si el producto cumple con las expectativas generales de los usuarios.

En el contexto de las pruebas del software, el concepto de verificación hace referencia a que el software funciona sin fallos, mientras que la validación se centra en comprobar que el software cumple las necesidades recogidas en los requisitos.

Las pruebas de *software* pueden realizarse de forma manual o automatizada (Valliammai y Sujatha, 2015):

* Las pruebas manuales se realizan interactuando con el software en función de algunos casos de prueba predefinidos. Los casos de prueba proporcionan explicaciones de las características examinadas y los resultados esperados. El proceso de prueba manual es costoso en tiempo y su mayor problema es que requiere una completa reproducción, con el mismo esfuerzo asociado, cada vez que existen cambios en el software para corregir errores o dentro del mantenimiento evolutivo.
* Las pruebas automatizadas hacen que este proceso sea más eficaz, ya que podemos definir casos de prueba solo una vez y ejecutarlos de forma automática muchas veces. Esta característica es fundamental para poder realizar pruebas de regresión que garanticen que el software sigue funcionando como se esperaba cuando realizamos cambios derivados de mantenimientos correctivos o nuevas funcionalidades.

En lo relativo a las posibles plataformas o herramientas de pruebas, existen muchas formas de poder clasificarlas en función de diversos criterios como la fase o ámbito del testeo, las tecnologías, el alcance técnico, el entorno en el que opera, los lenguajes de programación que son de aplicación, el cumplimiento o no de marcos normativos o las licencias de uso entre otros (Abdulwareth y Al-Shargabi, 2021).

La consultora Gartner, en su *Magic Quadrant for Software Test Automation* (Herschmann, Murphy, Scheibmeir, 2019), posiciona a los principales fabricantes de esta forma:

A white grid with blue dots

Description automatically generated

Figura 1. *Gartner Magic Quadrant for Software Test Automation*. Fuente: Herschmann, Murphy, Scheibmeir, 2019.

* Líderes: Tricentis, SmartBear, Eggplant y Micro Focus.
* *Challengers:* Microsoft, IBM y Worksoft.
* *Visionaries*: CA Technologies y Parasoft.
* *Niche players*: Ranorex y Froglogic.

En nuestro caso, vamos a hacer uso del modelo en V del testeo de software para establecer una posible clasificación de plataformas, de forma que las pruebas se realizan en diferentes niveles que involucran el sistema completo o partes de a lo largo del ciclo de vida de un producto de software. En el modelo de testeo en V, se plantean cuatro etapas de pruebas que son conocidas como pruebas unitarias, de integración, de sistema y de nivel de aceptación. Los tres primeros niveles de las pruebas son realizados por una serie de diferentes equipos participantes en el desarrollo organización y las pruebas de aceptación son habitualmente realizadas por los clientes. Por ello, hablaremos de plataformas de pruebas unitarias, de integración, de sistema y de aceptación. Si bien alguna de las herramientas podrá ofrecer funcionalidades en varias fases, tal y como refleja el modelo en la Figura 2:

A diagram of a diagram

Description automatically generated

Figura 2. Modelo de testeo en V simple. Fuente: elaboración propia.

9.3. Valor y utilidad de las plataformas para pruebas

El valor y la utilidad de estas herramientas, con carácter general, es reducir el esfuerzo y coste asociado a las pruebas, en los que se incluye:

* El coste de diseñar las pruebas: mediante librerías o entornos visuales, podremos definir de una forma más rápida qué queremos probar y generar el código o entorno apropiado para su futura ejecución.
* El coste de mantener las pruebas: en el caso de cambios en el código, podremos redefinir las pruebas y mantenerlas coherentes con cada versión del producto.
* El coste de ejecutar las pruebas: cuando tenemos muchos casos de prueba, las herramientas ayudan a automatizar su ejecución lo que permite establecer prioridades, grupos o incluso dependencias entre ellas.
* El coste de analizar el resultado de cada caso de prueba: una prueba debe tener un resultado esperado y las herramientas nos van a simplificar la forma en la que podemos determinar mediante reglas, expresiones regulares o anotaciones que determinen en qué condiciones daremos la prueba por válida y en cuáles es un posible error.
* El coste de documentar las pruebas: como buena práctica de ingeniería, es necesario documentar las diversas fases del proceso de desarrollo, por lo que las pruebas no son una excepción. Disponer de una documentación que nos permita evidenciar que el software ha sido probado y los resultados obtenidos son una garantía adicional hacia el equipo de desarrollo y el cliente de la calidad esperada del producto.

Para llevar a cabo una prueba, es preciso seguir una serie de pasos como los que se detallan a continuación:

* Identificar un objetivo a probar: el primer paso es definir qué unidad, módulo, sistema, funcionalidad o requisito queremos probar.
* Seleccionar las entradas: el segundo paso, dentro del diseño del caso de prueba, es definir los posibles valores de entrada que serán representativos para llevar a cabo la prueba, basándonos en los requisitos de usuario o técnicos en cada caso.
* Calcular el resultado esperado: el tercer paso es identificar qué valor o conjunto de valores harán que demos por correcta (*pass*) o por errónea (*fail*) una prueba.
* Configurar el entorno de ejecución del programa: como cuarto paso, debemos disponer de un entorno que permita la ejecución de las pruebas.
* Ejecutar la prueba: el quinto paso es la ejecución de la prueba con las entradas seleccionadas y obtener el resultado de la ejecución.
* Analizar el resultado de la prueba: el último paso es analizar el resultado obtenido con el esperado, para ver si es correcta o errónea.

