Los desafíos en las pruebas automáticas de las interfaces gráficas de usuario

David Maloof-Flores*
ORCID: 0000-0002-0404-791X

Olanda Prieto-Ordaz*
ORCID: 0000-0002-0023-9535

Miguel Ángel López-Santillán*
ORCID: 0009-0008-2087-1168

La GUI (interfaz gráfica de usuario) es una forma de software que hace uso de las capacidades visuales de la computadora al simplificar la interacción con las aplicaciones que usamos día a día (Yuan, Cohen y Memon, 2010). Las interfaces gráficas ofrecen a los usuarios una amplia variedad de caminos que ayudan a interactuar con el software; a pesar de que hacen que el software sea más amigable y sencillo de manejar, también introducen complejidades en el proceso de desarrollo. A fin de cuentas, es bien sabido que el resultado de tener una bien diseñada GUI atrae la atención de los usuarios en el mercado (Zheng, Hu y Ma, 2019).

Por este motivo, la ejecución de pruebas se efectúa con el objetivo de verificar que la interfaz cumpla con las especificaciones y funcione correctamente. Éstas involucran la ejecución de acciones y la comparación de los resultados obtenidos con los anticipados, mediante el uso de lo que se conoce en la ingeniería de software como "casos de prueba". Las verificaciones pueden desarrollarse de forma manual por seres humanos o de manera automatizada. Adicionalmente, la llegada abrupta de herramientas de inteligencia artificial (AI) de generación de código automático ha invadido la web con la finalidad de acelerar el desarrollo de aplicaciones (Ponce, 2023). Es por tal razón que la evaluación del software se ha convertido en una cuestión de gran relevancia.

La etapa de pruebas es una actividad fundamental para asegurar la calidad en el proceso de desarrollo de software. La ingeniería del software sostiene la premisa fundamental desde sus inicios: "la calidad del producto está estrechamente ligada a la del proceso" (Humphrey, 2005). Dicha afirmación resalta la significativa influencia de la etapa de "pruebas" en la culminación de un software de excelencia.

https://doi.org/10.29105/cienciauanl27.127-3

* Universidad Autónoma de Chihuahua, Chihuahua, México.
Contacto: dmaloof@uach.mx, oordaz@uach.mx, mlopezs@uach.mx

Durante esta fase, se crea y ejecuta un conjunto de casos de ensayo en una aplicación bajo observación. En el transcurso de éstas se emplean "oráculos" (*test oracles*, por su definición en inglés), que no son más que un método cuyo objetivo es comprobar si el sistema evaluado se ha comportado correctamente en una ejecución particular según lo previsto. En pocas palabras, determina si el resultado de correr un programa es acertado. En este sentido, se refiere a expectativa, aceptabilidad y precisión (Monperrus, 2017). Estos oráculos pueden ser automatizados o manuales; en ambos casos, se compara la salida real con una esperada que se considera adecuada.

La proporción de casos destinados a examinar las GUI en una aplicación dependen directamente de la cantidad de código asignado a la interfaz gráfica, donde varían desde un pequeño porcentaje en aplicaciones orientadas a servicios o procesamiento de datos, hasta uno sustancial en aquéllas centradas en la experiencia del usuario, como software de diseño gráfico o videojuegos. En general, la importancia de la interfaz gráfica ha aumentado con el tiempo debido a la creciente atención a la experiencia del usuario, abarcando todos los aspectos de la interacción con un producto (y por ende, su éxito). Su complejidad no puede ser subestimada (Hassenzahl, 2018) y con ello, la necesidad de obtener un funcionamiento libre de errores.

En este contexto, las pruebas manuales son llevadas a cabo por individuos, probadores o desarrolladores, pero presentan vulnerabilidad a errores y la posibilidad de omitir diversos escenarios. Además, consumen una cantidad significativa de tiempo (Retna y Retna, 2012), ya que son repetitivas y susceptibles a errores cuando son realizadas por un evaluador humano. Para abordar estos desafíos, se han propuesto soluciones en forma de automatización, haciendo uso de exámenes unitarios y reproducción de grabaciones, las cuales han demostrado ser exitosas en la práctica. A esto se le llama "pruebas automatizadas de GUI", que consisten en la automatización de tareas que previamente se llevaban a cabo de forma manual, y que generalmente resultan más eficientes, precisas, confiables y rentables (Adamoli *et al.*, 2011).

Indistintamente, en ambos escenarios es necesario contar con un conjunto de casos con directrices y metas detalladas, donde cada uno generalmente incluye una entrada, una salida, un resultado anticipado y uno real. Diversos casos son requeridos en la evaluación de todas las funcionalidades de la GUI.

