UNIVERSIDAD DE CUENCA

Facultad de Ingeniería

Carrera de Ingeniería de Sistemas

Validación Basada en Pruebas de una Interfaz Gráfica de Usuario en un Entorno de Desarrollo Dirigido Por Modelos

Trabajo de titulación previo a la obtención del título de Ingeniero en Sistemas

Autor:

Bryan Andrés Alba Sarango

CI: 1401010994

Correo electrónico: a.alba2509@gmail.com

Directora:

Ing. María Fernanda Granda Juca, PhD.

CI: 0702952441

Codirector:

Ing. Otto Parra González, PhD.

CI: 0102214749

Cuenca, Ecuador

20 de agosto de 2021

Resumen:

En la etapa de pruebas de software, es posible beneficiarse de la combinación de los requisitos con las actividades de especificación de pruebas. Por un lado, la especificación de las pruebas requerirá menos esfuerzo manual, ya que se definen o generan automáticamente a partir de la especificación de requisitos. Por otro lado, la propia especificación de requisitos terminará teniendo una mayor calidad debido al uso de un lenguaje más estructurado, reduciendo problemas típicos como la ambigüedad, inconsistencia e inexactitud. Esta investigación propone un marco metodológico basado en modelos y su herramienta de soporte UI-Test, que promueven la práctica de generar casos de prueba basados en la especificación de historias de usuarios ágiles para validar que los requisitos funcionales estén incluidos en la versión final de las interfaces de usuario del software desarrollado. Para mostrar la aplicabilidad del enfoque, se utilizan una especificación de requisitos basada en historias de usuarios, un modelo de tarea que usa la notación ConcurTaskTree y el lenguaje Sikulix para generar pruebas a nivel de interfaz gráfica. La propuesta descrita en este trabajo de titulación hace uso de dos transformaciones de modelos para obtener los scripts de prueba de las historias de usuario que se aplicarán en el proceso utilizando SikuliX para las pruebas de IU (Interfaz de Usuario) visual automatizadas. Los resultados de la evaluación empírica de la efectividad y la experiencia del usuario de la solución propuesta y su herramienta de soporte sugieren que la herramienta UI-Test puede beneficiar a los testers al confirmar que las acciones propuestas en las historias de usuario se pueden ejecutar en las IUs.

Palabras claves: UI-Test. Prueba de Interfaz de Usuario Visual. Historias de Usuario. Scripts de Prueba. Pruebas Basadas en Modelos. Marco de Pruebas.

Abstract:

In the software testing stage, it is possible to benefit from combining the requirements with the testing specification activities. On the one hand, the specification of the tests will require less manual effort, since they are defined or generated automatically from the requirements specification. On the other hand, the specification of requirements itself will end up having a higher quality due to the use of a more structured language, reducing typical problems such as ambiguity, inconsistency, and inaccuracy. In this research work, the UI-Test model-based methodological framework and its tool support that promotes the practice of generating test cases based on the specification of Agile user stories to validate that the functional requirements are included in the final version of the user interfaces of the developed software. To show the applicability of the approach, a specification of requirements based on user stories, a task model using the notation ConcurTaskTree, and the Sikulix language are used to generate tests at the graphical interface level. The proposal described in this degree work makes use of two model transformations to obtain the test scripts from user stories that will be applied in the process using SikuliX for automated visual UI (User Interface) testing. The results of the empirical evaluation of the effectiveness and user experience of the proposed solution and its tool support suggest that the UI-Test tool can benefit testers by confirming that the actions proposed in the user stories can be run on the UIs.

Keywords: UI-Test. Visual UI testing. User Stories. Test Scripts. Model-based Testing. Testing Framework.

Índice

Índice de Tablas

Índice de Figuras.

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Yo, Bryan Andrés Alba Sarango en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "Validación Basada en Pruebas de una Interfaz Gráfica de Usuario en un Entorno de Desarrollo Dirigido Por Modelos", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 20 de agosto de 2021

Bryan Andrés Alba Sarango

C.I: 1401010994

Cláusula de Propiedad Intelectual

Yo, Bryan Andrés Alba Sarango, autor del trabajo de titulación "Validación Basada en Pruebas de una Interfaz Gráfica de Usuario en un Entorno de Desarrollo Dirigido Por Modelos", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, 20 de agosto de 2021

Bryan Andrés Alba Sarango

C.I: 1401010994

DEDICATORIA

Este trabajo dedico a mi familia.

A mis padres Bety y Wigberto que con su cariño, trabajo y sacrificio han sido uno de los pilares fundamentales para cumplir con esta etapa más en mí vida.

Sin su apoyo, esto no hubiese sido posible.

A mis demás familiares, que supieron apoyarme en los momentos en los que más necesité.

AGRADECIMIENTO

Agradezco a mi familia por ser un apoyo constante en todo este proceso. Especialmente a mis padres y hermana quienes me han apoyado cada día a pesar de la distancia y cualquier circunstancia.

Agradezco a mi directora de tesis la ingeniera María Fernanda Granda por apoyarme y ayudarme en base a sus conocimientos profesionales a solucionar problemas que se pudieron presentar en el desarrollo de este trabajo. Agradezco también al ingeniero Otto Parra por haber aportado significativamente a la realización de este trabajo. Agradezco a los docentes quienes transmitieron sus amplios conocimientos y compartieron experiencias profesionales, las cuales me ayudaron a tener una visión más amplia en el ámbito laboral relacionado a la carrera.

CAPÍTULO 1: INTRODUCCIÓN

1.1 Motivación y contexto

Para reaccionar al mercado cambiante del desarrollo de software de una manera más eficiente, la adopción de prácticas de desarrollo ágil está ganando impulso (Kassab, 2015). La metodología Agile es un enfoque iterativo e incremental para el desarrollo de software, donde los requisitos y las soluciones evolucionan con el tiempo de acuerdo con las necesidades de las partes interesadas. Probar la aplicación para buscar evidencia de que la aplicación proporciona la funcionalidad solicitada por los usuarios finales o las partes interesadas ahora surge como un problema. Sin embargo, diseñar y ejecutar casos de prueba requiere mucho tiempo y es una tarea propensa a errores cuando se realiza manualmente y los cambios frecuentes en los requisitos reducen la capacidad de reutilización de estos casos de prueba escritos manualmente. Según (Latiu et al., 2013), las pruebas automáticas basadas en interfaces gráficas de usuario (GUI) pueden ser una buena alternativa porque son más precisas, confiables y eficientes.

Los métodos existentes para generar casos de prueba a partir de historias de usuarios no han sido ampliamente aceptados en la práctica, porque requieren una participación humana sustancial o porque los resultados obtenidos tienen una precisión muy baja (Lucassen et al., 2017). Además, no todos los testers tienen conocimiento previo del funcionamiento del sistema, lo que genera dificultades en el diseño de los casos de prueba para que coincidan con los requisitos.

El desarrollo dirigido por modelos ágiles (AMDD, agile model-driven development) es la versión integrada de dos metodologías para el desarrollo de software las cuales son la metodología agile y el desarrollo dirigido por modelos (Alfraihi et al., 2018)**.** Por un lado, la Ingeniería Dirigida por Modelos es un conocido paradigma de desarrollo de software que proporciona muchos beneficios para desarrollar soluciones de software adecuadas. Por otro lado, los métodos ágiles son un buen paradigma para comprender mejor los requisitos (Grangel & Campos, 2019)**.**

Por lo mencionado anteriormente este trabajo se enfoca en validar que los requisitos funcionales estén incluidos en la versión final de las interfaces de usuario del software

desarrollado generando evidencia basada en la especificación de historias de usuarios ágiles.

1.2 Planteamiento del Problema

La verificación y validación del software a través de prácticas de testing es la técnica más utilizada en el proceso de aseguramiento de la calidad (Azimian et al., 2019). Probar aplicaciones con una interfaz gráfica de usuario (GUI) es una tarea importante, aunque desafiante y que requiere de mucho tiempo (Aho et al., 2016). Las pruebas de las GUIs son inevitables cuando el único medio de interacción con un software es su GUI, las pruebas del sistema, es decir, probar el software como un todo, requiere que se pruebe con su GUI.

Para automatizar las pruebas de software antes de la era de las GUIs, los evaluadores (testers) se basaron en scripts de prueba con una colección de comandos. Las ejecuciones de los programas eran independientes de los estados de las GUIs. Sin embargo, probar los componentes de las GUI es diferente y más difícil porque requiere un script para reasignar el flujo de entrada, tal como dar clic en los botones, mover el apuntador, y presionar las teclas. Cada vez que la GUI cambia, los widgets se mueven, las ventanas se fusionan, algunos scripts se vuelven inutilizables porque ya no codifican secuencias de entrada válidas (Gao et al., 2016). Entonces los scripts necesitan tener mecanismos para registrar respuestas y cambios de los estados dinámicos del software. Luego, comparando las respuestas y cambios con los esperados, los scripts son capaces de reportar fallos o errores en la aplicación.

Actualmente existen varias herramientas especializadas disponibles para probar aplicaciones basadas en GUIs, cada una con una variedad de características (Kresse & Kruse, 2016). Los evaluadores esperan que esas características permitan realizar pruebas de GUI independientes de la plataforma, reconociendo de manera automática los componentes de la GUI tales como botones, cuadros de texto entre otros, automatizando los scripts de prueba, y verificando resultados a través de una fácil gestión. El estado de arte de la técnica de pruebas para GUI aún son herramientas de captura y reproducción, que pueden simplificar la grabación y ejecución de secuencias de entrada, pero que no son compatibles con el tester para

encontrar casos de prueba sensibles a fallas y conllevan una gran sobrecarga en el mantenimiento de los casos de prueba cuando la GUI cambia (Iyama et al., 2018). Con esta técnica, los testers son requeridos para ejecutar una interacción de intensa labor con la GUI a través del uso del ratón y secuencia de teclas, el resultado es el registro de la secuencia de eventos que luego son convertidos en pruebas automáticas.

Las pruebas basadas en modelos (MBT) han surgido como una aproximación prometedora para probar aplicaciones con GUIs (Aho, Suarez, et al., 2014; Kull, 2012). Actualmente, los modelos de pruebas son usados para modelar requisitos cercanos a la implementación de los GUI con limitadas habilidades para representar acciones abstractas (Aho et al., 2015). En este trabajo de titulación se propone usar un modelo de tareas de la aplicación y en base a este modelo generar los casos de prueba para la GUI. En nuestro estudio se discutirá formas de tomar los requisitos previos al modelo de tareas, así como generar scripts de prueba de manera semiautomática. Esos scripts tendrán funciones para probar diferentes componentes de la GUI y cuando haya cambios en estos componentes por adición o eliminación de controles en la GUI, un nuevo conjunto de pruebas puede ser generado al modificar los requisitos de la aplicación de software a validar.

1.3 Objetivos

1.3.1 Objetivo General

Diseñar e implementar una herramienta de validación basada en pruebas de una interfaz gráfica de usuario en un entorno de desarrollo dirigido por modelos.

1.3.2 Objetivos específicos

- Definir un marco conceptual relacionado para la validación de interfaces de usuario gráficas usando técnicas de prueba en un ambiente dirigido por modelos.
- Diseñar e implementar una herramienta de validación basada en modelos para interfaces de usuario gráficas integrada en un ambiente dirigido por modelos.

- Realizar una prueba de concepto de la herramienta para validar la contribución en el aseguramiento de la calidad de las interfaces gráficas.

1.4 Estructura del Trabajo

A continuación, se presenta la estructura que posee el presente trabajo y una breve descripción de cada uno de los capítulos, con el fin de describir claramente lo que se ha realizado.

- Capítulo 1. Introducción

Presenta la motivación, el contexto, la problemática actual, la solución propuesta, incluyendo el objetivo general y los objetivos específicos de este trabajo de titulación.

- Capítulo 2. Metodología

En este capítulo se especifican los métodos y técnicas aplicados sistemáticamente durante todo el proceso de desarrollo del trabajo de titulación.

- Capítulo 3. Marco Teórico

Dentro de este capítulo se describen los conceptos relacionados a este trabajo para un mejor entendimiento sobre lo que se realiza.

- Capítulo 4. Trabajo Relacionado

Se mencionan los trabajos relacionados a este tema realizando una comparación entre lo que realizan con lo que este trabajo aporta.

- Capítulo 5. UI-Test Framework: la propuesta y herramienta de soporte.

En este capítulo se describe el enfoque del trabajo en dos etapas: derivación de los casos de prueba y la prueba visual.

- Capítulo 6. Evaluación cualitativa y cuantitativa

Este capítulo describe todo el proceso para evaluar la efectividad y la experiencia de usuario. También se incluye la discusión en base a los resultados obtenidos; y, se mencionan las posibles amenazas que podrían afectar los resultados del experimento.

- Capítulo 7. Conclusiones y trabajos futuros

Este capítulo presenta las conclusiones finales del trabajo de titulación, así como los trabajos futuros que se pueden abordar a partir de los resultados obtenidos. Referencias. Lista de referencias bibliográficas tomadas para el estudio y análisis de este trabajo de titulación.

- Referencias.

Lista de referencias bibliográficas tomadas para el estudio y análisis de este trabajo de titulación.

- Anexos.

En este apartado se presentan evidencias experimentales realizadas en este trabajo, capturas, modelos de cuestionarios y evidencias de recolección de datos. Adicionalmente, se anexa el artículo académico “Towards a Model-Driven Testing Framework for GUI Test Cases Generation from User Stories” que fue enviado y aceptado en la conferencia ENASE 2021. También se anexa el artículo académico “UI-Test: A Model-based Framework for Visual UI Testing– Qualitative and Quantitative Evaluation” el cual es una versión extendida del antes mensionado mencionado y que actualmente está en revisión previo a su publicación.

CAPÍTULO 2: METODOLOGÍA

En este capítulo se describe la metodología aplicada para la elaboración de la propuesta. La metodología empleada en el estudio está basada en una metodología estructurada conforme al modelo de transferencia de Runeson & Host (Runeson & Höst, 2008), el cual plantea ocho actividades para encontrar una solución. El proceso que sigue la metodología se puede apreciar a continuación en la Fig. 1.

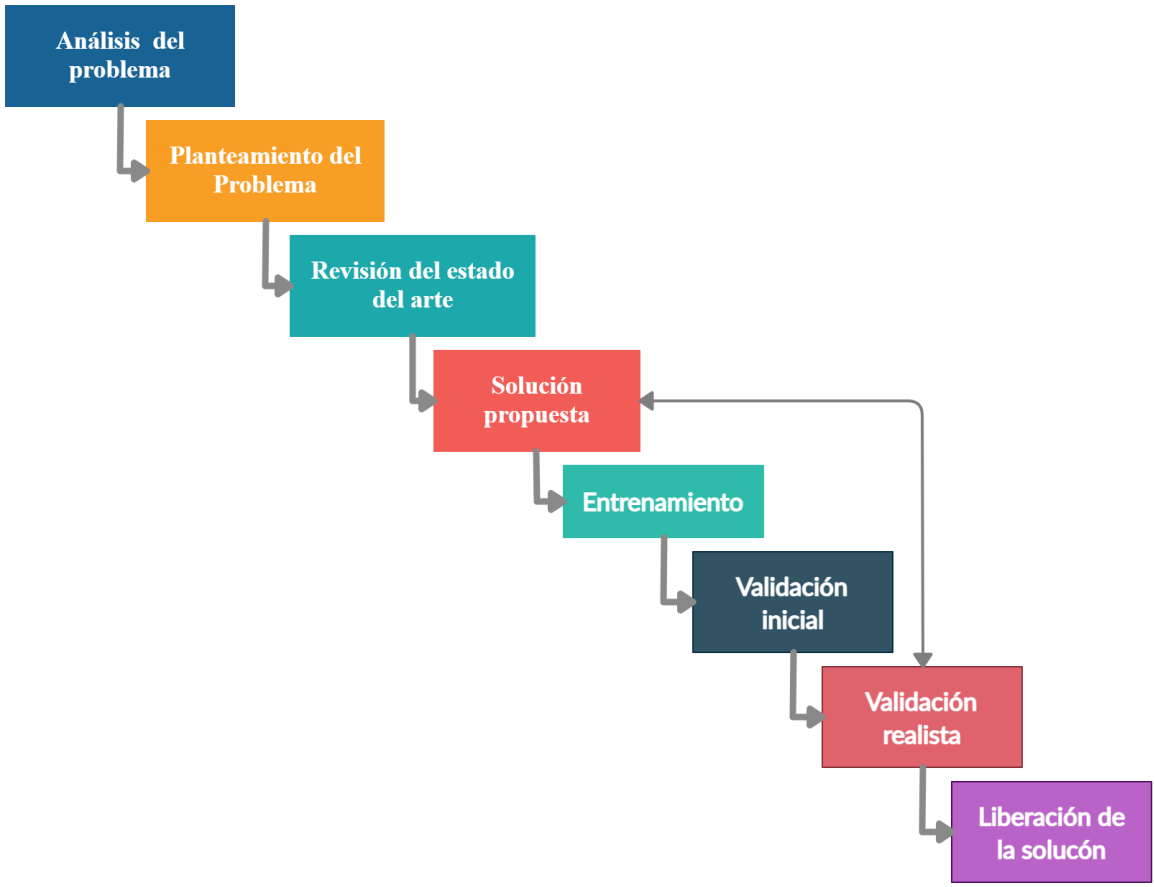


Fig. 1. Esquema general de la metodología

Las fases de la metodología se describen a continuación.