El resultado de una prueba puede ser correcto (*pass*) o erróneo (*fail*). Que una prueba falle no significa que exista un error en el software, sino que el resultado obtenido en la prueba no satisface las reglas de validación establecidas. Es necesario revisar si en efecto se trata de un error o es un escenario no contemplado en la prueba.

A la hora de seleccionar las plataformas de pruebas, cada organización debe establecer sus propios criterios, basados en múltiples perspectivas(Raulamo-Jurvanen, Mäntylä, y Garousi, 2017), en los que los principales son:

* El coste de licenciamiento y soporte.
* La curva de aprendizaje necesaria, condicionada por la cantidad de documentación o formación existente.
* La generación de informes de prueba.
* La compatibilidad con entornos de integración y despliegue continuos.
* El nivel de conocimiento de programación requerido y las tecnologías o lenguajes soportados.

Las plataformas y herramientas de apoyo a la automatización de las pruebas nos pueden reducir el esfuerzo, pero no son suficientes, ya que requieren de un proceso de pruebas que defina qué hay que probar, cómo probar, cuándo probar, cómo medir la calidad y eficiencia de las pruebas y cómo gestionar las incidencias o errores detectados.

Puedes aprender más sobre esto en *ISO/IEC/IEEE 29119-2:2021 Software and systems engineering — Software testing — Part 2: Test processes* en el siguiente enlace <https://www.iso.org/standard/79428.html>

9.4. Plataformas para pruebas unitarias

La prueba unitaria es una parte del proceso de desarrollo de software en el que las partes comprobables más pequeñas de una aplicación, llamadas unidades, se analizan de forma individual e independiente para garantizar un funcionamiento adecuado. Esta actividad de pruebas la realizan habitualmente los desarrolladores de software y, a veces, el personal de control de calidad durante el proceso de desarrollo. El objetivo principal de las pruebas unitarias es aislar el código escrito para probar y determinar si funciona según lo previsto.

Las pruebas unitarias son un paso importante en el proceso de desarrollo porque, si se realizan correctamente, pueden ayudar a detectar fallas tempranas en el código que pueden ser más difíciles de encontrar en etapas de prueba posteriores. Al estar centradas en unidades de código, las plataformas de apoyo a la definición o ejecución de las pruebas son específicas para cada tecnología o lenguaje de programación que estemos utilizando.

Algunas de las herramientas más populares para facilitar la definición y ejecución de pruebas unitarias son:

A screenshot of a computer

Description automatically generated

Tabla 1. Plataformas para pruebas unitarias. Fuente: elaboración propia.

Chai

Es una biblioteca de aserciones para el Node.js y el navegador que se puede combinar maravillosamente con cualquier marco de prueba de JavaScript. Cuenta con varias interfaces que permiten al desarrollador elegir la más cómoda.

Embunit

Es un marco de prueba de unidad gratuito para software en sistemas embebidos para desarrolladores y para evaluadores de aplicaciones de software escritas en C o C++. Su diseño es algo similar a JUnit, define los casos de prueba en formato estructurado para generar código fuente. Reduce el retrabajo de las pruebas unitarias, ya que los casos de prueba relacionados se almacenan en el mismo conjunto de pruebas y el resultado final se genera en formato XML.

HtmlUnit

Es una biblioteca Java de código abierto que contiene un navegador sin interfaz de usuario para programas Java. Esta herramienta es compatible con JavaScript y proporciona funciones visuales como formularios, enlaces, tablas, etc. Admite protocolos como HTTP, HTTPS junto con *cookies*, métodos de envío como GET, POST y servidor *proxy*. Permite probar aplicaciones web que se utilizan dentro de plataformas como JUnit y TestNG, usando el motor de JavaScript Mozilla Rhino.

Jasmine

Es un marco de pruebas basadas en el comportamiento (*behavior-driven development*) para JavaScript. No depende de ningún otro marco de JavaScript ni de un DOM y tiene una sintaxis limpia que facilita la escritura de las pruebas. Admite pruebas de código de interfaz de usuario a través de una extensión *front-end* de Jasmine llamada Jasmine-jQuery(Hahn, 2013).

JUnit

Es un marco de pruebas unitarias de código abierto diseñado para el lenguaje de programación Java. Con base en anotaciones, ayuda a escribir código de prueba de manera fácil y rápida (Gulati y Sharma, 2017).

A screenshot of a computer

Description automatically generated

Tabla 2. Anotaciones de Junit. Fuente: elaboración propia.

En el vídeo *JUnit5, pruebas sobre la Java Virtual Machine*, aprenderemos a utilizar el entorno JUnit5 para automatizar las pruebas sobre la máquina virtual de Java.

Karma

Es un marco de pruebas de código abierto que proporciona un entorno de creación de pruebas productivo para JavaScript y que ejecuta pruebas en dispositivos reales. Facilita la depuración fácil y la integración eficiente con Jenkins, Travis o Semaphore.

Mocha

Es un marco de prueba de JavaScript rico en funciones que se ejecuta en Node.js y en el navegador, lo que hace que las pruebas asincrónicas sean simples. Las pruebas de Mocha se ejecutan en serie, lo que permite informes flexibles y precisos, al tiempo que asigna excepciones no detectadas a los casos de prueba correctos.

NUnit

Es un marco de pruebas unitarias para cualquiera de los lenguajes incluidos en la plataforma .NET. Su funcionamiento es similar a JUnit, incluyendo pruebas basadas en datos que se pueden ejecutar en paralelo y cuenta con un Console Runner para cargar y ejecutar pruebas (Thomas, Hunt. y Hargett, 2007).