EL DESARROLLO DE LOS CASOS DE PRUEBA PARA GUI

En la práctica se utilizan distintos enfoques al crear casos de prueba para las GUI, siendo la más popular el "oráculo manual", es decir, un evaluador que interactúa personalmente con la GUI, realizando eventos (acciones llevadas a cabo por el usuario) como hacer clic en un botón o escribir algo en un campo de texto, posteriormente comprobar visualmente si la GUI actuó de manera esperada y documentar los posibles errores. Proporcionar un oráculo de prueba preciso es el principal requisito previo si se desea lograr técnicas y herramientas de evaluación de software robustas y realistas (Valueian *et al.*, 2020).

Las técnicas y herramientas más utilizadas desde hace décadas en el análisis de una interfaz gráfica requieren se modele y segmente la GUI en componentes llamados *widgets*: elementos visuales básicos, con sus respectivas propiedades (Xie y Memon, 2007). También se necesita que los eventos de la GUI sean deterministas con el objetivo de predecir su resultado. Por lo tanto, para proporcionar un enfoque de cómo es el proceso básico de automatización, en este artículo se ejemplifica una subclase particular de GUI, que se define a continuación.

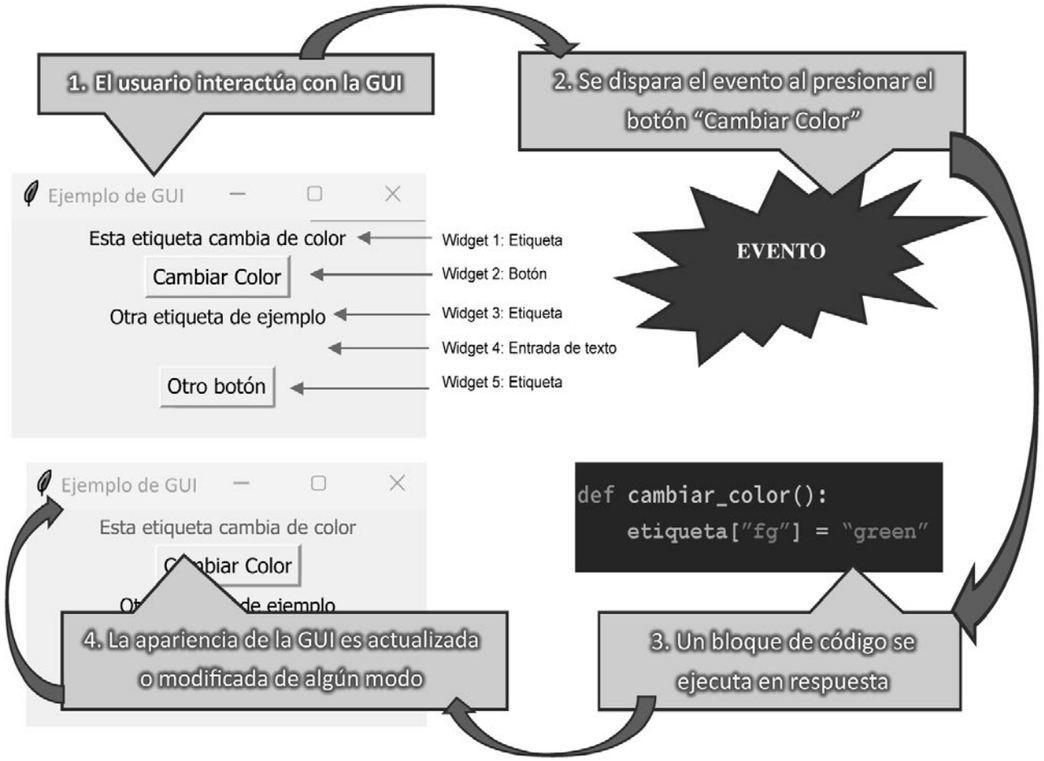

Figura 1. Ejemplo de la ejecución de un evento y descripción de los componentes de una interfaz gráfica de usuario hecha con Tkinter en Python.