1. **Análisis del problema.** Es necesario comprender el problema que da origen y propósito a la investigación, para esto se debe observar el dominio e identificar las necesidades de las partes interesadas.
2. **Planteamiento del problema.** En esta etapa se formula el problema de forma clara y precisa, incluyendo factores de contexto, objetivos perseguidos por la investigación, planteamiento de preguntas de investigación y la justificación del estudio.

3. **Revisión del estado del arte.** Describe las tecnologías relacionadas a la validación de interfaces gráficas en un entorno de desarrollo dirigido por modelos, revisando soluciones existentes e identificando brechas que la investigación desea abordar.
4. **Solución Propuesta.** En esta etapa, se plantea la solución al problema establecido en base a la información recolectada en la revisión del estado del arte.
5. **Entrenamiento.** Esta es una actividad de tipo incremental, en la cual se busca proporcionar a los expertos del dominio el conocimiento necesario para tener una vista general acerca de la solución propuesta.
6. **Validación Inicial.** Se realiza una validación inicial de la propuesta en un entorno de laboratorio, utilizando prototipos o casos de estudio.
7. **Validación Realista.** Se realizan experimentos controlados, utilizando múltiples casos de estudio de aplicaciones reales. Se evalúa la cobertura de los casos de prueba usando aplicaciones de escritorio y web. Además, se evalúa la usabilidad y la experiencia de usuario de la herramienta UI-Test propuesta.
8. **Liberación de la Solución.** Se evalúan los resultados finales obtenidos y se preparan los requerimientos para liberar la solución y se despliega la solución en una empresa. El alcance de este trabajo no incluye esta etapa.

CAPÍTULO 3: MARCO TEÓRICO

En este capítulo se presentan todos los conceptos correspondientes a los términos principales que deben tomarse en cuenta para contextualizar el desarrollo del presente trabajo de titulación.

3.1 Desarrollo Dirigido por Modelos (MDD)

Según (Rengifo et al., 2015)**,** MDD es un paradigma emergente que resuelve varios problemas vinculados con la integración y composición de sistemas a gran escala basado en el uso de modelos, soportado por potentes herramientas que tienen como objetivo reducir el tiempo de desarrollo y mejorar la calidad de los productos, separando el diseño de la arquitectura.

Los modelos según (Rengifo et al., 2015), se van generando desde la parte más abstracta a lo más concreto, a través de transformaciones y/o refinamientos. Los puntos claves de la iniciativa del MDD según (Rengifo et al., 2015) son: La abstracción, automatización y estándares, trayendo consigo beneficios como la adaptación de los cambios tecnológicos, requisitos, re-uso y mejora la comunicación tanto para los usuarios como para los desarrolladores.

3.2 Transformación de modelos

Las transformaciones de modelo a texto (M2T) y de texto a modelo (T2M) son de gran importancia en la ingeniería basada en modelos. Las transformaciones M2T se utilizan normalmente para cerrar la brecha entre los lenguajes de modelado y los lenguajes de programación mediante la definición de generaciones de código, pero se pueden emplear de forma genérica para producir texto a partir de modelos como documentación o representaciones textuales del contenido de un modelo. Las transformaciones de T2M se utilizan normalmente para la ingeniería inversa, por ejemplo, la transformación de aplicaciones heredadas (legacy applications) en aplicaciones dirigidas por modelos, en el caso de la modernización de software impulsada por modelos. (Wimmer & Burgueño, 2013)

3.3 Interfaces gráficas de usuario (GUI)

La interfaz gráfica de usuario es la parte de un programa de computadora que maneja las entradas de la persona que usa el programa y la salida a la pantalla. Una GUI contiene objetos gráficos; cada objeto tiene un conjunto fijo de propiedades. En cualquier momento durante la ejecución de la GUI, estas propiedades tienen valores, cuyo conjunto constituye el estado de la GUI (Memon, 2007). Una GUI toma eventos (clics del mouse, selecciones, escritura en campos de texto) como entrada de los usuarios y en consecuencia cambia el estado de sus widgets(Banerjee et al., 2013).

Las GUI constituyen una gran parte del software que se desarrolla en la actualidad. Desde el punto de vista de la ingeniería de software, el GUI de un software es a menudo grande, posiblemente más de la mitad de todo el código del programa, y complejo, lo que requiere que los desarrolladores manejen gráficos elaborados, múltiples formas de dar comandos, múltiples dispositivos de entrada asíncronos y retroalimentación semántica rápida. La interacción con las computadoras ha evolucionado desde las interfaces de línea de comando (CLI) hasta la manipulación directa o las interfaces WIMP (Windows, Icon, Mouse y Pointer) y actualmente continua evolucionando con las interfaces de usuario natural (NUI: natural user interface) (Aho, Kanstrén, et al., 2014).

En cuanto a los tipos de fallas que pueden ocurrir en una IU, consideramos la clasificación presentada por Lelli et al. (Lelli et al., 2015), que considera dos grupos: 1. fallas en la interfaz de usuario, tal como la falla de la IU en un navegador web específico y, 2. fallas en la interacción del usuario como datos que los usuarios no quieren dar.

3.4 Interfaces de usuario web (WUI).

El diseño WUI es el diseño de navegación, y presentación de información donde se presta atención a los contenidos. El usuario no se mueve entre aplicaciones (como con GUI) sino de un sitio a otro. (Sakal, s. f.).

Algunos de los elementos que tiene este tipo de interfaz son: Hyperlinks, boton de recargar, menús.

3.5 Pruebas IU

Según SWEBOK (Software Engineering Body of Knowledge) (Abran, 2004) las pruebas son una actividad que se realiza para evaluar la calidad del producto y para mejorarla, mediante identificación de problemas y defectos cometidos durante la implementación de la aplicación bajo prueba (AUT) y asegurar su calidad. **La ejecución de un programa es correcta cuando su comportamiento coincide con los requerimientos funcionales y no funcionales de sus especificaciones** (Belli & Linschulte, 2008)**.**

Para las pruebas de una GUI, se realiza secuencias de eventos o interacciones con sus widgets y la corrección de su comportamiento se determina examinando sólo el estado de las widgets de la GUI (Yuan et al., 2011). La GUI es uno de los componentes de los proyectos de software que normalmente se somete a pruebas menos sistemáticas. Su comportamiento impulsado por eventos y su esencia visual consideran inadecuadas las herramientas y métodos tradicionales que comúnmente son utilizados para probar otro tipo de componentes. Sin embargo, las pruebas de GUI son importantes ya que con ellas son esenciales para la correcta ejecución del software en general (J. C. Silva et al., 2009).

Según la literatura relacionada, hay tres generaciones de pruebas de GUI automatizadas (Lelli et al., 2015): la primera generación se basa en las coordenadas de GUI pero no se utiliza en la práctica debido a los costos de mantenimiento inviables causados por la fragilidad del cambio de GUI. En cambio, las herramientas de segunda generación operan contra la arquitectura GUI del sistema, bibliotecas o interfaces de programación de aplicaciones. Si bien este enfoque se utiliza con éxito en la práctica, no verifica la apariencia de la GUI y está restringido a tecnologías específicas de GUI, lenguajes de programación y plataformas. La tercera generación, conocida como Visual GUI Testing (VGT), es una técnica emergente en la práctica industrial con propiedades que mitigan los desafíos experimentados con técnicas anteriores.

Visual UI Testing es una técnica emergente en la práctica industrial y utiliza herramientas con capacidades de reconocimiento de imágenes para interactuar con la capa de mapa de bits de un sistema, es decir, lo que se muestra al usuario en el monitor de la computadora. SikuliX, JAutomate, etc. son algunos ejemplos de herramientas que aplican este tipo de pruebas. En este trabajo de investigación se

utiliza SikuliX (Granda et al., 2021), un lenguaje de prueba estandarizado para pruebas visuales basadas en UI (VGT) en GUI y WUI. Alégroth y col. (Alegroth et al., 2015), en su estudio relacionado con los desafíos, problemas y limitaciones (CPL) de la VGT en la práctica, su principal conclusión es que la VGT es una técnica valiosa y rentable con una capacidad de detección de defectos igual o incluso mejor que las pruebas manuales.

3.6 Historias de usuario

En el ciclo de vida del desarrollo de software (SDLC: software development life cycle), la obtención de requisitos es una etapa crucial porque se definen requisitos funcionales (y no funcionales). Entrevistar a las partes interesadas es una técnica típica para obtener los requisitos. El resultado de este proceso son las historias de usuario que son una notación textual cada vez más popular para capturar los requisitos (Lucassen et al., 2016) en el desarrollo ágil de software.

El término “user stories” fue acuñado por Beck y Andres (Beck & Andres, 2004) y se refiere a la descripción de las tareas de los usuarios mediante una plantilla. La Figura 2 muestra los elementos de la plantilla, sin embargo, el último elemento (SO THAT I CAN) es opcional.

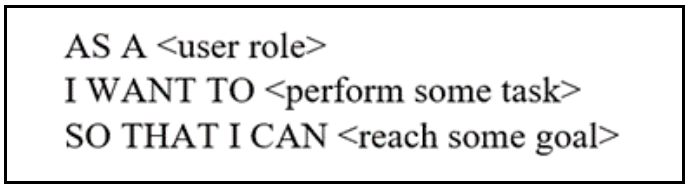


Fig. 2. Plantilla para definir una historia de usuario.

Por lo general, una historia de usuario incluye tres componentes (Figura 3): (1) una breve descripción de la historia de usuario utilizada para planificar (¿Quién?), (2) conversaciones sobre la historia de usuario para descubrir los detalles (¿Qué?) Y (3) criterios de aceptación (¿Por qué?) (Mori et al., 2002). Moreira (M. E. Moreira, 2013) describe la jerarquía de requisitos dentro de un contexto ágil incorporando algunos conceptos: temas, epics, historias de usuario y tareas (Figura 4). Dentro de los métodos ágiles, las historias de usuario se utilizan principalmente como artefactos de requisitos primarios y unidades de funcionalidad de un proyecto de software (Wautelet et al., 2014).

Component	Describes
As a [user/stakeholder]	Who?
I want to [requirement]	What?
So that [motivation]	Why?

Fig. 3. Componentes de una historia de usuario

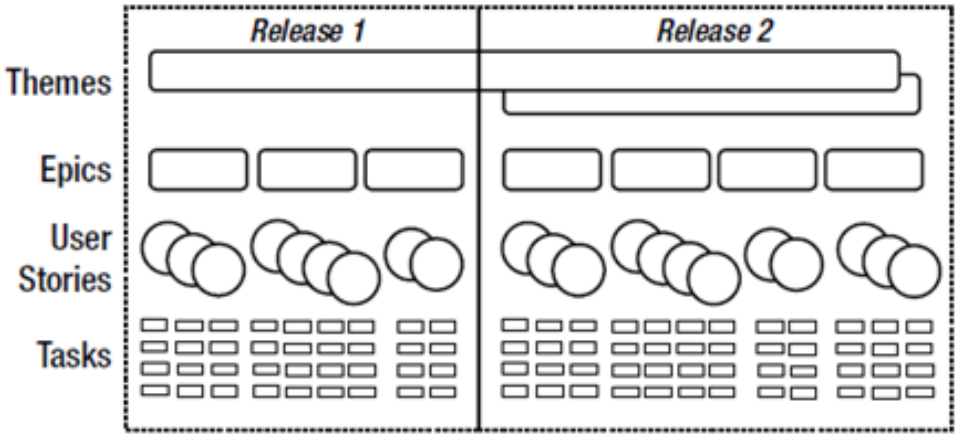


Fig. 4. Jerarquía de tipos de requisitos dentro de un Contexto Agile (tomado de Moreira 2013).

Según Moreira (M. E. Moreira, 2013), los temas son objetivos de alto nivel que pueden abarcar múltiples lanzamientos y productos. Los temas deben descomponerse en epics que se puedan aplicar a un producto o lanzamiento específico. Los epics son el padre de múltiples historias de usuarios y son aproximadamente equivalentes a una característica o una historia muy grande que encapsula una gran parte de la funcionalidad. Las tareas son hijos de las historias de usuario y equivalen a una descomposición incremental de la historia de usuario.

Los criterios de aceptación son un atributo importante de una historia de usuario. Cada historia de usuario debe tener su propio conjunto único de criterios de aceptación (M. E. Moreira, 2013). Los criterios de aceptación responden a la pregunta: "¿Cómo sabré cuando haya terminado con la historia?" Lo hacen al proporcionar información funcional y no funcional que ayuda a establecer límites para el trabajo y establece criterios de aprobación/falla para que los testers establezcan los casos de prueba que se utilizan para probar una historia de usuario.

3.7 Modelo de tarea

Un modelo de tarea es una descripción del proceso que lleva a cabo un usuario para alcanzar una meta en un dominio específico. Los modelos de tareas se encuentran entre los modelos más utilizados durante el diseño de sistemas interactivos.

Normalmente, ConcurTaskTree (CTT) (Paterno et al., 1997) se utiliza para describir de forma gráfica la secuencia de pasos para realizar una tarea. En nuestro trabajo usaremos CTT como notación de modelado de tareas. La Figura 5 muestra algunos tipos de tareas en un modelo CTT.

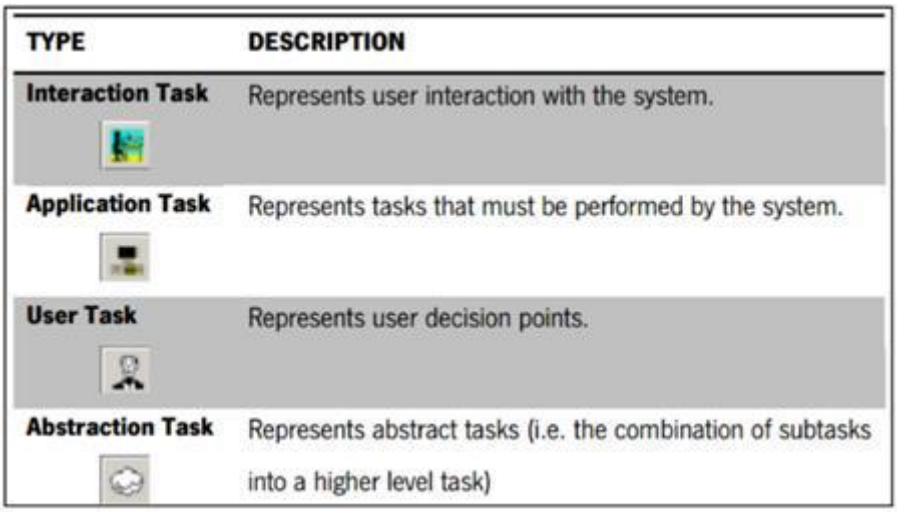


Fig. 5. Tipos de tareas en un modelo CTT

ConcurTaskTree pertenece a la familia de notaciones de análisis de tareas jerárquicas, el enfoque más común para el análisis de tareas (Mori et al., 2002). El uso de ConcurTaskTree como notación de modelado de tareas tiene algunas ventajas (Alegroth et al., 2015): (i) los modelos tienen un nivel de abstracción familiar para los diseñadores/desarrolladores de interfaces de usuario; (ii) las pruebas seguirán el uso previsto del sistema; y, (iii) el costo incurrido en el desarrollo del oráculo se reduce mucho.

Se utilizan los conceptos de temas, epics, historias de usuarios y tareas para obtener el modelo de tareas. Por ejemplo: en el contexto del uso de un editor de texto como el Bloc de notas, un tema podría ser "Managing documents in Notepad", un epic podría ser "As a user can create a document to write an essay", una historia de usuario podría ser "As a user I want to enter text in the document"; y finalmente, algunas tareas podrían ser "As a user I want to type text in the document", "As a user I want to copy text in the document" y así sucesivamente.

3.8 Lenguajes para secuencias de comandos GUI.

En la literatura relacionada sobre herramientas de software para probar los diferentes caminos en el proceso de prueba, encontramos varias alternativas, entre ellas: AutoIt 5[1], RobotFramework6[2], Squash7[3] y SikuliX[4]. Para seleccionar la herramienta a utilizar en el proceso, se hizo una comparativa de características de cada una. Los resultados de esta comparación se incluyen en la Tabla 1. De acuerdo con estos resultados, se selecciona a SikuliX para probar los diferentes caminos en la propuesta. SikuliX automatiza las pruebas de pantalla de la computadora de escritorio que ejecuta Windows, Mac o algunos Linux/Unix mediante el uso de scripts. Utiliza el reconocimiento de imágenes impulsado por OpenCV (Open Computer Vision) para identificar los componentes de la GUI. Además, SikuliX es de código abierto, no requiere ningún pago por su uso.

Tabla 1. Comparación de herramientas de software para pruebas.