PHPUnit

Es un marco de prueba orientado al programador para el lenguaje PHP. Es una instancia de la arquitectura xUnit para marcos de pruebas unitarias.

SimpleTest

Es un marco de pruebas unitarias de código abierto dedicado al lenguaje de programación PHP. Admite SSL, formularios, *proxies* y autenticación básica. Las clases de casos de prueba se generan ampliando desde las clases de prueba base junto con los métodos y códigos. Incluye una funcionalidad paratransformar casos de prueba en *scripts* de prueba ejecutables.

TestNG

Es un marco de prueba de automatización de código abierto para el lenguaje de programación Java. Esta herramienta está fuertemente influenciada por *JUnit* con pruebas simultáneas, soporte de anotaciones, admite pruebas parametrizadas y basadas en datos junto con un modelo de ejecución potente y una configuración de prueba flexible (Jones, 2018).

TypemockIsolator

Es un marco gratuito de código abierto para probar el código del sistema para C y C++ principalmente para Windows. Reduce el consumo de tiempo para la corrección de errores y la entrega de valor. Contiene API simples y métodos de entrada sin cambiar el código heredado.

xUnit.net

Es una herramienta gratuita para pruebas unitarias, de código abierto y centrada en la comunidad para .NET Framework. Escrito por el inventor original de NUnit v2, es la última tecnología para pruebas unitarias de C#, F#, VB.NET y otros lenguajes .NET, funciona con extensiones de los entornos de desarrollo integrado (IDE) como ReSharper, CodeRush, TestDriven.NET y Xamarin(Adewole, 2018).

9.5. Plataformas para pruebas de integración

Las pruebas de integración son un tipo de pruebas de software en las que las diferentes unidades, módulos o componentes de una aplicación de software se prueban como un sistema combinado. Cada una de sus partes o módulos han podido ser codificados por diferentes programadores. El objetivo de las pruebas de integración es probar las interfaces entre los módulos y exponer cualquier defecto que pueda surgir cuando estos componentes están integrados y necesitan interactuar entre sí.

La principal diferencia entre las pruebas unitarias y las pruebas de integración es que en las pruebas unitarias se prueban módulos individuales. En las pruebas de integración estos módulos se combinan y prueban como una sola unidad para verificar la funcionalidad de la aplicación en general.

Si la arquitectura del sistema es monolítica, es posible hacer uso de las plataformas y herramientas de pruebas unitarias para desarrollar pruebas de integración. De esta forma, se modifica el ámbito de la prueba en el sentido de que lo que se cubre no es una pieza de código está bien desarrollada, sino sus interfaces de entradas y salidas con el resto de los módulos con los que interactúa responden al objetivo perseguido.

En arquitecturas distribuidas, los diferentes módulos o subsistemas interactúan a partir de las API (*application programming interfaces*) definidos con lenguajes o estándares, por lo que en esta sección nos centraremos en aquellas plataformas que de forma específica nos ayudan a generar pruebas a partir de esta distinción de protocolos o a emular el comportamiento de esos sistemas.

Algunas de las herramientas más populares para facilitar la definición y ejecución de pruebas de integración de API son:

A screenshot of a computer program

Description automatically generated

Tabla 3. Plataformas para pruebas de integración. Fuente: elaboración propia.

Pruebas API

Apigee

Es un software de prueba de API del ecosistema de herramienta de Google Cloud, basado en JavaScript que permite a los desarrolladores y evaluadores acceder a sus funciones a través de diferentes editores. Esta herramienta es compatible con API que contienen grandes cantidades de datos, por lo que son ideal para negocios digitales robustos y complejos. Soporta pruebas de rendimiento mediante el análisis del tráfico de la API, los tiempos de respuesta y las posibles tasas de error y también permite la creación de *proxies* a partir de OpenAPI Specification.

Assertible

Se trata de una de las principales herramientas de prueba de API con un alto enfoque en la confiabilidad,considerando que admite pruebas de API en cada paso que va desde la integración continua hasta la canalización de entrega. Soporta la integración con GitHub, Zapier y Slack, y valida las respuestas HTTP con aserciones.

Katalon

Es una popular herramienta de automatización de pruebas para API, así como aplicaciones web y aplicaciones móviles y de escritorio. Admite solicitudes SOAP y REST y proporciona múltiples funciones y comandos de parametrización. Sigue un enfoque basado en datos, admite su vínculo con flujos de integración y despliegue continuos y permite a los probadores ejecutar pruebas exploratorias y automatizadas.

Postman

Aunque inicialmente creada como un complemento para Google Chrome, en la actualidad también se puede usar con Windows y Mac y es una excelente herramienta para probar los servicios de API, incluida la supervisión, la depuración, la ejecución de solicitudes, etc. No requiere que los desarrolladores o evaluadores aprendan un nuevo idioma y además admite varios formatos (como Swagger y RAML). Además, también proporciona herramientas integradas, espacios de trabajo y colecciones.

SoapUI

Es una herramienta dedicada a las pruebas de API que permite ejecutar pruebas sobre las API de SOAP y también sobre los servicios web REST, considerando que es *headless*. A menudo, es una de las herramientas preferidas por los equipos de prueba, ya que les permite crear escenarios complejos y admite aún más las pruebas asíncronas. Además, también se destaca por sus excelentes pruebas basadas en datos y su facilidad de uso.

Para automatizar la generación de pruebas sobre un API REST*,*también podemos hacer uso de la librería Swagger Codegen*,*que permite generar automáticamente la librería de invocación a un sistema especificado mediante. Puedes verlo en el proyecto Swagger Codegena continuación.