Según el diagrama de la figura 1, se presenta una GUI jerárquica de un software que acepta como entrada eventos generados por el usuario y por el sistema, a partir de un conjunto finito de éstos, y produce una salida gráfica determinista. De igual manera, es posible clasificar la interfaz de usuario con los elementos que contiene: los *widgets*. Cada *widget* tiene un conjunto fijo de propiedades. En cualquier momento, durante la ejecución de la GUI, estas propiedades tienen valores discretos, cuyo conjunto constituye el estado de la GUI en un momento determinado y se modela como un conjunto de *widgets* $W = \{w_1, w_2, ..., wn\}$ (por ejemplo, botones, campos de texto), uno de sus respectivas propiedades $P = \{p1, p2..., pn\}$ (color de fondo, tamaño, fuente) y uno de valores asignados a éstas $V = \{v1, v2..., vn\}$ (negro, cursiva, 12pt, etcétera), asociados con las de los mismos elementos. Por lo tanto, la GUI se puede describir completamente en términos de los *widgets* específicos que contiene actualmente y los valores de sus propiedades.

Entonces, el estado de una GUI en un momento particular t es el conjunto S de la terna $\{(w_i, p_j, v_k)\}$, donde $wi \in W, p_j \in P, v_k \in V$; de igual manera, un conjunto de eventos $E = \{e_1, e_2, ..., en\}$ que representan la función que cambia de un estado a otro a la GUI, pudiendo entonces describir al conjunto de pruebas T como el de pares ordenados $T = \{(S_0, e_1), (S_0, e_2), ..., (S_n, e_n)\}$. Es así que la descripción del estado completo contendría información sobre todas las propiedades y sus valores en cada uno de esos w*idgets*, antes y después de un evento. Los casos de prueba se encargan de comparar el

conjunto de la terna de los elementos de cada *widget*, posterior a la ejecución de un evento, con el conjunto de resultados esperados. Este enfoque presenta ventajas: generación de conjuntos de pruebas que exploran la secuencia de los eventos, además de la capacidad de proporcionar casos viables. Sin embargo, es importante señalar que manualmente no es posible escalarlo a aplicaciones más grandes debido al incremento mayúsculo de estados.

Es aquí donde las herramientas de inteligencia artificial (IA) y de automatización permiten la comprobación automática de las GUI que presenten cantidades inmensas de eventos. Éstas son llamadas "Pruebas de GUI automatizadas". Algunas de estas herramientas utilizan IA para simplificar y mejorarlas. Entre las características y capacidades comunes de éstas resaltan:

1. Grabación y reproducción de acciones del usuario: permiten grabar las acciones del usuario en una GUI y luego reproducirlas automáticamente, lo que facilita la creación rápida de *scripts* de verificación.

2. Reconocimiento de elementos de la GUI: utilizan técnicas de reconocimiento de objetos y patrones que identifican y manipulan elementos de la interfaz gráfica: botones, campos de entrada y menús.

3. Pruebas de rendimiento: pueden evaluar el rendimiento de la GUI, detectando problemas de velocidad y capacidad de respuesta.

4. Generación de datos de prueba: pueden generar automáticamente datos en diferentes escenarios, lo que ayuda a probar una amplia gama de condiciones.

5. Pruebas de compatibilidad de múltiples plataformas: algunas herramientas permiten examinar diversas plataformas y navegadores.

Por mencionar algunas herramientas populares, aunque costosas, que utilizan inteligencia artificial y técnicas avanzadas en las pruebas automáticas de GUI se encuentran:

- Selenium: una de las herramientas más populares y ampliamente utilizadas. Emplea IA y aprendizaje automático para mejorar las pruebas de GUI.

- TestComplete: ofrece detección inteligente de elementos de la interfaz de usuario y la generación de *scripts* de prueba.

- Applitools: compara y valida visualmente la apariencia de las GUI en diferentes dispositivos y resoluciones.

- Test.ai: utiliza la IA al realizar evaluaciones de GUI en aplicaciones móviles.

- Mabl: emplea técnicas de aprendizaje automático para crear y mantener automáticamente casos de prueba de GUI.

ACTUALIDAD DE LAS PRUEBAS AUTOMÁTICAS

A pesar de buscar que las comprobaciones automatizadas se conviertan en el estándar de la industria para verificar el funcionamiento de las GUI, todavía es común que las manuales sean ejecutadas sin ningún grado de automatización. Esta preferencia sugiere la existencia de obstáculos específicos, observación respaldada en la bibliografía (Nass *et al.*, 2021).

Entre los principales desafíos en la automatización de pruebas GUI que enfrentan las actuales herramientas que se apoyan de la inteligencia artificial se encuentran:

- Elementos gráficos que no tienen un identificador único en todos los lenguajes de programación (por ejemplo, los TextBox, ComboBox, RadioButton, etcétera).