Características de las Herramientas	AutoIt	RobotFramework	Squash	SikuliX
Tipo de licencia	Freeware	Open source	Comercial, requiere pago para usarlo	Open source
Plataforma soportada	Microsoft Windows	Sistemas operativos y aplicaciones independientes	Microsoft Windows	Microsoft Windows, MacOs, Linux
Tipo de aplicaciones	Aplicaciones de escritorio	Pruebas web, Swing, SWT, GUI, bases de datos.	Aplicaciones web y basadas en kubernetes	Aplicaciones web y de escritorio
Tecnología usada	Expresiones regulares	Orientado a palabras clave y datos	Código nativo JUnit, enfoque basado en palabras clave	Reconocimiento de imágenes para controlar los elementos de la GUI.
Lenguaje	Visual Basic y C#	Python y Java	Jira	Python, Java y Ruby
Método de automatización	Grabar/reproducir para automatizar	Automatización de pruebas de nivel de	Automatización basada en	Scripts de automatización del

[1] https://www.autoitscript.com/site/
[2] https://programmerclick.com/article/33341609244/
[3] https://www.squashtest.com/
[4] http://sikulix.com/

	el proceso	aceptación	plantillas	flujo de trabajo

CAPÍTULO 4: TRABAJO RELACIONADO

En este capítulo se revisan los trabajos relacionados más relevantes sobre validación de interfaces gráficas en un entorno de desarrollo dirigido por modelos, de acuerdo con varias perspectivas.

En los siguientes párrafos, se describe varios trabajos reportados por la literatura relacionada en el campo de las pruebas automatizadas:

En el contexto de la generación de casos de prueba a partir de historias de usuario ágiles, Rane (Rane, 2017) desarrolló una herramienta para derivar casos de prueba a partir de requisitos de lenguaje natural automáticamente mediante la creación de diagramas de actividad UML. Sin embargo, su trabajo requiere la descripción del escenario de prueba y el diccionario para ejecutar el proceso de generación de casos de prueba.
Los autores desarrollaron una herramienta que utiliza técnicas de NLP (procesamiento del lenguaje natural) para generar casos de prueba funcionales a partir de la descripción del escenario de prueba de forma libre automáticamente.

Elghondakly y otros. (Elghondakly et al., 2015) poponen un enfoque de prueba basado en requisitos para la generación de prueba automatizada para modelos Waterfall y Agile. Este sistema propuesto analiza los requisitos funcionales y no funcionales para generar rutas de prueba y casos de prueba. El documento propone la generación de casos de prueba a partir de historias de usuario ágiles, pero no discute ningún aspecto de implementación como las técnicas de análisis o el formato de las historias de usuario que se analizan. Esta implementación no sigue un enfoque basado en modelos.

Finsterwalder, M. (Finsterwalder, 2001), informa cómo utiliza pruebas de aceptación automatizadas para aplicaciones gráficas interactivas. Sin embargo, según el autor, es difícil automatizar las pruebas que involucran interacciones intensivas de GUI. Para

probar la aplicación en su totalidad, las pruebas deben ejercitar la GUI de la aplicación y verificar que los resultados sean correctos. En la programación extrema (XP), el cliente escribe pequeñas historias de usuario para capturar los requisitos y especifica las pruebas de aceptación. Estas pruebas se implementan y ejecutan con frecuencia durante el proceso de desarrollo.

Tao, C. y otros. (Tao et al., 2017) proponen un enfoque novedoso para las pruebas de aplicaciones móviles basado en secuencias de comandos en lenguaje natural. Se ha desarrollado un enfoque de generación de secuencias de comandos de prueba basado en Java para admitir la generación de secuencias de comandos de prueba ejecutables en función de la secuencia de comandos de operación de prueba de la aplicación móvil basada en lenguaje natural. Según los autores, no se ofrece una infraestructura de automatización unificada con las herramientas de prueba existentes. Para hacer frente a la ejecución masiva de múltiples pruebas móviles, hay una falta de métodos de secuencias de comandos de prueba móviles bien definidos, por lo que se necesita un control central de automatización de pruebas para respaldar las pruebas basadas en el comportamiento o las pruebas basadas en escenarios en múltiples niveles.

Ramler y otros. (Ramler et al., 2019), describen la introducción de pruebas basadas en modelos (MBT) para pruebas de GUI automatizadas en tres proyectos industriales de diferentes empresas. Cada uno de los proyectos ya tenía pruebas automatizadas para la GUI, pero se consideraron insuficientes para cubrir la gran cantidad de escenarios posibles en los que un usuario puede interactuar con el sistema bajo prueba (SUT). MBT se introdujo para complementar las pruebas existentes y para aumentar la cobertura con pruebas de extremo a extremo a través de la GUI.

Kamal (Kamal et al., 2019) presenta un modelo de generación de casos de prueba para construir una suite de pruebas para páginas web usando su archivo HTML. El modelo propuesto tiene dos ramas. El primero se centra en generar casos de prueba para cada elemento web individualmente en función de su tipo. La otra rama se enfoca

en generar casos de prueba basados en diferentes rutas entre elementos web en la misma página web.

Coppola y otros. (Coppola et al., 2018) proporcionan una taxonomía detallada de las modificaciones realizadas en el código de producción de las aplicaciones de Android, que pueden desencadenar la necesidad de mantenimiento de conjuntos de pruebas. Los autores creen que el problema de la fragilidad, un problema que ya se ha explorado ampliamente en el campo de las aplicaciones web, puede obstaculizar seriamente el uso a gran escala de las pruebas automatizadas para las aplicaciones de Android.

Silva y otros. (J. Silva et al., 2007) describen un esfuerzo para desarrollar herramientas de apoyo que permitan el uso de modelos de tareas como oráculos para las pruebas basadas en modelos de interfaces de usuario. El tema de interés en su investigación es el oráculo de prueba que se utilizará como medida de la calidad de la implementación.

La contribución de este trabajo de titulación es un marco de trabajo (framework) basado en modelos para aplicar pruebas de UI visuales con el objetivo de verificar si todos los requisitos de la historia de usuario de un sistema de software están incluidos en la versión final (UI) del producto de software desarrollado. Para ello se utilizan modelos de tareas, un proceso de análisis y transformaciones utilizando lenguaje Java y SikuliX.
La Tabla 2 muestra un resumen detallando las propuestas, pros y contras de los trabajos relacionados que se investigaron.

Tabla 2. Trabajos relacionados: resumen

Ref	Propuesta	Pros	Contras
(Rane, 2017)	Generación de casos de prueba a partir de historias de usuario.	Automatización mediante la creación de diagramas de actividad UML.	Requiere el diccionario y la descripción del escenario de prueba para ejecutar el proceso.
(Elghondakly	Enfoque basado en requisitos para la	Este sistema propuesto analiza los requisitos	No discute ningún aspecto de implementación como

et al., 2015)	generación de pruebas automatizadas para modelos ágiles y en cascada. Generación de casos de prueba a partir de historias de usuarios.	funcionales y no funcionales para generar rutas de prueba y casos de prueba.	las técnicas de análisis o el formato de las historias de usuario que se analizan. No sigue un enfoque basado en modelos.
(Finsterwalder, 2001)	Pruebas de aceptación automatizadas para aplicaciones de gráficos interactivos.	Captura de requisitos a través de historias de usuarios. Se especifican las pruebas de aceptación. El cliente gana confianza en la funcionalidad de la aplicación desarrollada.	Para probar completamente la aplicación, las pruebas deben ejercitar la GUI de la aplicación y verificar que los resultados sean correctos.
(Tao et al., 2017)	Pruebas de aplicaciones móviles basadas en secuencias de comandos en lenguaje natural.	Pruebas ejecutables basadas en el script de operación de prueba de la aplicación móvil basada en lenguaje natural.	Existe una falta de un método de secuencia de comandos de prueba móvil bien definido para hacer frente a la ejecución masiva de múltiples pruebas móviles.
(Ramler et al., 2019)	Modelos basados en pruebas (MBT) para GUI automatizadas.	MBT se introdujo para complementar las pruebas existentes y para aumentar la cobertura con pruebas de un extremo a otro a través de la GUI.	Las pruebas automatizadas se consideran insuficientes para cubrir la gran cantidad de escenarios posibles.
(Coppola et al., 2018)	Verificar el estado de las pruebas de GUI basadas en widgets de Android considerando la fragilidad de los conjuntos de pruebas.	Los autores tienen como objetivo validar y ampliar la taxonomía mediante la aplicación de procedimientos más rigurosos basados en la teoría fundamentada.	Las pruebas de GUI automatizadas tienen un alto costo en términos de implementación y mantenimiento, y la rápida evolución de las GUI de Android puede exacerbar esos costos para los desarrolladores.
(J. Silva et al., 2007)	Desarrollar soporte de herramientas que permita el uso de modelos de tareas como oráculos para pruebas basadas en modelos de interfaces de usuario.	Han demostrado en su trabajo de investigación que los modelos de tareas, a pesar de las limitaciones, pueden usarse para generar oráculos de manera económica en un contexto de prueba basado en modelos de sistemas interactivos.	Han encontrado limitaciones para generar los oráculos de prueba directamente a partir de modelos de tareas.

CAPÍTULO 5: UI-Test Framework: la propuesta y una herramienta de soporte

En este capítulo se abordará, el enfoque propuesto en dos etapas: derivación de los casos de prueba y la prueba visual. Para demostrar y discutir la idoneidad del enfoque, se lo aplica en la aplicación Bloc de notas de Microsoft. Esta aplicación fue seleccionada porque es una aplicación común y conocida por los lectores, lo que facilita la explicación del enfoque. Se implememta una herramienta de soporte utilizando la plataforma Eclipse[5] y el lenguaje de programación Java. Esta herramienta de soporte se puede utilizar en tres plataformas: Microsoft Windows, Linux, MacOS ya que la máquina virtual Java[6] está disponible en cada plataforma.

5.1 Derivación de casos de prueba

En esta primera etapa de derivación de los casos de prueba, se define seis pasos (Figura 6):

Paso 1. Se necesita la especificación de requisitos obtenida de las partes interesadas mediante algún método (por ejemplo, especificación de requisitos usando lenguaje natural, usando análisis de casos, usando el Taller de atributos de calidad - QAW, o análisis global) (Bass et al., 2006). El modelador/ingeniero de requisitos luego escribe las historias de usuario en función de los requisitos. En este caso, se consideran solo los requisitos funcionales porque este tipo de requisito involucra principalmente acciones en la interfaz de usuario.

[5] https://www.eclipse.org/
[6] https://www.java.com/

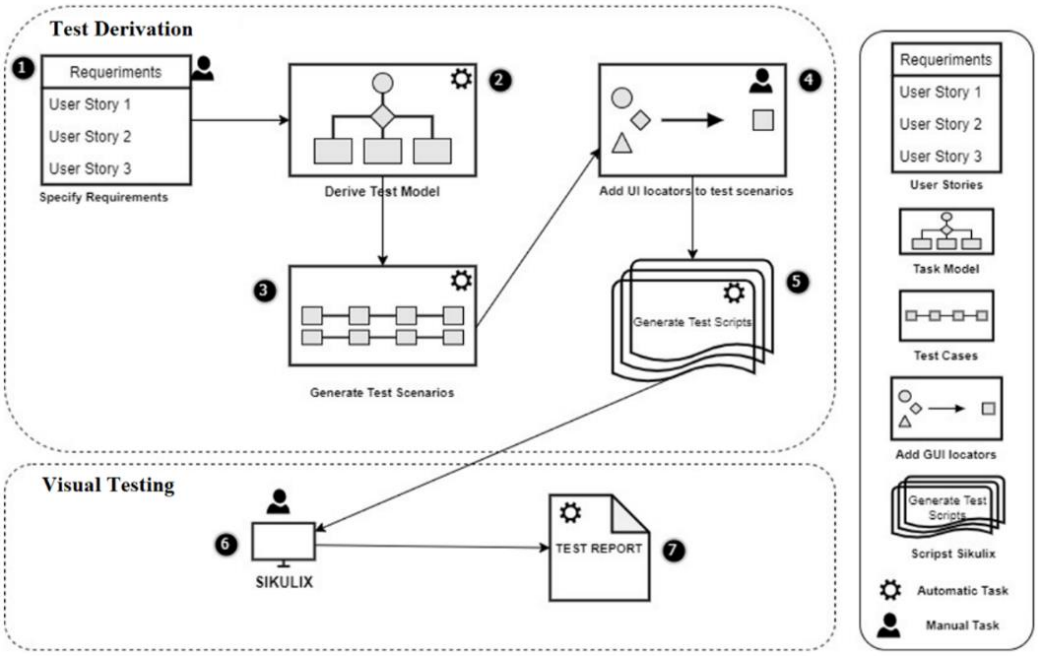


Fig. 6. El enfoque propuesto

En este trabajo solo se usan dos aspectos de una historia de usuario: (1) quién quiere la función; y (2) qué funcionalidad los usuarios finales o las partes interesadas quieren que proporcione el sistema. Las historias de usuario se escriben de acuerdo con la plantilla (Fig. 3) y luego se almacenan en un archivo de texto. En la figura 7 se muestra un extracto de las historias de usuario definidas para la aplicación Bloc de notas.

En esta versión de la herramienta de soporte, las historias de usuario se incluyen editando directamente en el editor de la herramienta o usando "copiar y pegar". En este contexto, se verifican todas las historias de usuario para confirmar que cada historia de usuario está escrita de acuerdo con la plantilla antes mencionada (Fig. 8).

Paso 2. En el segundo paso, se usan las historias de usuario previamente insertadas en el soporte de la herramienta como base para la derivación del modelo de prueba (es decir, modelo de tarea). Como se mencionó en el apartado anterior, se utiliza CTT para representar el modelo de tareas mediante un archivo XML para describir las tareas incluidas en el modelo siguiendo la sintaxis definida en CTT. La figura 9 muestra un extracto de un archivo CTT que describe una tarea.

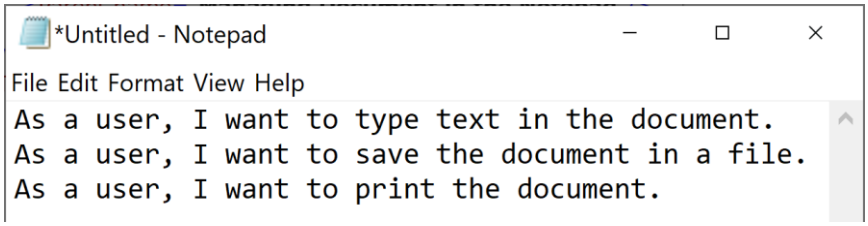


Fig. 7. Un extracto de las historias de usuario definidas en el Bloc de notas

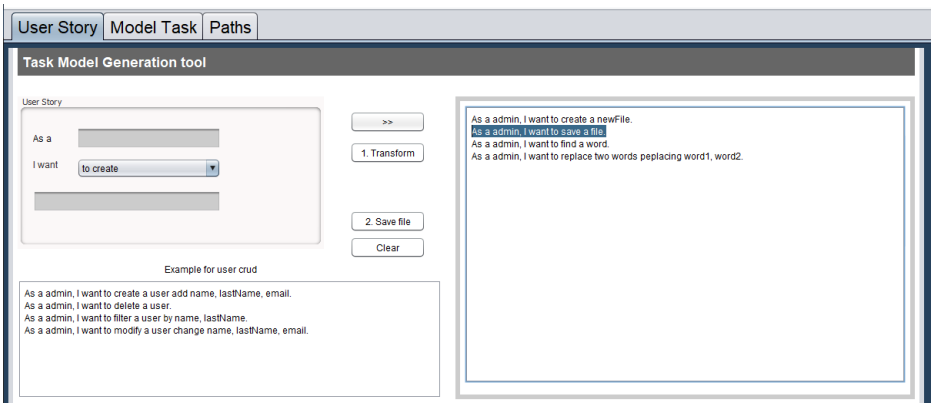


Fig. 8. Incluyendo las historias de usuario en UI-Test

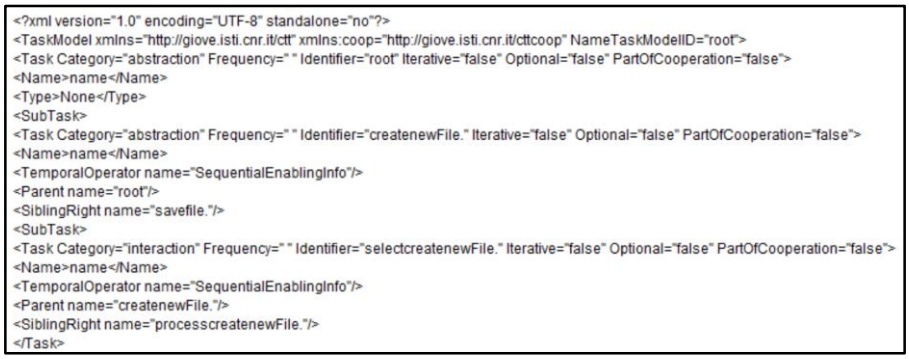


Fig. 9. Extracto de un árbol CTT que describe una tarea.