Accede al contenido a través del siguiente enlace: <https://github.com/swagger-api/swagger-codegen>

En este vídeo, *Cucumber y OpenAPI, testeando servicios distribuidos*, aprenderemos a automatizar las pruebas sobre interfaces descritos con OpenAPI usando Cucumber.

Emuladores API

Cuando queremos hacer pruebas de integración es posible que no tengamos acceso al código o a un entorno de despliegue de todos los módulos. Este escenario es típico cuando se realiza un desarrollo en paralelo al tener equipos que se encargan de la parte del servidor (*back-end*) y otros de la parte del cliente (*front-end*), así como en sistemas que requieren de integración de servicios externos (por ejemplo, conectarnos a un sistema bancario o a una pasarela de pago de un tercero).

Para evitar los cuellos de botella del desarrollo (tener que esperar a que el *back-end* esté desarrollado para poder crear el *front-end*) o para poder generar escenarios de prueba específicos (por ejemplo, cómo se comportaría mi sistema si el módulo externo no responde al protocolo establecido), surge el concepto de emular, simular o *mockear* un sistema al crear una versión falsa de un servicio externo o interno que pueda reemplazar al real, lo que ayuda a que las pruebas se ejecuten de manera más rápida y confiable.

Algunas de las herramientas más populares para facilitar esta emulación de partes de un sistema son:

Beeceptor

Es una herramienta en línea gratuita para simular una interacción API REST utilizando cualquier solicitud HTTP. Podemos personalizar las respuestas para simular prácticamente cualquier respuesta o situación de falla. También ofrece una excelente funcionalidad para simular la latencia en las API posteriores: podemos simular la latencia de transmisión y los tiempos de espera, lo cual es un excelente caso de uso para muchas API y, a menudo, se pasa por alto durante las pruebas.

JMockit

Es una herramienta de código abierto para pruebas unitarias con la colección de herramientas y API. Los desarrolladores pueden usar estas herramientas y API para escribir pruebas usando TestNG o JUnit. Se plantea como una alternativa al uso convencional de objetos simulados (*mock objects*), ya que proporciona tres tipos de cobertura de código como la cobertura de línea, la cobertura de ruta y la cobertura de datos.

Mockito

Se trata de una librería en Java que, combinada con JUnit*,* facilita el desarrollo de pruebas de integración al emular partes del sistema. A diferencia de las anteriores, muy orientada al testeo de API en sistemas distribuidos, esta herramienta se centra en la simulación de módulos para arquitecturas monolíticas (Grzejszczak, 2014).

MockServer

Es una biblioteca que lo ayuda a emular API en cuestión de segundos, con base en la creación de las respuestas HTTP simuladas en una colección de archivos simulados y los atiende como si vinieran de una API real. De esta forma, el equipo puede desarrollar el lado del cliente sin necesidad de esperar a que el *back-end* está realmente listo o no.

Nock

Es una biblioteca HTTPS diseñada para replicar y simular servidores y expectativas en Node.js. Funcionalmente, Nock hace esto anulando las funciones http.request y http.ClientRequest al interceptar solicitudes y responder con una respuesta simulada específica mediante el uso de interceptores.

WireMock

Es un simulador de API basadas en HTTP. Puede considerarse como una herramienta de virtualización de servicios o un servidor simulado. Permite mantener la productividad del equipo de desarrollo cuando una API de la que depende no existe o no está completa. Admite pruebas de casos extremos y modos de falla que la API real no producirá de manera confiable.

9.6. Plataformas para pruebas de sistema

Las pruebas a nivel de sistema incluyen un amplio espectro, como pruebas de funcionalidad, pruebas de seguridad, pruebas de robustez, pruebas de carga, pruebas de estabilidad, pruebas de estrés, pruebas de rendimiento y pruebas de confiabilidad, entre otras.

En entornos en los que tenemos una interfaz con el usuario, ya sea de escritorio, web o aplicación móvil, las pruebas de interacción generalmente se incluyen en este nivel como pruebas funcionales del sistema.Algunas de las plataformas que ayudan a realizar pruebas de interfaz de usuario son:

A screenshot of a computer

Description automatically generated

Tabla 4. Plataformas para pruebas de sistema. Fuente: elaboración propia.

Selenium

Es una librería con un junto de utilidades que facilita la generación de casos de prueba para aplicaciones web. Nos permite definir casos de prueba con un mayor nivel de abstracción y, de forma combinada con JUnit, lleva a cabo la ejecución automática de las mismas con varios navegadores (Gundecha y Avasarala, 2018).

En este vídeo, *Selenium: testeando interfaces de usuario web*, aprenderemos a automatizar las pruebas sobre interfaces de usuario utilizando Selenium. Accede al vídeo a través del aula virtual

Selenium IDE

Es un proyecto de código abierto que aprovecha el poder de Selenium en un complemento de Chrome y lo pone a disposición de forma gratuita. Comenzar con Selenium IDE no requiere configuración adicional más que instalar la extensión en el navegador, alineándose con la filosofía impulsora del proyecto de proporcionar una herramienta que sea fácil de usar y brinde retroalimentación instantánea. Mediante la extensión, podemos reproducir un caso de prueba de forma manual y se podrá exportar en un *script*, lo que la convierte en una herramienta increíblemente poderosa para la creación de pruebas de interfaz de usuario.

LabdaTest

Es una herramienta comercial que permite escalar las pruebas de Selenium en la nube al utilizar miles de navegadores de escritorio y móviles, lo que genera un entorno de ejecución paralelo en el caso de querer asegurar el funcionamiento en toda la gama de navegadores y dispositivos.