- Elementos gráficos que se encuentran prefijados y diseñados visualmente diferentes en un software en específico.

- Aplicaciones desarrolladas en múltiples idiomas y sistemas operativos.

- Elementos gráficos y controles sin el adecuado orden Z (Z-order: una ordenación de objetos bidimensionales superpuestos, por ejemplo, colocar elementos gráficos que impidan visualizar un botón).

- Problemas de sincronización entre la herramienta que evalúa la GUI y la actualización de la misma al ejecutar eventos.

Estas revisiones bibliográficas también destacan diversos beneficios, como la reducción del esfuerzo humano y una mayor capacidad de detección de fallas. Sin embargo, señalan enfáticamente que la automatización no puede reemplazar por completo las pruebas manuales y advierten sobre la generación de expectativas poco realistas.

CONCLUSIONES

Es escaso el número de publicaciones de investigación pertinentes a este tema en particular, debido a la complejidad que significa evaluar todos los escenarios posibles que se pueden representar en una interfaz gráfica, y aunque son modelos deterministas, manifiestan retos importantes en implementar esquemas de automatización. Abordar esta limitación actual de investigación y bibliografía en el tema exigirá una colaboración estrecha entre la industria y la academia, ya que la demostración empírica es necesaria si se quiere obtener una comprensión más profunda para conquistar eventualmente los desafíos y restricciones actuales de la automatización de pruebas de GUI. A fin de cuentas, la industria de software será la que tendrá la última palabra en decidir si los análisis automáticos de la GUI representan una verdadera mejora en eficacia y tiempos de producción sobre las pruebas tradicionales hechas por humanos.

Por esta razón resulta relevante mencionar que, aunque los beneficios de utilizar herramientas de automatización de pruebas en las GUI provienen de fuentes de evidencia sólidas (como experimentos y estudios de casos respaldados en la bibliografía), precisamente las desventajas y limitaciones se documentan con mayor frecuencia en los informes de experiencias de la industria al momento de aplicar dichas herramientas en el desarrollo de un software comercial. Esto genera un evidente sesgo en las publicaciones de las desventajas con respecto a los beneficios.

Lo más importante es que persevere la idea de que a pesar de los desafíos aquí mencionados, con el continuo avance de aplicaciones de inteligencia artificial, indudablemente se logre una mayor adopción de esquemas automatizados para verificar el correcto funcionamiento de las interfaces gráficas.

REFERENCIAS

Adamoli, A., Zaparanuks, D., Jovic, M., *et al.* (2011). Automated GUI performance testing, *Software Qual J*, 801-839, doi:10.1007/s11219-011-9135-x

Hassenzahl, M. (2018). The Thing and I: Understanding the Relationship Between User and Product. En M. Blythe y A. Monk, *Funology 2 From Usability to Enjoyment* (301-313). Springer. doi:10.1007/978-3-319-68213-6_19

Humphrey, W.S. (2005). Acquiring Quality Software, *CrossTalk The Journal of Defense Software Engineering*, 18, 19-23.

Monperrus, M. (2017). Automatic Software Repair: a Bibliography, *ACM Computing Surveys*, 51, 1-24. ACM. doi:10.1145/3105906

Nass, M., Alégroth, E., y Feldt, R. (2021). Why many challenges with GUI test automation (will) remain, *Information and Software Technology*, 138.

Ponce, R. (03 de agosto de 2023). *Best AI tools of 2023*. Techradar Pro, the Business Technology Experts, https://www.techradar.com/best/best-ai-tools

Retna, I., y Retna, E. (2012). Study paper on test case generation for gui based testing, *International Journal of Software Engineering & Applications (IJSEA)*, 3(1), 139-147.

Valueian, M., Attar, N., Haghighi, H., *et al.* (2020). Constructing automated test oracle for low observable software, Scientia Iranica, *Transactions on Computer Science & Engineering and Electrical Engineering*, 1333-1351.

Xie, Q., y Memon, A.M. (2007). Designing and Comparing Automated Test Oracles for, *ACM Transactions on Software Engineering and Methodology,* 16(1).

Yuan, X., Cohen, M.B., y Memon, A.M. (2010). GUI Interaction Testing: Incorporating Event Context, *IEEE Transactions on Software Engineering*, 99.

Zheng, S., Hu, Z., y Ma, Y. (2019). Faceoff: Assisting the manifestation design of web graphical user interface, *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining* (774-777), Melbourne, VIC, Australia: ACM.

Recibido: 25/01/2024
Aceptado: 24/05/2024

Descarga aquí nuestra versión digital.