Se deriva un modelo de prueba basado en las tareas del CTT, y mediante un análisis de las rutas en este modelo se generan los escenarios de prueba con casos de prueba abstractos. Este proceso es iterativo con cada historia de usuario. Este proceso de derivación se basa en las relaciones que se establecen entre la especificación de las historias de usuario y la sintaxis del modelo de tareas. Es posible asociar los conceptos de la historia del usuario con la sintaxis del modelo de tarea y algunas de las palabras clave que usan las aplicaciones que se han utilizado para evaluar la herramienta (por ejemplo, Bloc de notas, Calculadora, etc.). Estas palabras clave están asociadas con el menú principal y sus opciones en una aplicación (por ejemplo, archivo, edición, formato, etc.) y estas palabras clave describen los pasos necesarios

para realizar una acción. Por ejemplo, la secuencia de comandos "Formato" y "Fuente" permite cambios en la fuente del texto, el estilo de fuente y el tamaño del texto en un documento usando un editor de texto.

El resultado obtenido en este paso es un árbol que contiene la información de cada nodo del modelo de tareas. Cada nodo del árbol está definido por tres campos: (a) la tarea a realizar ("Guardar un documento"); (b) si cada nodo tiene hijos, la referencia a cada hijo; (c) la relación con otros nodos del árbol. Las relaciones se crean de forma predeterminada como entrelazado (|||), ya que las tareas se pueden realizar en cualquier orden. Sin embargo, el tester puede cambiarlos editando el modelo CTT.

Paso 3. En este paso se obtienen escenarios de prueba, los cuales se basan en la definición de diferentes caminos obtenidos como resultado de aplicar dos operaciones básicas en el árbol CTT: enumerar (recorrer el árbol) y buscar (encontrar un nodo específico)

Para generar escenarios de prueba, necesitamos recorrer el árbol. En el ejemplo (Figura 10) en el Bloc de notas, se puede obtener un primer escenario cuando el árbol se recorre desde el nodo raíz (Documento de gestión en el Bloc de notas), y luego se visita el nodo izquierdo (Documento abierto). Se pueden obtener otros escenarios de prueba cuando se comienza en el nodo raíz y luego se visita el nodo central (Editar documento). Considerando este último nodo como la raíz del subárbol, el siguiente nodo a visitar es "Guardar documento", el siguiente es "Nombre de archivo:" y el último nodo es "Guardar". El escenario de prueba generado en UI-Test de “Guardar Documento” se muestra en la Figura 11. En esta travesía, se debe considerar la relación entre los nodos para definir el próximo nodo a visitar. La relación se muestra mediante las operaciones temporales definidas en ConcurTaskTree (Brüning & Forbrig, 2011).

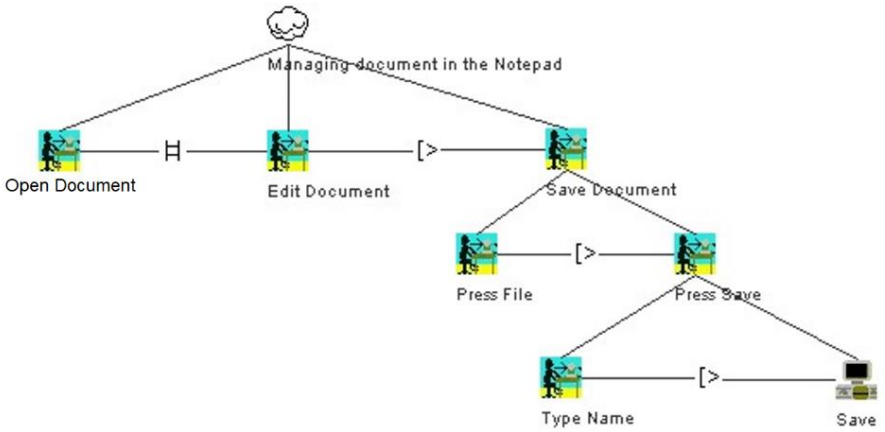


Fig. 10. Un extracto del árbol CTT que describe una tarea del ejemplo del Bloc de notas.

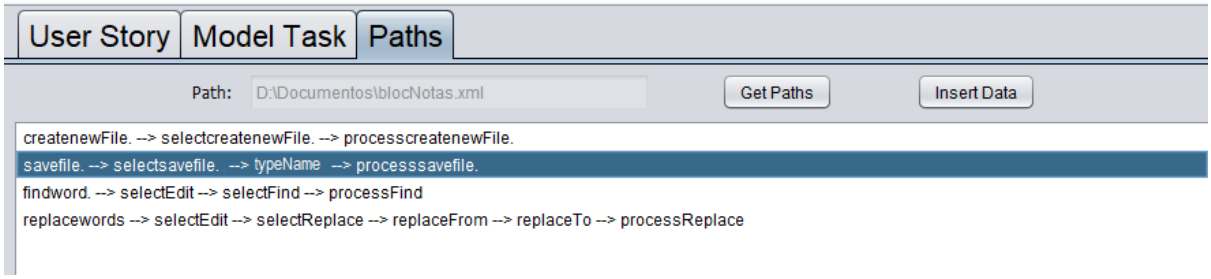


Fig. 11. Escenarios de prueba

Paso 4. Ahora, es necesario completar los scripts de prueba generados en el paso anterior utilizando localizadores para seleccionar los widgets de destino. En este caso, los localizadores representan (i) ruta a un archivo de imagen o (ii) simplemente texto sin formato (p. Ej., Nombre de archivo en la figura 12, lado derecho), que se puede utilizar como parámetro del widget (Imagen del elemento de la interfaz de usuario).

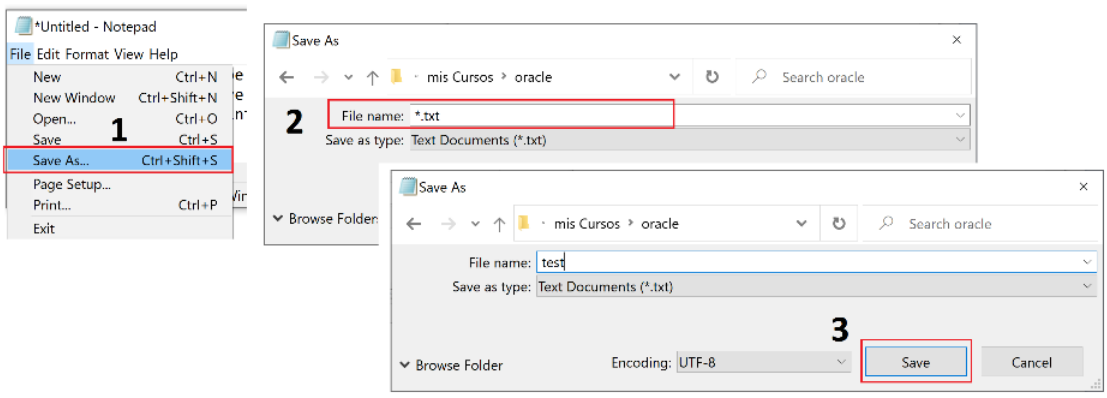


Fig. 12. Secuencia de pasos en el Bloc de notas para guardar un documento

En este trabajo se incluyen widgets para ingresar datos o para realizar una acción, excepto widgets como la barra de desplazamiento y los controles deslizantes. La Tabla 3 ofrece algunos ejemplos de widgets utilizados en aplicaciones de Windows/Web.

Tabla 3. Ejemplos de widgets

GUI widgets	Action	WUI widgets	Action
Save	Guardar un archivo / imagen	☰	Menú
test	Entrada de texto	↻	Recargar página web
☐ Unpublished work · Mendeley Web cat	Checkbox	Copy link	Hyperlink/button

Una aplicación normalmente contiene algunos elementos (widgets) como botones, campos de texto, enlaces, etc. en su interfaz de usuario que permiten al usuario realizar alguna acción previamente definida en la aplicación (un tipo de interacción entre el usuario y la aplicación). Cada uno de estos elementos tiene un localizador específico, que lo identifica de los demás elementos de la interfaz de usuario. Durante las pruebas de IU visual, estos elementos se utilizan para ubicar una determinada posición definida por el caso de prueba. Para automatizar la generación y ejecución del script de prueba, es necesario identificar estos localizadores para incluir los elementos respectivos de la interfaz de usuario. Adicionalmente, el tester debe ingresar el valor de las variables requeridas (p. Ej., nombre de archivo de un archivo, texto para buscar, etc.) utilizando el soporte de herramientas (Figura 13).

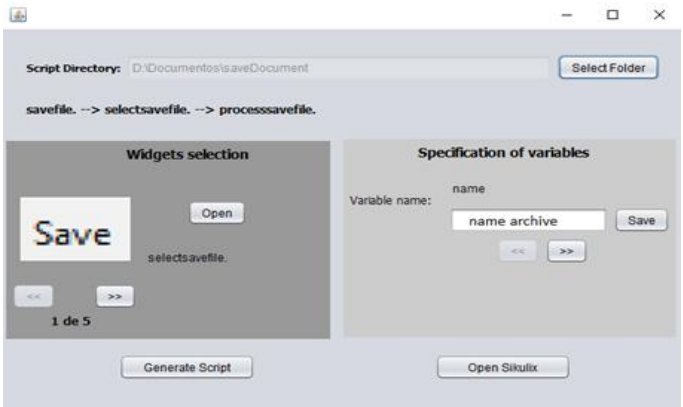


Fig. 13. Interfaz para especificar variables en la herramienta

Paso 5. En este paso se generan los scripts de prueba. El proceso de generación se basa en las relaciones establecidas entre la especificación de la historia del usuario (ver la columna 1 en la tabla 4), los widgets de IU (ver la columna 2 en la tabla 4) y el código SikuliX (ver la columna 3 en la tabla 4). Es posible asociar los conceptos de IU con la sintaxis del marco UI-Test y algunas de las palabras clave disponibles en los menús y la interfaz de usuario de la aplicación Bloc de notas (ver Tabla 4).

Tabla 4. Elementos y funciones de SikuliX para guardar un archivo en el Bloc de notas (adaptado de (Granda et al., 2021))

Tipo de tarea de la historia de usuario	Elemento IU	Código Generado
Correr la aplicación.		Screen s = new Screen();
Seleccionar/ordenar/filtrar una opción en la aplicación.	Button Element	s.click($locator);
Editar un campo de texto.	Campo de texto	s.type($locator, “text”);

5.2 Pruebas de IU Visual

Paso 6. En la figura 14 se muestran los elementos y funciones de SikuliX y el código para aplicar las pruebas en una aplicación (p. Ej., el Bloc de notas). Se puede usar el editor de la plataforma Eclipse o cualquier editor de texto para revisar/editar este código fuente.

En primer lugar, SikuliX encuentra un botón de campo de texto y escribe "Bloc de notas" y hace clic en "Entrar". A continuación, se inicia el proceso para realizar una acción en la aplicación. En el extracto del código fuente que se muestra en la figura 14, se puede ver cómo se utilizan los elementos incluidos en la Tabla 4. Cuando la aplicación se está ejecutando y el usuario selecciona la opción "Archivo" en el menú del Bloc de notas, se selecciona la opción "Guardar". El script identifica un campo de texto en el que el usuario escribe el nombre del archivo. El último paso es hacer clic en el botón "Guardar". En la misma figura, también se puede ver la frase “s.wait (1.0)” que equivale al período de tiempo de espera para cargar la aplicación (por ejemplo,

el Bloc de notas) y que su interfaz está activa y es capaz de ejecutar las pruebas en su elementos.

```
5   public class Notepad {
6       public static void main(String[] args) {
7           Screen s = new Screen();
8           s.click("file.png");// click file button
9           s.click("save.png");// click save button
10          s.find("textFiel.png"); // identify textFiel
11          s.wait(1.0);// wait for 1 second to show results
12          s.type("nameFile"); // write name file
13          s.click("saveButton.png");
14      }
15  }
```

Fig. 14. Extracto del código fuente para aplicar pruebas basadas en UI para el Bloc de notas.

Paso 7. Una vez que la secuencia de comandos está completa, se ejecutan las pruebas y se muestran los resultados de la prueba. En esta prueba seleccionamos la opción "Archivo", luego se da clic en la opción "Guardar" para guardar un archivo en el Bloc de notas que se ejecuta en Microsoft Windows 10. Se necesita escribir el nombre del archivo y luego hacer clic en el botón "Guardar" para guardar el archivo en el disco duro de la computadora. Al final, la prueba arrojó un resultado como se esperaba y, por lo tanto, el conjunto de pruebas generado no pudo encontrar ninguna falla (Fig. 15).

Cuando no se puede encontrar el localizador de IU (imagen/texto), SikuliX detendrá el script generando una excepción FindFailed, de modo que la prueba pudo haber encontrado una falla (Fig. 16).

Hay varias razones por las que se activará la excepción, por ejemplo: un localizador de IU de la interfaz está deshabilitado, el evaluador no asignó correctamente la imagen de un localizador de IU incluido en la interfaz, el valor de una variable no se ha asignado, entre otros. En todos estos casos, el tester debe analizar los resultados para verificar (es decir, detectar inconsistencias o problemas en las prueba) y validar (es decir, asegurarse de que los requisitos del cliente se pueden encontrar en la UI) la especificación de requisitos.

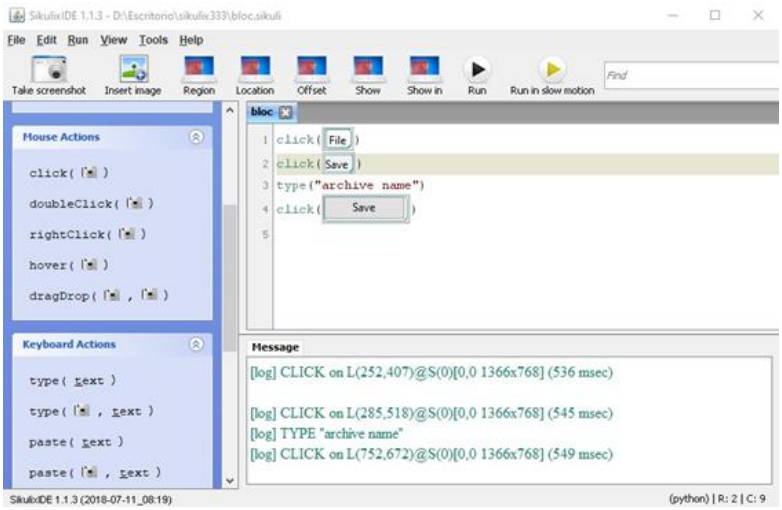


Fig. 15. Script de texto escrito en SikuliX para ejecutar la opción “Guardar” del Bloc de notas.

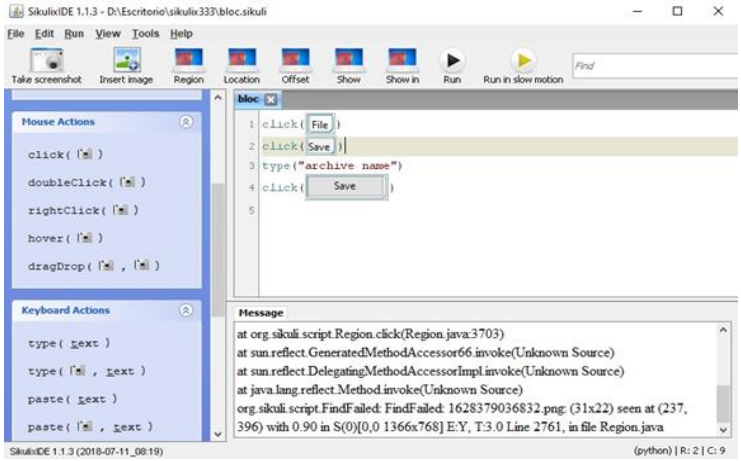


Fig. 16. Excepción FindFailed cuando la prueba falló en SikuliX

CAPÍTULO 6: EVALUACIÓN CUANTITATIVA Y CUALITATIVA

Dado que la evaluación fue motivada por la necesidad de evaluar la efectividad del marco de trabajo propuesto para las pruebas visuales de IU, se realizan dos evaluaciones, una cualitativa y otra cuantitativa. Esta primera evaluación fue un cuasi-experimento realizado en junio de 2021. Este capítulo describe los objetivos de los estudios, las preguntas de investigación, las variables y métricas utilizadas, las aplicaciones analizadas y los escenarios experimentales de la evaluación cuantitativa y cualitativa.

6.1 Evaluación Cuantitativa

6.1.1 Objetivo

El enfoque que se ha utilizado en el experimento es la metodología Objetivo, Pregunta, Métrica (GQM por sus siglas en inglés). Usando la plantilla de objetivos de GQM, el objetivo del estudio es analizar la efectividad de los casos de prueba generados por la herramienta UI-Test con el fin de realizar una evaluación con respecto a su cobertura de ejecución desde el punto de vista de los testers en el contexto del tipo de IU (GUI y WUI) para diez aplicaciones reales.

6.1.2 Preguntas de investigación

La siguiente pregunta de investigación se derivó de nuestro objetivo:

PI1: ¿Cómo influye el tipo de IU en la eficacia de los casos de prueba generados por la herramienta UI-Test cuando se prueban historias de usuario a nivel visual?

6.1.3 Hipótesis

Se definió la siguiente hipótesis. La hipótesis nula (representada por el subíndice 0) corresponde a la ausencia de impacto de las variables independientes sobre las variables dependientes. La hipótesis alternativa muestra la existencia de un impacto y es el resultado esperado.

- **H_{10}:** El tipo de interfaz de usuario no influye en la eficacia de los casos de prueba generados por la herramienta UI-Test para validar historias de usuario a nivel visual.