Parasoft Selenic

Es una solución comercial orientada a usuarios de Selenium que dedican mucho esfuerzo a la hora de crear y mantener sus pruebas de interfaz de usuario. Su grabadora inteligente facilita la creación de pruebas de Selenium que utilizan el modelo de objetos de página y se puede integrar en el flujo de integración y distribución continuas (CI/CD).

Abbot Java GUI Test Framework

Es un marco de código abierto que se utiliza para probar la interfaz gráfica de usuario (GUI) de Java proporcionando pruebas unitarias y pruebas funcionales para AWT y SWING.

AutoIt

Es un lenguaje de secuencias de comandos gratuito con estructuras básicas que se utilizan para automatizar la interfaz gráfica de usuario de Windows y las secuencias de comandos generales.

FlaUI

Es una biblioteca .NET que ayuda con las pruebas de interfaz de usuario automatizadas de las aplicaciones de Windows (Win32, WinForms, WPF, Store Apps, etc.). Se basa en las bibliotecas nativas de automatización de la interfaz de usuario de Microsoft y, por lo tanto, es una especie de contenedor que las rodea.

WinAppDriver

Es un servicio similar a Selenium para aplicaciones de Windows. Este servicio admite la prueba de aplicaciones de la plataforma universal de Windows (UWP), Windows Forms (WinForms), Windows Presentation Foundation (WPF) y Classic Windows (Win32) en PC con Windows.

Para poder satisfacer los requisitos no funcionales de rendimiento, es necesario desarrollar pruebas que generen carga en el sistema y lo lleven a situaciones de estrés.

Algunas de las plataformas que ayudan a realizar pruebas de rendimiento son:

Apache JMeter

Es una herramienta de código abierto que se puede utilizar para pruebas de rendimiento y carga para analizar y medir el rendimiento de una variedad de servicios. Es una de las mejores herramientas de prueba de carga que se utiliza principalmente como herramientas de prueba de carga de sitios web para aplicaciones de servicios web (Gomes Rodrigues, Demion y Mouawad, 2019).

SmartMeter.io

Es una alternativa a JMeter y tiene como objetivo solucionar sus inconvenientes. Es una de las mejores herramientas de prueba de carga que permite la creación sencilla de escenarios de prueba sin secuencias de comandos utilizando el llamado grabador, pero aún le permite realizar ediciones avanzadas de la prueba. También sobresale en los informes de prueba y hace uso de funciones como la evaluación automática de criterios de prueba, la comparación de ejecuciones de prueba y el análisis de tendencias. Es totalmente compatible con la integración de CI/CD y está disponible para Windows, macOS y Linux.

LoadNinja

Permite crear rápidamente pruebas de carga sofisticadas sin secuencias de comandos, reducir el tiempo de prueba en un 50 %, reemplazar los emuladores de carga con navegadores reales y obtener métricas procesables basadas en navegadores, todo a la velocidad de un ninja. Puede capturar fácilmente interacciones del lado del cliente, depurar en tiempo real e identificar problemas de rendimiento de inmediato.

K6

Es una herramienta de prueba de carga gratuita y de código abierto que tiene como objetivo hacer que la prueba de rendimiento sea una experiencia productiva y agradable. Se trata de una herramienta de línea de comandos disponible para Windows, Linux y Mac. Alternativamente, hay una imagen oficial de Docker. La herramienta admite secuencias de comandos y las pruebas k6 están escritas en JavaScript, lo que brinda la posibilidad de escribir sus pruebas en uno de los lenguajes de programación más conocidos.

En este vídeo*, K6 y openAPI, pruebas de carga de servicios distribuidos,* aprenderemos a utilizar K6 como plataforma para automatizar pruebas de carga sobre sistemas distribuidos.

StresStimulus

Se orienta a escenarios de aplicaciones que son difíciles de probar con otras herramientas. Corrige automáticamente los errores de reproducción gracias a su autocorrelación patentada. Registra las acciones del usuario y las reproduce para emular patrones de uso variables. También supervisa el impacto de la carga en la capacidad de respuesta de las aplicaciones y la infraestructura del servidor. El analizador de resultados encuentra errores de concurrencia ocultos que no quedaron expuestos en las pruebas funcionales e incluye métricas detalladas de rendimiento de la aplicación.

WebLOAD

Es una solución de prueba de carga a escala empresarial capaz de generar escenarios de carga reales y confiables, incluso para los sistemas más complejos, con análisis inteligentes que brindan información sobre el rendimiento. Se basa en una plataforma flexible con soporte integrado para cientos de tecnologías e integración con muchas herramientas, desde canalizaciones de CI/CD hasta monitoreo.

Neoload

Es una herramienta de prueba de estrés y carga de alta eficiencia y bajo precio que se utiliza para medir el rendimiento de las aplicaciones web y móviles. Simula el tráfico a través de usuarios virtuales para determinar el rendimiento de la aplicación bajo carga y analizar los tiempos de respuesta de la transacción e identificar la cantidad de usuarios simultáneos que Internet, la intranet o la aplicación móvil pueden manejar. Una de sus ventajas es que el trabajo del generador y evaluador de las pruebas es más fácil en comparación con otras herramientas.

Kobiton

Ofrece opciones flexibles de acceso a dispositivos reales que incluyen una nube pública, una nube privada y local. Los usuarios pueden integrar pruebas de dispositivos reales con soluciones de pruebas de carga y estrés. Esto se puede hacer manualmente, con automatización a través de soluciones basadas en *scripts*como Appium y utilizando el motor sin *scripts* impulsado por su inteligencia artificial (IA). Da soporte para analizar las métricas de rendimiento específicas de la aplicación y el dispositivo y se integra fácilmente con soluciones como NeoLoad o JMeter.

