Automated Testing Framework for Mobile Applications based in User-Interaction Features and Historical Bug Information

Abel Méndez-Porras y Jorge Alfaro-Velásco
Escuela de Computación
Instituto Tecnológico de Costa Rica
amendez@itcr.ac.cr, joalfaro@itcr.ac.cr

Marcelo Jenkins, Alexandra Martínez Porras y Abel Méndez-Porras
Escuela de Ciencias de la Computación e Informática
Universidad de Costa Rica
marcelo.jenkins@ecci.ucr.ac.cr,
alexandra.martinez@ecci.ucr.ac.cr

***Abstract*. Mobile applications support a set of user-interaction features that are independent of the application logic. These features include content presentation or navigation features. Rotating the device, gestures such as scroll or zoom into screens are some examples. In this paper an automated testing framework for mobile applications is proposed. Our framework integrates user-interaction features, historical bug information, and an interest point detector and descriptor to identify new bugs. A model of the application is automatically created and explored. While the exploration the model is performed we introduce user-interaction features and we capturing images. These images are passed to a bug analyzer to search bugs. The bug analyzer uses an interest point detector and descriptor to search for areas prone to bugs in the captured images. The use of historical bug information is proposed to determine sequences of events to better search bugs in applications. Preliminary results show that using the proposed technique is feasible to identify bugs in mobile applications.**

Keywords — automated software testing; mobile applications; user-interaction features; historical bug information and interest point detector and descriptor

I. INTRODUCCIÓN

Las aplicaciones móviles se han convertido en herramientas de trabajo. La portabilidad y el acceso a la información son características que favorecen esta adopción. Según Amalfitano y Fasolino [1] la calidad de las aplicaciones móviles es más baja de lo esperada debido a procesos de desarrollo rápidos, donde la actividad de pruebas de software se descuida o se lleva a cabo de una manera superficial. Las pruebas de software para aplicaciones móviles se consideran demasiado complejas y difíciles de automatizar, además son costosas en tiempo y esfuerzo. Sin embargo, las pruebas de software juegan un papel fundamental para desarrollar aplicaciones móviles confiables y con el menor número posible de defectos.

Según Wasserman [2] las áreas de aplicaciones móviles relacionadas con experiencias de usuario, requieren de mayor investigación para solventar las limitaciones actuales. Los gestos, sensores y datos de localización desempeñan un papel dominante en aplicaciones móviles. Garantizar que las aplicaciones móviles hacen el uso correcto de estos recursos, es una tarea compleja y poco desarrollada. Varios estudios indican la importancia del contexto en el que se ejecutan las aplicaciones móviles para garantizar su calidad [3, 4, 5]. La variabilidad de las condiciones de funcionamiento de una aplicación móvil depende de la posibilidad de utilizarla en contextos variables, donde un contexto representa el entorno general que la aplicación es capaz de percibir.

La incorporación de capacidades sensibles al contexto en aplicaciones móviles permite aprovechar la información contextual para proporcionar servicios adicionales. Sin embargo, debido a que estas capacidades introducen un nuevo espacio de entradas para la aplicación, pueden afectar el comportamiento de la misma en cualquier momento de su ejecución, por lo que su validación es compleja [4].

Los cortos periodos de desarrollo y la presión por lanzar al mercado las aplicaciones, se contraponen a los procesos de pruebas de software manuales, que típicamente son lentos y en especial cuando se deben probar plataformas móviles heterogéneas y fragmentadas. La fragmentación en dispositivos móviles se manifiesta en hardware y software [6, 7, 8, 9]. La fragmentación obliga a realizar pruebas de software en diferentes dispositivos. Una alternativa para afrontar esta problemática es el uso de herramientas automatizadas [10].

Las aplicaciones móviles soportan un conjunto de características de interacción del usuario que son independientes de la lógica de la aplicación. Estas características incluyen presentación de contenido o de navegación. Rotar el dispositivo o el uso de gestos para desplazarse o modificar el tamaño en las pantallas son ejemplos. Algunos defectos en aplicaciones móviles se pueden atribuir a las características de interacción del usuario [16].

978-1-4673-9143-6/15/$31.00

Evaluar de forma manual las aplicaciones móviles en busca de defectos atribuidos a características de interacción del usuario es una tarea compleja debido a la infinita posibilidad de combinaciones de pruebas de software requeridas. Una alternativa es buscar soluciones automatizadas para realizar las pruebas de software basadas en características de interacción del usuario.