6.1.4 Variables y Métricas

En este estudio se identificaron las siguientes variables:

6.1.4.1 Variables Independientes

- **Tipo de IU seleccionado.** Consideramos dos tipos de IU:
  - Interfaz de usuario basada en Windows (GUI): solo se puede acceder a estas aplicaciones desde el sistema en el que está instalada y tiene una interfaz gráfica de usuario.
  - Interfaz de usuario basada en web (WUI): una aplicación web a la que se puede acceder desde cualquier sistema a través del navegador de Internet.

6.1.4.2 Variables Dependientes y Métricas

Se consideró la siguiente variable dependiente y se espera que sea influenciada en cierta medida por la variable independiente. En este estudio, la efectividad de los casos de prueba generados por la herramienta UI-Test al validar historias de usuario en la interfaz de usuario de la aplicación se mide mediante:

- **Porcentaje de cobertura de ejecución de prueba (TEC por sus siglas en inglés).** Esto también se denomina pruebas ejecutadas y es el porcentaje de pruebas aprobadas/ejecutadas del número total de pruebas. La ventaja de esta métrica es que ofrece una descripción general del progreso de las pruebas contando el número de pruebas aprobadas y fallidas.

$$TEC = \frac{number\ of\ test\ cases\ run}{Total\ number\ of\ tests\ to\ be\ run}\ x\ 100\% \quad (1)$$

6.1.5 Aplicaciones analizadas

Se utilizaron diez aplicaciones populares para este primer estudio, que contiene una variedad de widgets de entrada que se pueden probar en el nivel de la interfaz de usuario, incluidos botones, campos de texto, iconos, menús, casillas de verificación e hipervínculos. Hemos seleccionado 5 aplicaciones con GUI y 5 con WUI (consulte la Tabla 5).

Tabla 5. Tipos de widgets de las aplicaciones seleccionadas.

Aplicación	UI type	Button	Text Field	Icon	Menú	Radio Button	Check Box	Hyper Link
App1: Eclipse es una plataforma de código abierto con un conjunto de herramientas de programación.	GUI	✓	✓	✓	✓			✓
App2: Microsoft Word es una suite ofimática.	GUI	✓	✓	✓	✓	✓	✓	✓
App3: Microsoft Excel es una hoja de cálculo	GUI	✓	✓	✓	✓	✓	✓	✓
App4: Windows Calendar es una aplicación de calendario personal.	GUI	✓	✓	✓	✓			✓
App5: Windows Mail es un cliente de correo electrónico y grupos de noticias	GUI	✓	✓	✓	✓		✓	✓
App6: Youtube es un sitio web dedicado a compartir videos.	WUI	✓	✓	✓	✓			✓
App7: Google Translator es un servicio gratuito que traduce palabras, frases y páginas web a más de 100 idiomas.	WUI	✓	✓	✓	✓			✓
App8: Netflix es un servicio de transmisión.	WUI	✓	✓	✓	✓			✓
App9: Facebook es una red social.	WUI	✓	✓	✓	✓			✓
App10: Instagram permite a los usuarios editar y cargar fotos y videos.	WUI	✓	✓	✓	✓			✓

6.1.6 Procedimiento Experimental

Esta sección describe los detalles de la configuración experimental, incluidas las aplicaciones utilizadas, la instrumentación, la recopilación de datos y el análisis. La figura 17 resume el proceso experimental, que involucró los siguientes ocho pasos:

1. **Elegir las aplicaciones.** Las aplicaciones seleccionadas se describen en la Sección 6.1.5. Estas aplicaciones son de diferentes tamaños y dominios (por ejemplo, una aplicación de correo electrónico, red social, servicio de traducción, etc.). Las aplicaciones son casos reales y se seleccionaron porque contenían los widgets de IU necesarios para usar la herramienta de prueba de IU.

2. **Escribir las historias de los usuarios.** Para esta fase se hizo un proceso inverso al que se debería hacer normalmente. Se seleccionan aplicaciones reales ya implementadas y se escriben historias de usuarios que describen cómo funcionan. Luego, se seleccionaron al azar 187 historias de usuario para nuestras 10 aplicaciones seleccionadas. Para cada historia de usuario, se realizó un análisis de sintaxis utilizando el analizador UI-Test para garantizar que la historia de usuario fuera válida y pudiera usarse en el proceso de prueba. Se remite al lector a la URL (**https://xurl.es/5un0l**) para obtener información más detallada.

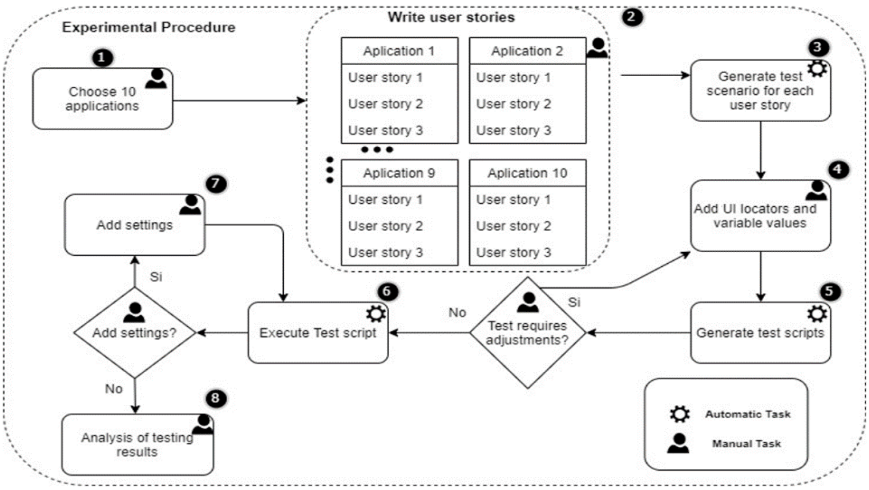


Fig. 17. Procedimiento experimental para medir la eficacia del UI-Test utilizando 10 IUs

3. **Generar escenarios de prueba.** Cada historia de usuario se transforma en un script de prueba ejecutable utilizando el módulo UI-Test respectivo (consulte el Paso 1 en la Sección 5.1).
4. **Agregar localizadores de IU y valores de variables para probar scripts.** Los valores se especificaron manualmente para cada escenario de prueba.
5. **Ejecutar scripts de prueba en aplicaciones y recopilar datos.** Cada script de prueba se ejecutó utilizando la herramienta SikuliX. El estado de los scripts de prueba (es decir, pasa/falla/no es ejecutable) se registró manualmente.
6. **Análisis de los resultados de las pruebas.** Finalmente, el valor de la cobertura de ejecución de los scripts de prueba UI-Test se calcula a partir de la información registrada en la etapa anterior. Estos resultados se dan en la siguiente sección.

6.1.7 Análisis e Interpretación de Resultados

Esta sección describe el análisis e interpretación de los resultados relacionados con nuestra variable de respuesta para PI1. El análisis estadístico se realizó sobre el paquete estadístico de ciencias sociales (SPSS).

Dado que la primera pregunta de investigación (PI1) tenía como objetivo evaluar la cobertura de ejecución de las pruebas UI-Test, se compara la cantidad de pruebas que se podrían ejecutar por tipo de IU (es decir, GUI y WUI) en las diferentes aplicaciones seleccionadas. La Tabla 6 muestra el número de historias de usuario utilizadas con la herramienta (columna 2), y la tasa y porcentaje de cobertura de ejecución (columnas 3 y 4) de las pruebas generadas para cada aplicación.

Se realizaron pruebas de Shapiro-Wilk para evaluar la normalidad de las muestras. Usamos esta prueba porque es más apropiada para tamaños de muestra pequeños (<50 muestras).

Tabla 6. Resultados de la cobertura del script de prueba generado por UI-Test

Aplicación	Historias de usuario seleccionadas	Ejecución de Cobertura Rate	Porcentaje
App 1	13	0.8	80%
App 2	12	0.6	60%
App 3	12	0.6	60%
App 4	19	0.8	80%
App 5	24	0.9	90%
App 6	24	0.8	80%
App 7	24	0.6	60%
App 8	11	0.6	60%
App 9	24	0.8	80%
App 10	24	0.4	40%
Total	**187**		

Dado que los valores Sig. para las pruebas de Shapiro-Wilk fueron 0.201 para GUI y 0.314 para WUI (ver Tabla 7). Ya que estas variables siguen una distribución normal (> 0,05), y se consideran ambos tipos de IU como grupos independientes. Se utilizó la prueba T-Student para probar la primera hipótesis nula (H10). La Fig. 18 muestra el diagrama de caja que contiene la cobertura de ejecución por tipo de IU y la Tabla 8 muestra la desviación estándar y media.

Tabla 7. Pruebas de normalidad para la métrica de cobertura de ejecución.

	UI Type	Shapiro-Wilk		
		Statics	df	Sig.
Coverage	GUI	0.852	5	0.201
	WUI	0.881	5	0.314

La Tabla 9 muestra los resultados de la prueba T-student. No se encuentra una diferencia estadísticamente significativa entre la cobertura de ejecución de las GUIs (M=0,74; DE=0,13416) y las WUIs (M=0,64; DE=0,16733) t(8)=1,043, p=0,328, d=0,000659 y por tanto se aceptó la hipótesis nula H10. En otras palabras, con un 95% de confianza se puede decir que el valor mediano de la tasa de cobertura de ejecución es similar para ambos tipos de IU; p=0,728>0,05.

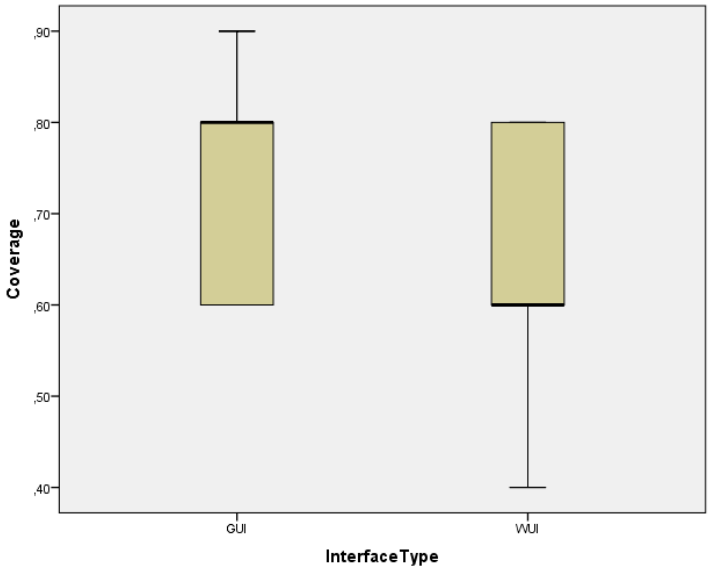


Fig. 18. Diagrama de caja para la cobertura de ejecución por tipo de interfaz de usuario

Tabla 8. Estadísticas de grupo para la cobertura de ejecución

	UI Type	N	Mean	Std. Derivation	Std Error Mean
Execution	GUI	5	0.7400	0.13416	0.6000
Coverage	WUI	5	0.6400	0.16733	0.7483

Tabla 9. Resumen de la prueba de hipótesis H10

		Prueba de muestras independientes								
		Prueba de Levene para la igualdad de varianzas		Prueba t para la igualdad de la media						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference Lower	Upper
Cobertura de ejecución	Igualdad de varianzas asumidas	0.130	0.728	1.043	8	0.328	0.1000	0.09592	-0.12118	0.32118
	Igualdad de varianzas no asumidas			1.043	7.639	0.329	0.100	0.9592	-0.12302	0.32302

6.2 Evaluación Cualitativa

Debido a la pandemia de COVID-19, este estudio se realizó completamente en línea. En esta sección, se presenta el diseño utilizado para ejecutar un cuasi-experimento.

6.2.1 Objetivo

El objetivo del estudio fue analizar el conjunto de pruebas generado por la herramienta UI-Test con el fin de evaluar la experiencia del usuario percibida de la herramienta desde el punto de vista de la academia en el contexto de las actividades de prueba de IU.

6.2.2 Preguntas de Investigación

Del objetivo anterior, se derivó la siguiente pregunta de investigación:

- **PI2:** ¿Cómo se vió afectada la experiencia del usuario durante las tareas de prueba en las IUs?

6.2.3 Hipótesis

La siguiente hipótesis fue definida.

- **H_{20}**: El tipo de IU no influye en la experiencia del usuario de los participantes

6.2.4 Variables y Métricas

En este estudio se identificaron las siguientes variables:

6.2.4.1 Variable Independiente

- **Tipo de GUI seleccionado.** Al igual que en el estudio cuantitativo, la variable independiente fue el tipo de interfaz de la aplicación (i) GUI y (ii) WUI.

6.2.4.2 Variables y métricas dependientes

Se consideró la siguiente variable dependiente y se esperaba que estuviera influenciada en cierta medida por la variable independiente.

- **Experiencia de usuario.** Para esta variable se utilizó la versión corta del cuestionario (UEQ-S) (Schrepp et al., 2017). El cuestionario aborda dos aspectos en general, los cuales son aspectos pragmáticos de calidad, es decir, describen cualidades de interacción que están relacionadas con las tareas u objetivos que el usuario pretende alcanzar al utilizar el producto software (en nuestro caso el UI-test), y aspectos de calidad hedónico, es decir, no están relacionados con tareas y objetivos, sino que describen aspectos relacionados con el placer o la satisfacción en el uso del producto.

6.2.5 Participantes

El experimento involucró a 17 sujetos voluntarios (14 hombres y 3 mujeres), que aceptaron una invitación para participar en el estudio. Los participantes tenían edades comprendidas entre los 21 y los 28 años. El conocimiento y la experiencia previos en la obtención de requisitos y las pruebas de software eran requisitos estrictos, por lo que esta competencia se solicitó durante el registro para participar en el evento.

6.2.6 Contexto de estudio

El estudio se llevó a cabo como parte de un curso en línea. Se envió una invitación a estudiantes de pregrado en Ciencias de la Computación de la Universidad de Cuenca (Ecuador). La duración del curso fue de una hora. Aunque el curso tuvo 26 asistentes, solo 17 participaron en el estudio (S1-S17 en la Tabla 10).

6.2.7 Diseño Experimental

Se seleccionó un diseño de estudio entre sujetos (o entre grupos) para el experimento, en el que diferentes personas prueban cada condición, de modo que cada persona solo esté expuesta a un único tipo de IU.

6.2.8 Instrumentación

Cuestionario: Se implementó una encuesta basada en la web utilizando Formularios de Google, que se componía de tres conjuntos de preguntas sobre:

- **Datos demográficos:** se pregunta el género, rango de edad, antecedentes educativos y experiencia en el dominio del experimento (elicitación de requisitos y pruebas de software).

- **Experiencia de usuario (EU):** la versión abreviada del cuestionario se compone de 8 pares de propiedades opuestas que puede tener el producto (es decir, la prueba de IU) (Schrepp et al., 2017). La UEQ considera aspectos de calidad pragmática y hedónica. Para cada uno de los adjetivos se utilizó una escala Likert de 7 puntos para medir los aspectos de calidad, considerando que los valores 1, 2 y 3 corresponden al adjetivo de la izquierda, y los valores 5, 6 y 7 al adjetivo de la derecha, 4 significa un valor neutral.

- **Comentarios sobre la herramienta**, un cuestionario que incluye una pregunta abierta sobre cómo se podría mejorar la herramienta.

Tareas de prueba de la IU: Los participantes tuvieron que derivar y ejecutar los scripts de prueba en dos interfaces de usuario: App1 (bloc de notas de Windows), que es un editor de texto incluido en los sistemas operativos Microsoft Windows, y App2 (Traductor de Google). Siguiendo la tarea de prueba propuesta en el capítulo 5, cada aplicación tiene tres historias de usuario asociadas para ser utilizadas con la herramienta de prueba de IU. Cada escenario de prueba generado por la herramienta valida los widgets en la interfaz de usuario de la aplicación correspondiente. Después de ejecutar los escenarios de prueba con SikuliX, la herramienta muestra la lista de acciones en las ventanas de salida que ejecuta el escenario de prueba. Cuando la acción no se pudo ejecutar, la herramienta muestra la excepción en la ventana de salida. Luego, los participantes deben decidir si el caso de prueba pasó, falló o tuvo algún otro problema que no permitió que se ejecutara. Se inyectaron en los escenarios de prueba diferentes defectos, como la apariencia incorrecta de los widgets o el estado incorrecto de los widgets.

Para la realización del estudio, se proporciona a los participantes el siguiente material: (i) una máquina virtual para VirtualBox con todo el software requerido para el estudio, (ii) una guía de instalación y uso, (iii) un ejemplo con cuatro historias de usuario que se utilizará durante la fase de formación, (iv) un video instructivo y (v) un conjunto de tres historias de usuario para cada sistema utilizado como App1 y App2.

6.2.9 Procedimiento

El procedimiento se muestra en la Fig. 19 y sus 8 etapas principales se explican a continuación.