9.7. Plataformas para pruebas de aceptación

Las pruebas de aceptación son aquellas realizadas generalmente por usuarios, clientes o quienes los representan para determinar si el software generado cumple las especificaciones. Son las encargadas de validar el sistema y, en ocasiones, determinan el hito de que el cliente acepte o no una entrega del software.

En general, cuando nos referimos a pruebas de aceptación estamos hablando de validar un sistema, es decir, comprobar que el software desarrollado es conforme a unos requisitos o necesidades del cliente o usuario. No es posible diseñar pruebas de aceptación sin conocer esas especificaciones, por lo que la tendencia en la ingeniería del software ha sido anticipar la escritura de los casos de prueba de validación antes de diseñar y construir el software y utilizar lenguajes que puedan ser comprensibles por el personal técnico y por el usuario o cliente.

Dentro de las pruebas de aceptación vamos a incluir aquellas que permiten la ejecución de escenarios de pruebas a partir de la definición de especificaciones en lenguajes específicos de dominio que pueden ser automáticamente procesados, las que facilitan la automatización de la interacción con el usuario, así como aquellas suites o conjunto de herramientas que dan soporte a todo el ciclo de vida de las pruebas:

A blue and white rectangular box with black text

Description automatically generated

Tabla 5. Plataformas de pruebas de aceptación. Fuente: elaboración propia.

En el contexto de las pruebas del software, las pruebas de aceptación requieren tener unas especificaciones o requisitos que establezcan qué es lo que se esperaba que hiciera el software.

El desarrollo dirigido por comportamiento (*behavioral driven development*, BDD) es un enfoque de desarrollo de software que ha evolucionado desde el desarrollo dirigido por las pruebas (*test driven development*, TDD). Se diferencia por estar escrito en un lenguaje compartido, lo que mejora la comunicación entre los equipos tecnológicos y no tecnológicos y las partes interesadas. En ambos enfoques de desarrollo, las pruebas se escriben antes que el código, pero en BDD, las pruebas están más centradas en el usuario y se basan en el comportamiento del sistema (Smart, 2014).

La gran ventaja del enfoque de BDD es el uso de un lenguaje común que es fácilmente comprensible por todas las partes. A nivel práctico, los requisitos se convierten en historias de usuario que ayudan a comprender cómo se desarrolla cada funcionalidad. Seguidamente, se definen los escenarios que permiten comprender todas las situaciones en las que se puede desarrollar la historia de usuario, en lugar de un caso concreto (Nagy y Rose, 2018). Para describir un escenario de prueba se utiliza la fórmula *Given-When-Then*(Nagy y Rose, 2021), de esta forma:

* Ante (*given*) un escenario determinado
* Cuando (*when*) se lleva a cabo una acción
* Entonces (*then*) este debería ser el resultado.

El lenguaje más popular para describir estos escenarios es Gherkin, que soporta más de setenta lenguajes y se basa en las siguientes anotaciones (Cucumber, s. f.):

A screenshot of a computer

Description automatically generated

Tabla 6. Palabras clave del lenguaje Gherkin. Fuente: elaboración propia.

Si tomamos el ejemplo de la calculadora, seguidamente tenemos un ejemplo de cómo podríamos especificar pruebas de aceptación para la funcionalidad de sumar números utilizando Gherkin (en su extensión para nuestro idioma).

* Característica: sumar números con una calculadora
  + Para no aprender matemáticas
  + Como alguien que es malo en matemáticas
  + Quiero poder sumar números usando una calculadora
* Escenario: sumar dos números positivos
  + Dado que tengo una calculadora
  + Cuando sumo 1 y 1
  + Entonces la suma debe ser 2
* Escenario: sumar un número positivo y negativo
  + Dado que tengo una calculadora
  + Cuando sumo 1 y -1
  + Entonces la suma debe ser 0
* Escenario: sumar dos números negativos
  + Dado que tengo una calculadora
  + Cuando sumo -1 y -1
  + Entonces la suma debe ser -2

Con Gherkin definimos casos de prueba que definen qué esperamos que haga el software en un determinado contexto, siendo comprensible por el equipo técnico y por el usuario o cliente.

Existen herramientas que nos ayudan a automatizar los casos de prueba a partir de la descripción Gherkin de los escenarios:

Cucumber

Es probablemente la herramienta disponible más reconocida que admite BDD (Chaubal, 2021). Se basa en escenarios de prueba que están escritos en texto y siguen el formato *given/when/then*. Cada escenario es una prueba individual. Los escenarios se agrupan en características, lo que es comparable a un conjunto de pruebas. Los escenarios deben estar escritos en la sintaxis de Gherkin para que la herramienta comprenda y ejecute los pasos del escenario. Los pasos legibles por humanos en los escenarios están vinculados a las definiciones de pasos en su código a través de la propia herramienta. Debido a que estas pruebas se ejecutan a partir de las definiciones de los pasos, los informes con los resultados de las pruebas permiten ver en qué parte del paso ha fallado la prueba. La herramienta se escribió originalmente en el lenguaje de programación Ruby y se usó exclusivamente para probar Ruby como complemento del marco RSpec BDD. En la actualidad, admite una variedad de diferentes lenguajes de programación, incluidos Java, JavaScript, PHP, Net, Python, Perl, etc. con varias implementaciones.

En este vídeo, *Cucumber, definiendo pruebas con Gherkin*, aprenderemos a automatizar las pruebas sobre interfaces descritos con OpenAPI usando Cucumber

SpecFlow

Se lo considera el Cucumber para .NET, surge del anterior y es muy similar en concepto. Se basa en texto con formato *Gherkin* para describir las historias de los usuarios y se orienta a probar soluciones .NET, por lo que puede integrarse con Visual Studio.

JBehave

Es muy similar a Cucumber, se puede usar la descripción de escenarios con Gherkin aunque también tiene su propia sintaxis. Frente a Cucumber, tiene más opciones de configuración y sus informes, aunque muy detallados, necesitan más configuración para obtener retroalimentación de cada paso. Es una herramienta poderosa, pero debido a que se puede personalizar más, no es tan fácil comenzar con ella. Los equipos deben preguntarse exactamente qué funciones necesitan y si vale la pena invertir tiempo en aprender las distintas configuraciones de la herramienta.

Plataformas de prueba para aplicaciones móviles

Aunque no son exclusivas para pruebas de aceptación, las plataformas de pruebas de aplicaciones móviles pueden ser utilizadas para este enfoque de alto nivel, ya que van más allá de una mera prueba de interacción con el usuario. Con carácter general, el testeo de aplicaciones móviles puede cubrir:

* Pruebas funcionales: tipo básico de prueba utilizado para verificar las funcionalidades de la aplicación según la especificación de requisitos.
* Pruebas de rendimiento: realizadas para probar el rendimiento de la aplicación del cliente, el rendimiento del servidor y el rendimiento de la red.
* Prueba de memoria: los dispositivos móviles vienen con memoria limitada en comparación con las computadoras, este tipo de prueba se realiza para probar el uso de memoria optimizado por parte de una aplicación.
* Prueba de interrupción: se usa para verificar interrupciones debido a llamadas entrantes o SMS, advertencia de memoria baja, advertencia de batería baja, etc. mientras se ejecuta la aplicación.
* Pruebas de instalación: se utilizan para verificar que el proceso de instalación sea fácil y fluido e incluye también la actualización y la desinstalación.
* Pruebas de usabilidad: se utilizan para comprobar la eficiencia, eficacia y satisfacción de la aplicación desde la perspectiva del usuario.

Dentro de este campo de plataformas para pruebas de aplicaciones móviles, encontramos las siguientes plataformas:

Appium

Es una herramienta de código abierto para la automatización de pruebas de aplicaciones nativas, móviles y web, así como híbridas en plataformas iOS y Android. Maneja aplicaciones de iOS, Android y Windows, ya que utiliza el protocolo WebDriver. Admite Safari en iOS y todas las demás aplicaciones de navegador integradas en Android. No es necesario modificar ningún código de aplicación para realizar pruebas, ya que es adecuado para ejecutarse en Android o iOS usando el dispositivo o el emulador (Hans, 2015).

UI Automator

Es un marco de código abierto, integrado en Android, que permite probar la interfaz de usuario utilizando casos de prueba funcionales automatizados. Facilita el desarrollo de pruebas funcionales entre aplicaciones en todo el sistema y en aplicaciones instaladas.

iOS-driver

Es una herramienta de código abierto para facilitar las pruebas de cualquier aplicación web nativa, híbrida o móvil de iOS utilizando la API de Selenium/WebDriver. Se ejecuta de manera eficiente en un emulador en lugar de dispositivos y no es necesario cambiar ningún código de aplicación ni cargar ninguna aplicación adicional para probar la aplicación en el dispositivo.

Selendroid

Selenium para Android es un marco de código abierto que interactúa simultáneamente con múltiples dispositivos y emuladores. Está impulsado por la interfaz de usuario de aplicaciones nativas e híbridas y por la web móvil, por lo que la prueba debe escribirse a través de la API del cliente Selenium. Se puede usar en emuladores y dispositivos reales y se puede integrar como un nodo en Selenium Grid para escalar y realizar pruebas en paralelo.

Frank

Es un marco de prueba exclusivo para iOS de código abierto que representa características combinadas de Cucumber y JSON. Ayuda a escribir pruebas y requisitos de aceptación estructurados y también incluye el inspector de aplicaciones Symbiote. No hay necesidad de modificar el código de la aplicación. El único desafío es que es difícil de usar directamente en el dispositivo, pero es más adecuado para aplicaciones nativas y web.

RobusTest

Es una herramienta de prueba móvil gratuita que proporciona pruebas basadas en la nube en dispositivos reales para Android e iOS. Soporta pruebas manuales rápidas y avanzadas automatizadas, pruebas de automatización sin secuencias de comandos, pruebas funcionales y de rendimiento. Ayuda a realizar pruebas de automatización rápidas junto con integración continua y API abierta.

Herramientas destacadas en la actualidad

En esta sección vamos a incluir algunas de las plataformas o herramientas destacadas en la actualidad que cubren varias de las fases anteriormente mencionadas, ya que pueden ser utilizadas por los equipos de desarrollo o los clientes para validar un software de forma completa.

Tricentis

Valorada por Gartner como la plataforma de pruebas continuas referente del mundo, incluye un conjunto único de capacidades que respaldan el ciclo de vida completo de las pruebas en aplicaciones empresariales y personalizadas. Permite acelerar todas las fases de las pruebas, desde las unitarias hasta las de rendimiento. No es necesario desarrollar código lo que hace que sea fácil de usar para usuarios comerciales no técnicos y miembros del equipo. Es impulsada por inteligencia artificial, por lo que se aprovechan las tecnologías avanzadas de *machine learning* para permitir pruebas más rápidas e inteligentes.