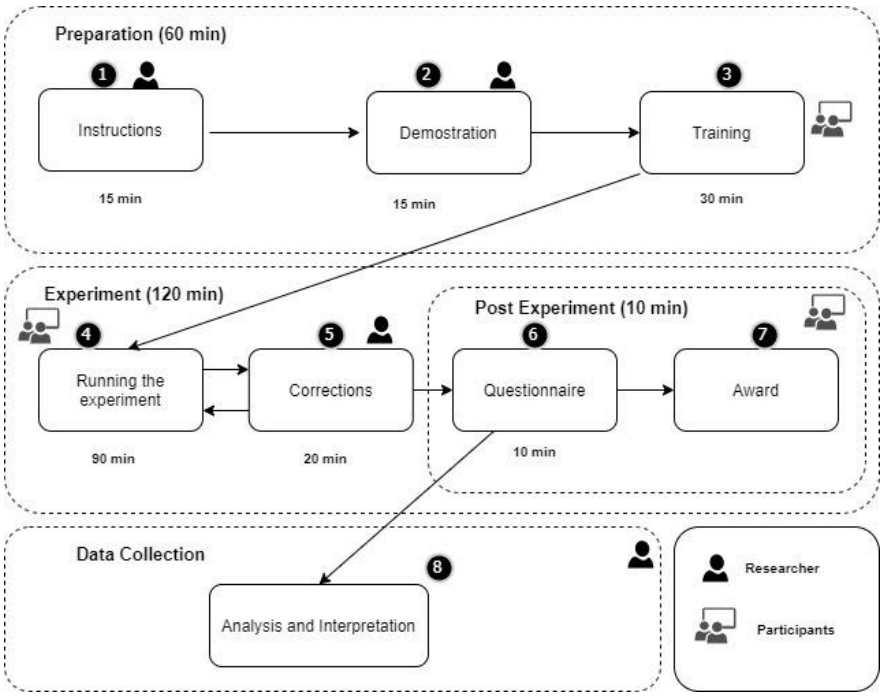


Fig. 19. Procedimiento experimental para medir la experiencia del usuario

6.2.9.1 Sesión de Capacitación

Los detalles del estudio se explicaron en la primera sesión (15 minutos), se mostró un video de demostración a los participantes. Debido a que UI-Test era una nueva herramienta para los participantes, se dio una capacitación durante unos 30 minutos, incluidas instrucciones para configurar la máquina virtual y la ejecución del proceso de prueba de la herramienta utilizando una aplicación de prueba (User CRUD). También se aseguró que todos los participantes tuvieran un conocimiento uniforme (por ejemplo, alguien podría haberse perdido en la secuencia de pasos realizados), por lo que se les pidió que reprodujeran las tareas de prueba de la IU en sus computadoras y que informaran resultados o cualquier problema que ocurriera.

6.2.9.2 Sesión Experimental

Esta etapa tomó alrededor de 120 minutos y sus actividades se pueden ver en la Sesión Experimental en la Fig. 19. Primero, se explicaron los detalles del experimento

(10 minutos). Luego, los participantes fueron divididos aleatoriamente en dos grupos (es decir, grupos de GUI y WUI) y se entregaron los recursos (por ejemplo, historias de usuarios, descripción de la aplicación) requeridos para esta sesión (20 minutos). Todos los participantes derivaron los modelos de prueba y los scripts de prueba de cada aplicación y los ejecutaron (consulte la Subsección 6.2.8 para obtener más detalles sobre las tareas de prueba) (40 minutos). Es importante resaltar que los participantes pudieron subir sus resultados en cualquier momento dentro de esta etapa. Seguidamente se indicaron las correcciones que debían realizarse para que se pudieran realizar los casos de prueba no ejecutados (20 minutos).

6.2.9.3 Post-experimento

Después de finalizar la actividad de prueba de IU, se pidió a los participantes que completaran un breve cuestionario demográfico (10 minutos) y dieran información sobre la experiencia del usuario de UI-Test y comentarios para mejorar la herramienta (consulte la Subsección 6.2.8 para obtener más detalles sobre los cuestionarios). Después de procesar todas las presentaciones de los participantes, se recompensó a los participantes que completaron todas las actividades.

6.2.9.4 Recopilación de datos

Los valores de la experiencia del usuario se calcularon a partir de la información registrada en este experimento. Los resultados se dan en la siguiente sección.

6.2.10 Análisis e Interpretación de Resultados

Esta sección describe el análisis e interpretación de los resultados relacionados con nuestra variable de respuesta para PI2. Al igual que en el estudio cuantitativo, el análisis estadístico se realizó en el software SPSS.

Dado que la segunda pregunta de investigación PI2 tenía como objetivo evaluar la calidad de la experiencia del usuario de UI-Test, se comparan los resultados del

cuestionario por el tipo de IU (es decir, GUI y WUI). La Tabla 10 muestra el valor de esta métrica.

Tabla 10. Valores de la experiencia del usuario recopilados por el cuestionario

Sujeto ID	UI Type	Experiencia de Usuario		
		Calidad Pragmática	Calidad Hedónica	General
S1	GUI	1.00	-0.50	0.25
S2	GUI	1.25	1.25	1.25
S3	GUI	2.00	3.00	2.50
S4	GUI	1.75	0.50	1.13
S5	GUI	0.50	2.50	1.50
S6	GUI	2.50	1.75	2.13
S7	GUI	3.00	3.00	3.00
S8	GUI	2.75	2.75	2.75
S9	WUI	2.50	0.75	1.63
S10	WUI	0.25	0.00	0.13
S11	WUI	1.25	1.25	1.25
S12	WUI	2.25	2.00	2.13
S13	WUI	1.00	1.00	1.00
S14	WUI	1.00	1.00	1.00
S15	WUI	0.25	0.00	0.13
S16	WUI	2.00	1.75	1.88
S17	WUI	1.00	1.25	1.13

La consistencia de las escalas de calidad pragmática y calidad hedónica fue razonablemente buena. Para GUI, los valores de Alfa de Cronbach correspondientes

fueron 0,87 (calidad pragmática) y 0,77 (calidad hedónica), y para WUI fueron 0,84 (calidad pragmática) y 0,65 (calidad hedónica).

Se realizaron pruebas de Shapiro-Wilk para evaluar la normalidad de las muestras. Se usó esta prueba porque es más apropiada para tamaños de muestra pequeños (<50 muestras).

Dado que todos los valores Sig. para las pruebas de Shapiro-Wilk fueron 0.767 para GUI y 0.471 para WUI (ver Tabla 11), estas variables siguen una distribución normal (> 0.05), por lo que se cosideran ambos tipos de GUI como grupos independientes. Luego se utilizó la prueba T-Student para probar la segunda hipótesis nula (H20).

Tabla 11. Pruebas de normalidad

	UI Type	**Shapiro-Wil**		
		Statistic	**Df**	**Sig.**
Experiencia de Usuario	GUI	0.956	8	0.767
Calidad General	WUI	0.929	9	0.471

La Figura 20 muestra el diagrama de caja que contiene los valores de la experiencia del usuario por tipo de interfaz de usuario y la Tabla 12 muestra las medias y la desviación estándar.

La Tabla 13 muestra los resultados de la prueba T-student, en la que no se encontró ninguna diferencia estadísticamente significativa entre la experiencia de usuario de GUI (M=1,81; DE=0,94) y WUI (M=1,14; DE=0,69) t(15)=1,69, p=0,112, g=0,820825.

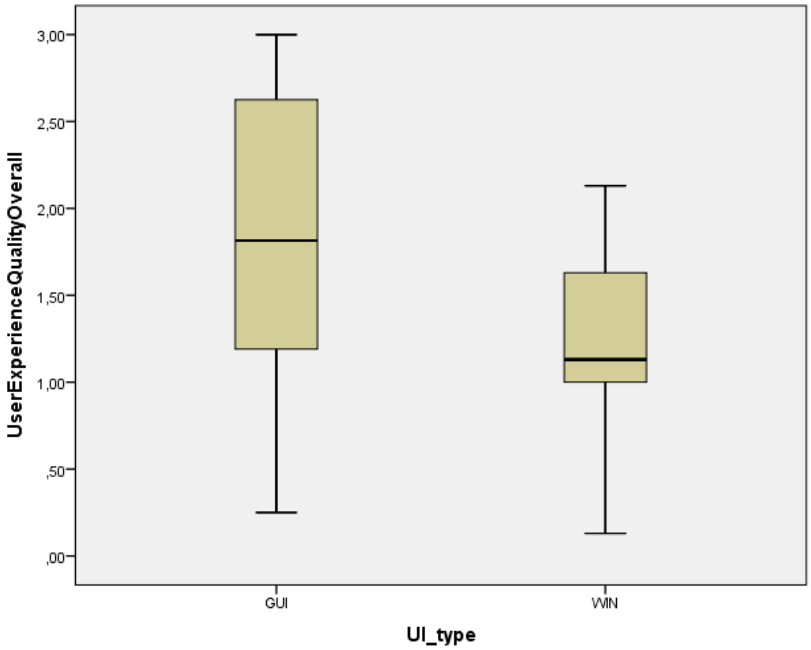


Fig. 20. Diagrama de caja para la experiencia del usuario por tipo de GUI

Por tanto, se acepta la hipótesis nula H20. En otras palabras, con un 95% de confianza se puede decir que el valor promedio de la calidad de la experiencia del usuario es similar para ambos tipos de IU.

Tabla 12. Métrica de estadísticas de grupo para la experiencia del usuario

	UI Type	N	Mean	Std. Derivation	Std Error Mean
Calidad Experiencia de Usuario	GUI	8	1.8138	0.93989	0.33230
General	WUI	9	1.1422	0.69456	0.23152

Tabla 13. Resumen de la prueba de hipótesis H20

	Prueba de muestras independientes									
		Prueba de Levene para la igualdad de varianzas		Prueba t para la igualdad de la media						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference Lower	Upper
Calidad de la experiencia del usuario en general	Igualdad de varianzas asumidas	1.618	0.223	1.689	15	0.112	0.67153	0.39760	-0.17594	1.51899
	Igualdad de varianzas no asumidas			1.658	12.805	0.122	0.67153	0.40500	-0.20478	1.54784

6.3 Discusión Final

A continuación, vamos a discutir e interpretar los hallazgos del experimento de acuerdo a las preguntas de investigación planteadas (sección 6.1.2 y sección 6.2.2). Los principales resultados de la eficacia de los casos de prueba UI-Test (PI1) y la Calidad de la experiencia del usuario (PI2) son los siguientes: aunque la GUI tiene un valor ligeramente mejor que la WUI, se puede confirmar que el tipo de IU no influye en las variables analizadas.

6.3.1 PI1: ¿Cómo influye el tipo de interfaz de usuario en la eficacia de los casos de prueba generados por la herramienta UI-Test cuando se prueban historias de usuario a nivel visual?

La mayoría de los scripts de prueba generados (> 60%) permitieron encontrar inconsistencias entre los requisitos que se obtuvieron (es decir, la historia del usuario) y los que realmente se implementaron como widgets para ambos tipos de IU (es decir, GUI y WUI). Estos resultados sugieren que las historias de usuarios muestran potencial para ser utilizadas como fuente para generar scripts de prueba y usarlos en pruebas de interfaz de usuario visual con la herramienta de prueba de interfaz de usuario.

Sin embargo, también se encontró que hay problemas que requieren la intervención manual del tester para ser resueltos y hacer que los scripts de prueba sean ejecutables, como se explica en la Tabla 14

Tabla 14. Algunos problemas encontrados en los scripts de prueba que impidieron su ejecución

Problema	Descripción	Posible solución
PROBLEMAS ENCONTRADOS EN AMBOS TIPOS DE IU: GUI Y WUI		
Apariencia incorrecta de widgets	No es posible hacer clic en un botón/icono porque no se reconoce.	Capturar la imagen del widget otra vez
Tipo o formato de datos incorrecto en un campo de texto.	Un campo de texto acepta un tipo de datos diferente al ingresado.	Cambiar los valores de las variables
Estado incorrecto de los widgets.	No es posible hacer clic en un botón/icono ya que no está activado/visible.	Especifique las acciones anteriores para que el botón se active/sea visible.
PROBLEMAS ENCONTRADOS EN WUIs		
Widget faltante	Retraso en la carga de la página web	Agregue un tiempo de espera para hacer clic en el widget.

Widget faltante	Necesita usar el desplazamiento antes de hacer clic en el widget.	Agregue el comando de desplazamiento con el código SikuliX.
PROBLEMAS ENCONTRADOS EN GUIs		
Widget faltante	Es necesario manipular varios menús y submenús para acceder al widget.	Agregue las acciones necesarias para acceder al widget a través de los menús y submenús.

A partir de estos resultados, aunque preliminares, se sugiere la necesidad de realizar más investigaciones con más aplicaciones e historias de usuarios que permitan evaluar la escalabilidad del marco y el soporte de la herramienta propuesta.

6.3.2 PI2: ¿Cómo se vio afectada la experiencia del usuario durante las tareas de prueba en las IU?

Se aplica el cuestionario de experiencia del usuario (UEQ) para medir la impresión subjetiva de los usuarios hacia la experiencia del usuario del marco y la herramienta. A partir de estos resultados, se puede ver una buena evaluación general (1,81 para GUI; 1,14 para WUI). Sin embargo, se logró un mejor valor en los aspectos pragmáticos que en los hedónicos, lo que indica que es posible mejorar la forma en que el producto (es decir, el marco y la herramienta de soporte) capta la atención y el interés del usuario. Es decir, hacerlo más interesante y estimulante para los usuarios. Estos resultados positivos se vieron reforzados por la retroalimentación cualitativa obtenida durante el post-estudio. Algunos participantes consideraron que UI-Test era útil e interesante, ya que les permitía realizar las primeras tareas de prueba. Sin embargo, algunos consideran necesario mejorar la facilidad de uso de la herramienta, especialmente cuando se agregan localizadores de IU y valores de variables para probar scripts.

6.4 Amenazas a la validez

Existen varias amenazas que potencialmente afectan la validez de nuestros experimentos, incluidas las amenazas a la validez interna, las amenazas a la validez externa y las amenazas a la validez de la construcción.

6.4.1 Validez interna.

Esta validez está relacionada con sujetos y mediciones en el experimento que podrían afectar las variables observadas. Como los sujetos de nuestro experimento podrían haber tenido un conocimiento previo diferente de las herramientas antes del experimento, se intentó minimizar esta amenaza entrenándolos para homogeneizar su conocimiento y experiencia. También se identificó la configuración del experimento como una posible amenaza; que se mitigó realizando el experimento en condiciones similares para cada participante (es decir, material, tareas de prueba). Por ejemplo, la configuración de las herramientas requeridas (es decir, UI-Test, ConcurTaskTree y SikuliX) fueron predefinidas en la máquina virtual para simplificar la instalación del software. Sin embargo, aunque se notificó a los participantes una semana antes del experimento, 9 sujetos no descargaron la máquina virtual y, por lo tanto, no participaron en el experimento.

6.4.2 Validez externa.

Este tema trata sobre la generalización de los resultados obtenidos; las posibles amenazas podrían ser la selección de temas y aplicaciones seleccionadas. Con respecto a los participantes (es decir, estudiantes de informática de último año) como sujetos experimentales, varios autores sugieren que los resultados pueden generalizarse a los profesionales industriales (Runeson, 2003). Sin embargo, una de las principales limitaciones de este estudio fue que hubo pocos participantes, por lo que se necesita más investigación empírica para confirmar los resultados. Con respecto a las aplicaciones seleccionadas, esta amenaza se redujo mediante el uso de diez aplicaciones reales de diferentes tamaños (ver Sección 6.1.5), dominios (p. Ej., red social, servicios, correo) que contenían los diferentes widgets de IU requeridos para este estudio.

6.4.3 Validez de Construcción.

Las amenazas a la validez de la construcción se refieren a la idoneidad de las métricas de evaluación y de los instrumentos seleccionados. Sin embargo, se utilizaron métricas conocidas para medir la cobertura de ejecución de los scripts de prueba generados por UI-Test. Con respecto a la calidad de la experiencia del usuario, los participantes lo completaron en su idioma nativo (es decir, español). Además, se usó un análisis de correlación entre ítems para evaluar su validez de construcción (ver Subsección 6.2.10). Por lo tanto, minimizamos las amenazas para la validez de construcción.

CAPÍTULO 7: CONCLUSIONES Y TRABAJOS FUTUROS

Dentro de este capítulo se presentan las conclusiones obtenidas en la ejecución de este trabajo de titulación, además, se plantean ciertas líneas de investigación y trabajos futuros que pueden surgir.

7.1 Conclusiones Finales

Las siguientes conclusiones se basan en el logro de los objetivos planteados en este trabajo de titulación, para lo cual se justifica mediante información que ha sido evidenciada en este trabajo, como se describen a continuación:

1. Definir un marco conceptual relacionado para la validación de interfaces de usuario gráficas usando técnicas de prueba en un ambiente dirigido por modelos.

Este objetivo fue cumplido en su totalidad, debido a que se resumió los fundamentos de nuestro marco teórico (ver Capítulo 3), que son importantes porque establecen los criterios utilizados para la construcción del marco de generación automática de casos de prueba.

2. Diseñar e implementar una herramienta de validación basada en modelos para interfaces de usuario gráficas integrada en un ambiente dirigido por modelos.

Este objetivo se cumplió en su totalidad ya que se construyó una herramienta llamada UI-Test qué promueven la práctica de generar evidencia basada en la especificación

de historias de usuarios ágiles para validar que los requisitos funcionales están incluidos en la versión final de las interfaces de usuario del software desarrollado.