SmartBear TestComplete

Es una aplicación que permite la creación de pruebas de interfaz de usuario automatizadas para entornos de escritorio, web y móvil. Permite la definición de pruebas basadas en datos y en palabras clave, que se integran de forma continua en entornos de desarrollo y operaciones (DevOps). Genera informes y análisis de pruebas automatizados, lo que da soporte a pruebas paralelas y con un enfoque de desarrollo dirigido por comportamiento. Con las pruebas entre navegadores y entre dispositivos, podemos registrar fácilmente una prueba de interfaz de usuario de forma automatizada y luego ejecutar la misma prueba en otros navegadores o dispositivos reales localmente o en la nube. Estas pruebas se pueden ejecutar en plataformas Windows, Mac, Linux, Android e iOS en todas las configuraciones de navegador, pudiendo usar la reproducción en JavaScript, Python o VBScript.

Eggplant

Es una solución comercial formada por diversas herramientas que permite automatizar las pruebas. A nivel de rendimiento, facilita la creación de *scripts* de prueba, identifica desafíos en la experiencia del usuario, monitorea tiempos de inactividad y analizar resultados. A su vez, permite a los analistas de datos generar automáticamente y simular *scripts* de prueba de viaje del usuario para monitorear el rendimiento general de la aplicación. Les posibilita a los evaluadores colocar inyectores en múltiples regiones geográficas para particionar usuarios virtuales y analizar la experiencia del usuario en función de las ubicaciones de los clientes. A nivel funcional, facilita probar sus aplicaciones mejor, más rápido y con menos esfuerzo al automatizar la ejecución de sus pruebas funcionales. Destaca su enfoque patentado basado en imágenes para las pruebas de GUI. Permite interactuar con cualquier dispositivo (incluidos dispositivos móviles, tabletas, computadoras de escritorio, servidores y dispositivos de Internet de las cosas) de la misma manera que lo hace un usuario, mirando la pantalla, y utiliza sofisticados algoritmos de búsqueda de imágenes y texto para ubicar objetos en la pantalla de una manera completamente independiente de la tecnología.

QMetry Automation Studio (QAS)

Ofrece automatización de pruebas escalable que permite una fácil transición de pruebas manuales a automatizadas. Admite pruebas multiplataforma en la web, servicios web/API, dispositivos móviles y escritorio de Windows. QAS también utiliza marcos de automatización de pruebas de código abierto como Selenium y Appium. Admite múltiples navegadores para la ejecución de pruebas, múltiples marcos, integraciones de ALM y complementos para escalar sus pruebas automatizadas. Se integra con herramientas de CI/CD como Jenkins, Bamboo y plataformas web y de pruebas en la nube de dispositivos como Sauce Labs y Perfecto Mobile.

TestProject

Es una herramienta de automatización de pruebas gratuita de extremo a extremo para web, móviles y API. Con una comunidad activa que desarrolla más de 200 complementos y posicionada como referente por la consultora Gartner, soporta un uso en modo grabación para personal no técnico, un kit de desarrollo de software (SDK) de comandos avanzados que importa pruebas desde Selenium y Appium, un almacenamiento de las pruebas en la nube, la generación de cuadros de mando y el análisis ejecutivos e integración con herramientas como SauceLabs, BrowserStack, Jenkins o Slack.

Micro Focus UTF

La familia UTF de soluciones de automatización de pruebas funcionales basadas en IA se compone de tres herramientas:

* UFT Uno, que acelera la automatización de pruebas con una solución inteligente para aplicaciones web, móviles, RPA (automatización robótica de procesos), API y empresariales.
* UTF Desarrollador, que facilita la programación de casos de prueba utilizando lenguajes de programación estándar, entornos de desarrollo integrado (IDE) y marcos de pruebas.
* Móvil UFT, que aumenta la productividad del equipo con un laboratorio integral de nivel empresarial de emuladores y dispositivos móviles reales.

Ranorex Studio

Es un conjunto de herramientas completo para realizar pruebas de extremo a extremo de aplicaciones de escritorio, web y móviles. Permite automatizar las pruebas en un escritorio de Windows y luego ejecutarlas de forma local o remota en dispositivos móviles o simuladores/emuladores iOS o Android reales. Soporta la ejecución de pruebas en paralelo con múltiples navegadores como Chrome, Firefox, Safari, Microsoft Edge y más. Se integra con Jira y otras herramientas líderes de DevOps para automatizar el seguimiento de errores y acelerar los ciclos de desarrollo.

Worksoft Certify

Es una *suite* comercial de automatización de pruebas. No es necesariogenerar código,lo que les permite a los usuarios comerciales y a TI crear, mantener, compartir y consumir fácilmente pruebas de procesos comerciales automatizados de extremo a extremo como parte de los ciclos continuos de prueba, integración y entrega. El marco de acción de objetos basado en modelos actualiza dinámicamente la automatización en todo su entorno de aplicaciones, lo que ofrece una solución verdaderamente escalable para acelerar los tiempos de entrega y reducir el riesgo.

Froglogic Squish GUI Tester

Es una herramienta para la automatización de las pruebas web, integradas, móviles y de escritorio multiplataforma. Soporta como tecnologías a probar Qt, QtQuick, QML, Java, AWT, Swing, SWT, RCP, JavaFx, Windows, Mac, Tk, 4Js, iOS, Android, sistemas embebidos (Qt, Windows, Java) y web, HTML5, Flex y JavaApplets. Ejecuta pruebas en Windows, Linux, macOS X, Unix. Soporta pruebas móviles e integradas en Android, iOS, Linux integrado y RTOS como QNX. Dispone de un IDE basado en Eclipse y puede usarse con un enfoque de desarrollo dirigido por el comportamiento, con grabación, reproducción e informes de archivos de funciones.