Se implementó la herramienta utilizando la plataforma Eclipse y el lenguaje de programación Java. Esta herramienta se puede utilizar en tres plataformas: Microsoft Windows, Linux, MacOS ya que la máquina virtual de Java está disponible en estas plataformas. Este marco de generación automática de casos de prueba ayudará a reducir el esfuerzo necesario, mejorando la calidad de los casos de prueba y la cobertura de los requisitos de los casos de prueba generados a partir de las historias de usuario. Este trabajo de investigación se puede aplicar en proyectos de desarrollo utilizando metodologías ágiles para probar sus productos de software.

3. Realizar una prueba de concepto de la herramienta para validar la contribución en el aseguramiento de la calidad de las interfaces gráficas.

Este objetivo se cumplió realizando dos evaluaciones. Primero, a través de una evaluación cuantitativa, se midió la efectividad de los scripts de prueba generados por la herramienta. En segundo lugar, mediante una evaluación cualitativa, se midió la impresión subjetiva de los usuarios hacia la experiencia del usuario del marco y la herramienta.

La efectividad del script de prueba se calculó en función de la cobertura de ejecución de los casos de prueba generados (es decir, la tasa de número de pruebas ejecutadas) para ambos tipos de IU (es decir, GUI y WUI). Las métricas planteadas en el experimento fueron evaluadas usando el método experimental GQM. A partir de los resultados, la eficacia de las pruebas no se vio afectada por el tipo de interfaz de usuario. Estos resultados sugieren que la herramienta UI-Test puede beneficiar a los evaluadores al validar que las acciones propuestas en las historias de usuario se pueden ejecutar en la interfaz de usuario de la aplicación.

Con respecto a la calidad de la experiencia del usuario, se encontró que la mayoría de los participantes estuvieron de acuerdo en considerar la herramienta UI-Test como de buena calidad para la experiencia del usuario.

7.2 Trabajo Futuro

A continuación, se plantean actividades que se podrían realizar para ampliar el estudio que se ha realizado en este trabajo de titulación.

- Como una de las principales limitaciones de este estudio fue que hubo pocos participantes, se necesita más investigación empírica para identificar las características de los escenarios de prueba que conducirían a una mayor efectividad. También se tiene la intención de replicar este experimento en una amplia variedad de dominios y aplicaciones para confirmar los resultados.
- Debido a que en la evaluación de experiencia de usuario se logró un mejor valor en los aspectos pragmáticos que en los hedónicos, es posible mejorar la forma en que la herramienta capta la atención y el interés del usuario, es decir, hacerlo más interesante y estimulante para los usuarios.
- Otra de las mejoras a la herramienta UI-Test podría ser agregar una nueva funcionalidad en la cual se gestione un diccionario de palabras (acciones) para lograr a una mejor cobertura del lenguaje usado en las historias de usuario.

REFERENCIAS

Abran, A. (Ed.). (2004). *Guide to the software engineering body of knowledge, 2004 version: SWEBOK ; a project of the IEEE Computer Society Professional Practices Committee*. IEEE Computer Society.

Aho, P., Alégroth, E., Oliveira, R. A. P., & Vos, T. E. J. (2016). Evolution of Automated Regression Testing of Software Systems Through the Graphical User Interface. *ACCSE 2016: The First International Conference on Advances in Computation, Communications and Services*, 16-21. https://research.ou.nl/en/publications/evolution-of-automated-regression-testing-of-software-systems-thr

Aho, P., Kanstrén, T., Räty, T., & Röning, J. (2014). Automated Extraction of GUI Models for Testing. En *Advances in Computers* (Vol. 95, pp. 49-112). https://doi.org/10.1016/B978-0-12-800160-8.00002-4

Aho, P., Suarez, M., Kanstrén, T., & Memon, A. M. (2014). Murphy Tools: Utilizing Extracted GUI Models for Industrial Software Testing. *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, 343-348. https://doi.org/10.1109/ICSTW.2014.39

Aho, P., Suarez, M., Memon, A., & Kanstrén, T. (2015). Making GUI Testing Practical: Bridging the Gaps. *2015 12th International Conference on Information Technology - New Generations*, 439-444. https://doi.org/10.1109/ITNG.2015.77

Alegroth, E., Feldt, R., & Ryrholm, L. (2015). Visual GUI testing in practice: Challenges, problems and limitations. *Journal of Empirical Software Engineering*, *20*(3), 694-744.

Alfraihi, H., Lano, K., Kolahdouz-Rahimi, S., Sharbaf, M., & Haughton, H. (2018). The Impact of Integrating Agile Software Development and Model-Driven Development: A Comparative Case Study. En F. Khendek & R. Gotzhein (Eds.), *System Analysis and Modeling. Languages, Methods, and Tools for Systems Engineering* (pp. 229-245). Springer International Publishing. https://doi.org/10.1007/978-3-030-01042-3_14

Azimian, F., Faghih, F., Kargahi, M., & Mahdi Mirdehghan, S. M. (2019). Energy Metamorphic Testing for Android Applications. *2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)*, 1-6. https://doi.org/10.1109/PIMRCW.2019.8880814

Banerjee, I., Nguyen, B., Garousi, V., & Memon, A. (2013). Graphical user interface (GUI) testing: Systematic mapping and repository. *Information and Software Technology*, *55*(10), 1679-1694. https://doi.org/10.1016/j.infsof.2013.03.004

Bass, L., Bergey, J., Clements, P., Merson, P., Ozkaya, I., & Sangwan, R. (2006). *A Comparison of Requirements Specification Methods from a Software Architecture Perspective*. https://doi.org/10.21236/ada455888

Beck, K., & Andres, C. (2004). *Extreme Programming Explained: Embrace Change (2nd Edition)*.

Belli, F., & Linschulte, M. (2008). Event-Driven Modeling and Testing of Web Services. *2008 32nd Annual IEEE International Computer Software and Applications Conference*, 1168-1173. https://doi.org/10.1109/COMPSAC.2008.144

Brüning, J., & Forbrig, P. (2011). TTMS: A Task Tree Based Workflow Management System. En T. Halpin, S. Nurcan, J. Krogstie, P. Soffer, E. Proper, R. Schmidt, & I. Bider (Eds.), *Enterprise, Business-Process and Information Systems Modeling* (pp. 186-200). Springer. https://doi.org/10.1007/978-3-642-21759-3_14

Coppola, R., Morisio, M., & Torchiano, M. (2018). Maintenance of Android Widget-Based GUI Testing: A Taxonomy of Test Case Modification Causes. *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 151-158. https://doi.org/10.1109/ICSTW.2018.00044

Elghondakly, R., Moussa, S., & Badr, N. (2015). Waterfall and agile requirements-based model for automated test cases generation. *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*, 607-612. https://doi.org/10.1109/IntelCIS.2015.7397285

Finsterwalder, M. (2001). *Automating Acceptance Tests for GUI Applications in an Extreme Programming Environment*.

Gao, Z., Chen, Z., Zou, Y., & Memon, A. M. (2016). SITAR: GUI Test Script Repair. *IEEE Transactions on Software Engineering*, *42*(2), 170-186. https://doi.org/10.1109/TSE.2015.2454510

Granda, M., Parra, O., & Alba-Sarango, B. (2021). Towards a Model-Driven Testing Framework for GUI Test Cases Generation from User Stories: *Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering*, 453-460. https://doi.org/10.5220/0010499004530460

Grangel, R., & Campos, C. (2019). Agile Model-Driven Methodology to Implement Corporate Social Responsibility. *Computers & Industrial Engineering*, *127*, 116-128. https://doi.org/10.1016/j.cie.2018.11.052

Iyama, M., Kirinuki, H., Tanno, H., & Kurabayashi, T. (2018). Automatically Generating Test Scripts for GUI Testing. *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 146-150. https://doi.org/10.1109/ICSTW.2018.00043

Kamal, M. M., Darwish, S. M., & Elfatatry, A. (2019). Enhancing the Automation of GUI Testing. *Proceedings of the 2019 8th International Conference on Software and Information Engineering*, 66-70. https://doi.org/10.1145/3328833.3328842

Kassab, M. (2015). The changing landscape of requirements engineering practices over the past decade. *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, 1-8. https://doi.org/10.1109/EmpiRE.2015.7431299

Kresse, A., & Kruse, P. M. (2016). Development and maintenance efforts testing graphical user interfaces: A comparison. *Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation*, 52-58. https://doi.org/10.1145/2994291.2994299

Kull, A. (2012). Automatic GUI Model Generation: State of the Art. *2012 IEEE 23rd*

International Symposium on Software Reliability Engineering Workshops, 207-212. https://doi.org/10.1109/ISSREW.2012.23

Latiu, G., Cret, O., & Vacariu, L. (2013). Graphical User Interface Testing Optimization for Water Monitoring Applications. *2013 19th International Conference on Control Systems and Computer Science*, 640-645. https://doi.org/10.1109/CSCS.2013.32

Lelli, V., Blouin, A., & Baudry, B. (2015). Classifying and Qualifying GUI Defects. *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, 1-10. https://doi.org/10.1109/ICST.2015.7102582

Lucassen, G., Dalpiaz, F., Werf, J. M. E. M. van der, & Brinkkemper, S. (2016). The Use and Effectiveness of User Stories in Practice. En M. Daneva & O. Pastor (Eds.), *Requirements Engineering: Foundation for Software Quality* (pp. 205-222). Springer International Publishing. https://doi.org/10.1007/978-3-319-30282-9_14

Lucassen, G., Robeer, M., Dalpiaz, F., van der Werf, J. M. E. M., & Brinkkemper, S. (2017). Extracting conceptual models from user stories with Visual Narrator. *Requirements Engineering*, *22*(3), 339-358. https://doi.org/10.1007/s00766-017-0270-1

Memon, A. M. (2007). An event-flow model of GUI-based applications for testing. *Software Testing, Verification and Reliability*, *17*(3), 137-157. https://doi.org/10.1002/stvr.364

Moreira, M. E. (2013). *Being Agile: Your Roadmap to Successful Adoption of Agile*. Apress.

Mori, G., Paternò, F., & Santoro, C. (2002). CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. *Software Engineering, IEEE Transactions on*, *28*, 797-813. https://doi.org/10.1109/TSE.2002.1027801

Paterno, F., Mancini, C., & Meniconi, S. (1997). ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. En S. Howard, J. Hammond, & G. Lindgaard (Eds.), *Human-Computer Interaction INTERACT '97: IFIP TC13 International Conference on Human-Computer Interaction, 14th–18th July 1997, Sydney, Australia* (pp. 362-369). Springer US. https://doi.org/10.1007/978-0-387-35175-9_58

Ramler, R., Klammer, C., & Wetzlmaier, T. (2019). Lessons learned from making the

transition to model-based GUI testing. *Proceedings of the 10th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, 22-27. https://doi.org/10.1145/3340433.3342823

Rane, P. (2017). Automatic Generation of Test Cases for Agile using Natural Language Processing. *Undefined*. https://www.semanticscholar.org/paper/Automatic-Generation-of-Test-Cases-for-Agile-using-Rane/4a3a06271cb81a915904378461752d5ff8e28feb

Rengifo, Y. S. P., Suarez, J. A. M., & Correa, E. D. C. (2015). Desarrollo Dirigido por Modelos (MDD) en el Contexto Educativo. *Scientia et Technica*, *20*(2), 172-181. https://doi.org/10.22517/23447214.9321

Runeson, P. (2003). Using Students as Experiment Subjects – An Analysis on Graduate and Freshmen Student Data. *Proceedings 7th International Conference on Empirical Assessment & Evaluation in Software Engineering*, 95-102.

Runeson, P., & Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131. https://doi.org/10.1007/s10664-008-9102-8

Sakal, M. (s. f.). *GUI vs. WUI Through the Prism of Characteristics and Postures*. Recuperado 14 de agosto de 2021, de https://www.semanticscholar.org/paper/GUI-vs.-WUI-Through-the-Prism-of-Characteristics-Sakal/be41c8d860bd84f6bcca286de90e566ab7971da3

Schrepp, M., Hinderks, A., & Thomaschewski, J. (2017). Design and Evaluation of a Short Version of the User Experience Questionnaire (UEQ-S). *International Journal of Interactive Multimedia and Artificial Intelligence*, *4*(6), 103. https://doi.org/10.9781/ijimai.2017.09.001

Silva, J. C., Saraiva, J., & Campos, J. C. (2009). A generic library for GUI reasoning and testing. *Proceedings of the 2009 ACM symposium on Applied Computing*, 121-128. https://doi.org/10.1145/1529282.1529307

Silva, J., Campos, J., & Paiva, A. (2007). Model-based User Interface Testing With Spec Explorer and ConcurTaskTrees. En *Electronic Notes in Theoretical Computer Science—ENTCS* (Vol. 208). https://doi.org/10.1016/j.entcs.2008.03.108

Tao, C., Gao, J., & Wang, T. (2017). *An Approach to Mobile Application Testing Based on Natural Language Scripting*. 260-265. https://doi.org/10.18293/SEKE2017-170

Wautelet, Y., Heng, S., Kolp, M., & Mirbel, I. (2014). Unifying and Extending User Story Models. En M. Jarke, J. Mylopoulos, C. Quix, C. Rolland, Y. Manolopoulos, H. Mouratidis, & J. Horkoff (Eds.), *Advanced Information Systems Engineering* (pp. 211-225). Springer International Publishing. https://doi.org/10.1007/978-3-319-07881-6_15

Wimmer, M., & Burgueño, L. (2013). Testing M2T/T2M Transformations. En A. Moreira, B. Schätz, J. Gray, A. Vallecillo, & P. Clarke (Eds.), *Model-Driven Engineering Languages and Systems* (pp. 203-219). Springer. https://doi.org/10.1007/978-3-642-41533-3_13

Yuan, X., Cohen, M. B., & Memon, A. M. (2011). GUI Interaction Testing: Incorporating Event Context. *IEEE Transactions on Software Engineering*, *37*(4), 559-574. https://doi.org/10.1109/TSE.2010.50

ANEXOS

Anexo 1: Artículo publicado en ENASE 2021 - XVI Conferencia Internacional sobre Evaluación de Nuevos Enfoques de Ingeniería de Software.

Towards a Model-Driven Testing Framework for GUI Test Cases Generation from User Stories

Maria Fernanda Granda[a], Otto Parra[b] and Bryan Alba-Sarango[c]
Department of Computer Science, Universidad de Cuenca, Av. 12 de Abril s/n, Cuenca, Ecuador

Keywords: Test Cases, User Stories, Requirements, GUI-based Testing, Model-Driven Testing.

Abstract: In the software testing stage, it is possible to benefit from combining the requirements with the testing specification activities. On the one hand, the specification of the tests will require less manual effort, since they are defined or generated automatically from the requirements specification. On the other hand, the specification of requirements itself will end up having a higher quality due to the use of a more structured language, reducing typical problems such as ambiguity, inconsistency, and inaccuracy. This research proposes a model-based framework that promotes the practice of generating test cases based on the specification of Agile user stories to validate that the functional requirements are included in the final version of the user interfaces of the developed software. To show the applicability of the approach, a specification of requirements based on user stories, a task model using ConcurTaskTree, and the Sikulix language are used to generate tests at the graphical interface level. The approach includes transformations; such as task models in test scripts. Then, these test scripts are executed by the Sikulix test automation framework.

1 INTRODUCTION

To react to the changing software development market in a more efficient way, the adoption of Agile development practices is gaining momentum (Kassab 2015). The Agile methodology is an iterative and incremental approach to software development, where the requirements and solutions evolve over time according to the need of the stakeholders. How to test the application to seek evidence that the functionality requested by end users or stakeholders is provided by the application now emerges as an issue. However, designing and executing test cases is very time-consuming and error-prone task when done manually and frequent changes in requirements reduce the reusability of these manually written test cases. According Latiu et al. (Latiu, Cret, and Vacariu 2013), automatic testing based on Graphical User Interfaces (GUIs) may be a good alternative because it is more accurate, reliable and efficient.

The existing methods to generate test cases from user stories have not been widely accepted in practice, or because the results obtained have a very low accuracy (Garm Lucassen et al. 2017).

In this work, we consider the version integrated of two methodologies to develop software such as Agile and Model-driven Development. This version is called Agile Model-driven Development (AMDD) (Alfraihi, Lano, and Kolahdouz-rahimi 2018). On the one hand, Model-Driven Engineering is a well-known software development paradigm which provides many benefits to develop suitable solutions of software. On the other hand, Agile Methods are a good paradigm to gain a better understanding of requirements (Grangel and Campos 2019).

In this paper, we aim at providing an approach to accommodate the following issues:

- How to generate test cases from user stories and that they to adapt to the evolution of the requirements in an easy way?
- How to transform the test cases into an executable script so that tester can minimize the effort to run them?
- How to simulate the interactions between the

Anexo 2: Captura de parte del artículo académico realizado en base al trabajo de titulación.

UI-Test: A Model-based Framework for Visual UI Testing– Qualitative and Quantitative Evaluation

Bryan Alba[1[0000-0001-9418-9489]], Maria Fernanda Granda[1[10000-0002-5125-8234]]

and Otto Parra[1[0000-0003-3004-1025]]

[1] Computer Science Department, Universidad de Cuenca, Av. 12 de Abril s/n, Cuenca, Ecuador
{bryan.albas, fernanda.granda, otto.parra}@ucuenca.edu.ec

Abstract. During the testing stage in the software development life cycle, developers can take advantage of combining the requirements specification with the testing specification. This will allow the specification of the tests to require less manual effort, since, they can be defined or generated automatically from the requirements specification. The requirements specification will thus be based on a more structured language, gaining in quality and reducing ambiguity, inconsistency, and inaccuracy. In this research work, the UI-Test model-based methodological framework and its tool support are proposed. Both of these can generate evidence based on the specification of agile user stories that are used in the validation of the functional requirements that must be included in the final version of the user interfaces of the developed software. Our proposal makes use of two model transformations to obtain the test scripts from user stories that will be applied in the process using SikuliX for automated visual UI testing. The results of the empirical evaluation of the effectiveness and user experience of the framework and its tool support suggest that the UI-Test tool can benefit testers by confirming that the actions proposed in the user stories can be run on the UIs.

Keywords: UI-Test, Visual UI testing, user stories, test scripts, model-based testing, testing framework.

1 Introduction

In the field of Software Engineering, agile software development methodologies have gained momentum because of their benefits to software developers in the software development life cycle (SDLC) [1]. In this context, organizations are increasingly deploying agile methodologies in their software development projects in order to efficiently produce higher quality software in a shorter period of time. This methodology enables a software developer to be more flexible and responsive to changing environments and customer demands. However, it has been pointed out that agile methods largely ignore issues of designing the user interface (UI). To some extent this is understandable: agile processes are highly iterative and incremental, while traditional approaches to user interface design have been big-bang with heavy reliance on upfront design [2].

Anexo 3: Evidencia del envío del segundo artículo para publicación en un libro de Springer

ENASE 2021 - Camera Ready Submission for Springer Book Externo Recibidos ×

ENASE Secretariat <enase.secretariat@insticc.org> mar, 10 ago 18:41 (hace 5 días)
para fernanda.granda, mí, otto.parra

inglés › español Ver mensaje traducido Traducir siempre: inglés

Dear Dr. Maria Fernanda Granda,

We would like to inform you that we have just received the camera-ready version of the following paper for the Springer Book of ENASE 2021:

Title: UI-Test: A Model-based Framework for Visual UI Testing– Qualitative and Quantitative Evaluation
Subtitle:
Co-Author(s): Bryan Alba-Sarango and Otto Parra
Submission Date: 2021-08-11 12:41:00 AM

If this was not submitted by you or your co-authors please contact us immediately.

IMPORTANT NOTES
- The paper will be checked for format compliance to the Springer template. If adjustments are needed we will contact you shortly. Any text or material outside the defined margins will not be printed. Do NOT add headers or footers. They will be electronically added to the document.
- Don't forget to go to your author's area in PRIMORIS, download the related Springer's copyright document and when all is filled and signed, upload it to our system.

Best regards,
ENASE Secretariat

INSTICC office
Av. S. Francisco Xavier Lote 7 Cv. C
2910-616 Setubal Portugal
Tel: +351 265 520 184
Fax: +351 265 520 186
Web: http://www.enase.org

Anexo 4: Cuestionarios para la evaluación de UI-Test

1. Cuestionario demográfico

*Obligatorio

Correo electrónico *

Tu dirección de correo electrónico

¿Cuál es tu edad? *

- ◯ 18 - 20
- ◯ 21 - 28
- ◯ 29 - 40

¿Cuál es tu genero? *

- ◯ Femenino
- ◯ Masculino

Nivel de experiencia en (1 es menor experiencia y 5 mayor experiencia): *

	1	2	3	4	5
Pruebas de software	◯	◯	◯	◯	◯
Diseño de interfaces centradas en tareas	◯	◯	◯	◯	◯
Captura de requisitos	◯	◯	◯	◯	◯

2. Cuestionario corto de UEQ

Por favor, rellene el siguiente cuestionario con el fin de evaluar el producto. Se compone de pares de propiedades opuestas que el producto puede tener.
Por ejemplo, "aburrido" y "emocionante".
Para cada uno de los adjetivos incluidos en la tabla siguiente, marque en la escala del 1 al 7, considerando que los valores 1, 2 y 3 corresponden al adjetivo de la izquierda, los valores 5, 6 y 7 al adjetivo de la derecha.
El valor 4 significa que es un valor neutro

Pregunta 1 *

	1	2	3	4	5	6	7	
aburrido	○	○	○	○	○	○	○	emocionante

Pregunta 2 *

	1	2	3	4	5	6	7	
no interesante	○	○	○	○	○	○	○	interesante

Pregunta 3 *

	1	2	3	4	5	6	7	
original	○	○	○	○	○	○	○	convencional

Pregunta 4 *

	1	2	3	4	5	6	7	
obstructivo	○	○	○	○	○	○	○	impulsor de apoyo

Pregunta 5 *

	1	2	3	4	5	6	7	
complicado	○	○	○	○	○	○	○	fácil

Pregunta 6 *

	1	2	3	4	5	6	7	
convencional	○	○	○	○	○	○	○	novedoso

Pregunta 7 *

	1	2	3	4	5	6	7	
ineficiente	○	○	○	○	○	○	○	eficiente

Pregunta 8 *

	1	2	3	4	5	6	7	
claro	○	○	○	○	○	○	○	confuso

Tiene alguna sugerencia con respecto a la herramienta.

Texto de respuesta breve

Anexo 5: Guía de instalación de la máquina virtual y ejecución de la herramienta UI-Test

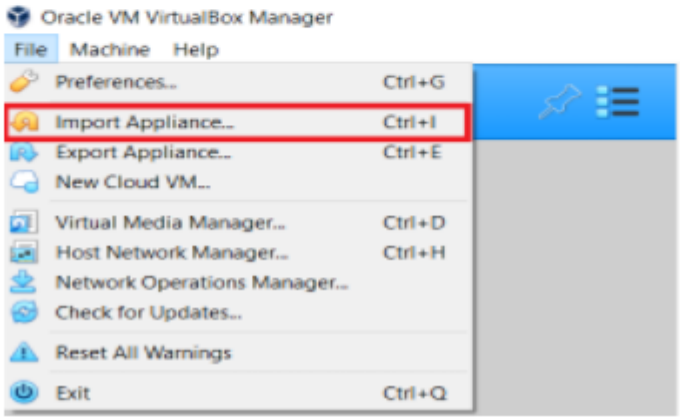
GUIA DE DESCARGA DE RECURSOS, INSTALACION E IMPORTACION DE MAQUINA VIRTUAL VIRTUALBOX Y EJECUCIÓN DE LA HERRAMIENTA UI-TEST

Paso 1: Descargar e instalar el software de virtualización VirtualBox para el sistema operativo que use, en caso de no tener, desde el siguiente link
https://www.virtualbox.org/wiki/Downloads

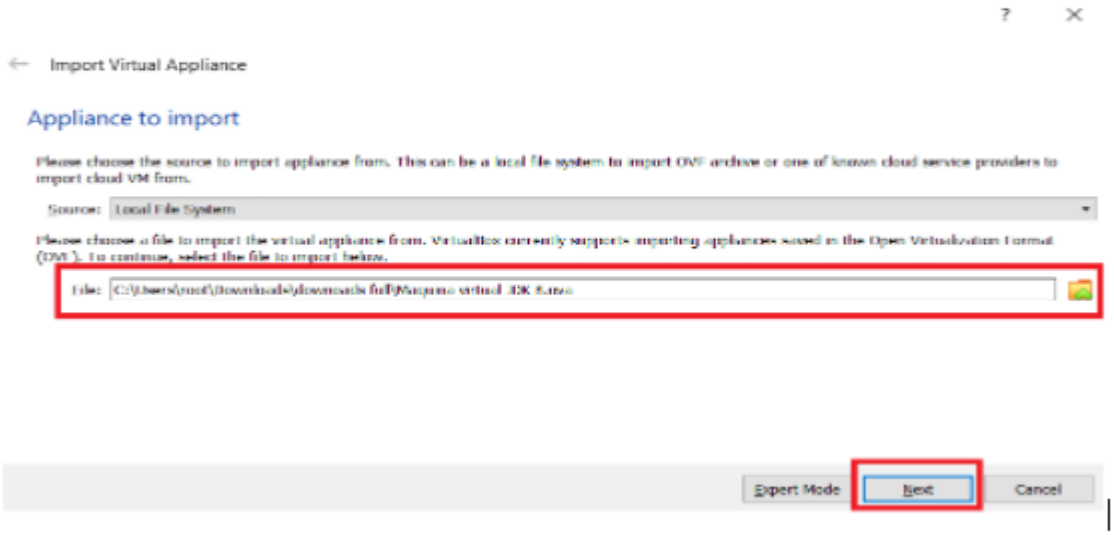
Paso 2: Descargar la máquina virtual .ova configurada desde el siguiente link
https://drive.google.com/drive/folders/1hj2Z3BVtTmQZ1H7a8JTqAiBZ6cor1Ygl?usp=sharing

Paso 3: Importar la máquina virtual a VirtualBox.

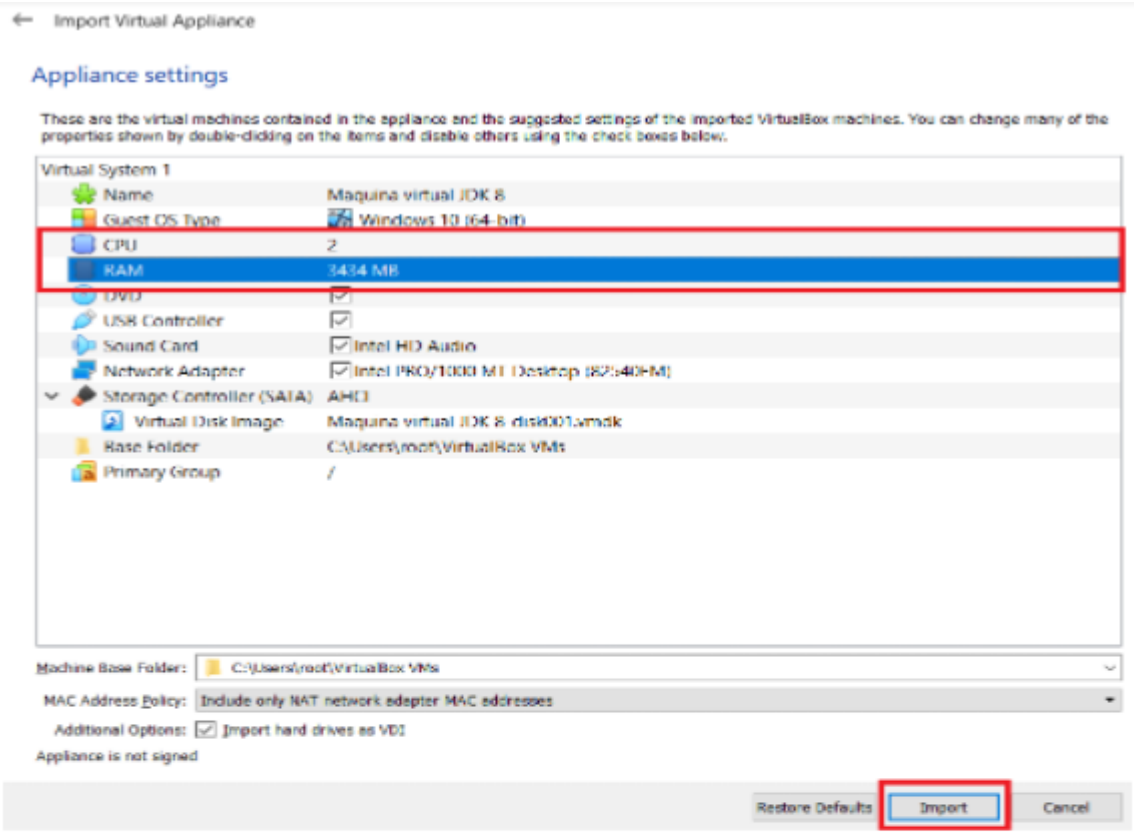
1. **Ir a File y seleccionar Importar o Ctrl + i**



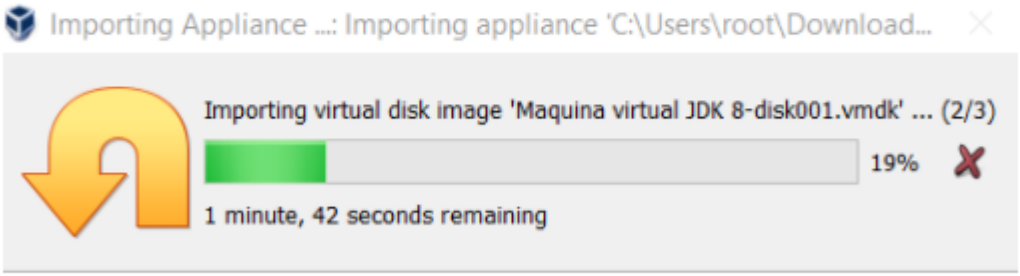
2. **En la siguiente ventana se selecciona la máquina virtual descargada en el paso 2 y presiona next.**



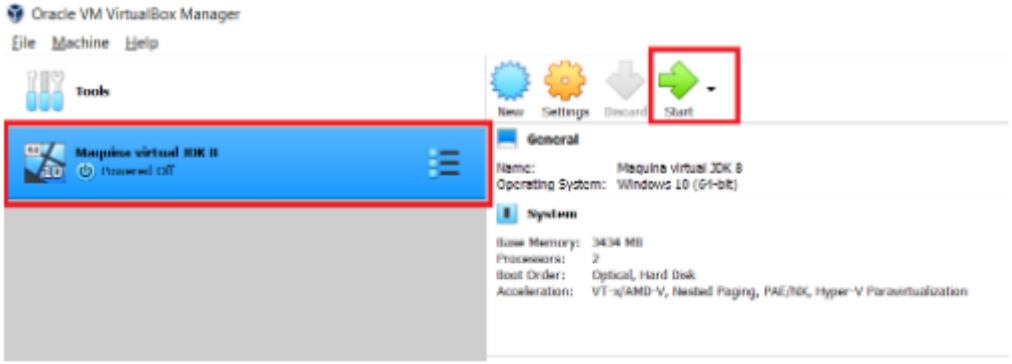
3. **En la siguiente ventana puede dejar todos los valores por defecto o aumentar la RAM y CPU si su computadora se los permite y luego presionar importar.**



4. **Aparecerá la siguiente ventana:**

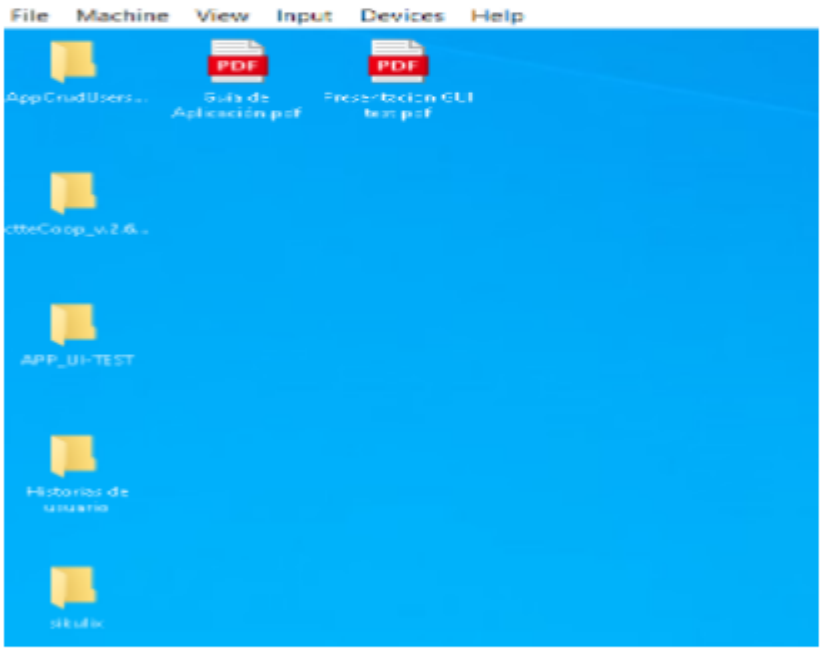


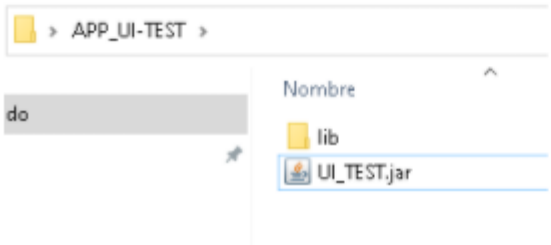
5. **Una vez termine la importación ya se puede iniciar la máquina virtual, seleccionando la máquina virtual, presionando start o doble click sobre la máquina virtual.**



Paso 4: Ejecutar la herramienta UI-Test.

1. Dentro se encontrará el siguiente material. La herramienta UI-Test se encuentra dentro de la carpeta APP_UI-TEST.





2. Para ejecutar la aplicación se debe dar doble click en el archivo .jar y se abrirá la aplicación.