En este trabajo se propone la creación de una herramienta para realizar pruebas de software automáticas en aplicaciones móviles para la plataforma Android. Dicha herramienta emplea un detector y descriptor de puntos de interés para ubicar posibles defectos por interacción del usuario. Además, plantea el uso de información histórica de defectos para mejorar la búsqueda de secuencias de eventos al explorar las aplicaciones. Los resultados preliminares de nuestra investigación confirman la pertinencia de la propuesta planteada. Las contribuciones de nuestro trabajo son:

- Combinar de forma novedosa el procesamiento digital de imágenes y especificaciones de la interfaz gráfica de la aplicación (GUI, por sus siglas en inglés), para detectar posibles defectos en aplicaciones móviles generados por la interacción del usuario.

- Utilizar información histórica de defectos para mejorar la selección de secuencias de eventos a utilizar como casos de prueba (para buscar defectos en las aplicaciones móviles).

El resto del documento está estructurado de la siguiente forma: en la sección 2 se analizan los trabajos relacionados; en la sección 3 se describe la propuesta para la creación de la herramienta de pruebas de software automáticas; en la sección 4 se presentan los primeros resultados obtenidos y en la sección 5 se presentan las conclusiones y trabajos futuros.

II. TRABAJOS RELACIONADOS

Las técnicas para pruebas de software automáticas para aplicaciones móviles han sido clasificadas en: (1) pruebas basadas en modelos, (2) pruebas basadas en aprendizaje del modelo, (3) pruebas sistemáticas, (4) pruebas difusas, (5) pruebas aleatorias y (6) pruebas basadas en script [11, 12, 13]. A continuación se describen varias herramientas por su pertinencia en el campo de pruebas de software para aplicaciones móviles.

Dynodroid [12] es un sistema que genera entradas para aplicaciones móviles en la plataforma Android. Utiliza el principio de "observar-seleccionar-ejecutar", el cual genera en forma eficiente una secuencia de entradas para las aplicaciones. Esta herramienta opera sobre los binarios sin modificar las aplicaciones, puede generar entradas de interfaz de usuario y entradas del sistema.

MobiGUITAR [14] es una herramienta para realizar pruebas de software totalmente automáticas a aplicaciones en la plataforma Android, con base en la interfaz gráfica de usuario. Esta herramienta observa, extrae y abstrae el estado en tiempo de ejecución de recursos de la interfaz gráfica de usuario. La abstracción se utiliza para crear un modelo de máquina de estados escalable, que junto con los criterios de cobertura de pruebas basados en eventos, proporciona una forma de generar automáticamente casos de prueba.

A3E [15] es una herramienta que permite explorar sistemáticamente aplicaciones en la plataforma Android mientras se ejecuta en los teléfonos reales. No necesita acceder al código fuente de la aplicación. Realiza un análisis estático del flujo de datos del bytecode de la aplicación y construye un grafo de flujo de control de alto nivel que captura las transiciones permitidas entre actividades. Utiliza este grafo para desarrollar una estrategia de exploración llamada "Targeted Exploration" que permite la exploración rápida y directa de las actividades, incluidas las que serían difíciles de alcanzar durante el uso normal de la aplicación. También han desarrollado una estrategia llamada "Depth-first Exploration" que imita las acciones de los usuarios para la exploración de las actividades y sus componentes de un modo más lento, pero más sistemático. Para medir la efectividad de las técnicas, utilizan dos indicadores: cobertura de actividades (número de pantallas exploradas) y cobertura de métodos.

SwiftHand [11] genera secuencias de entradas de pruebas de software para aplicaciones en la plataforma Android. Utiliza una técnica de aprendizaje de máquina para aprender un modelo de la aplicación durante las pruebas de software, utiliza el modelo aprendido para generar entradas del usuario para visitar estados inexplorados de la aplicación y utiliza la ejecución de la aplicación con las entradas generadas para refinar el modelo. Además, el algoritmo para pruebas de software evita reiniciar la aplicación durante las pruebas.

Ninguna de las anteriores propuestas se enfocan en características de interacción de usuario. Tampoco plantean el uso de información histórica de defectos, para mejorar la selección de secuencias de eventos, para buscar posibles defectos en las aplicaciones.

III. PROPUESTA DE HERRAMIENTA PARA PRUEBAS DE SOFTWARE AUTOMÁTICAS

Se plantea la creación de una herramienta para pruebas de software automáticas basada en características de interación del usuario. Además, se plantea el uso de información histórica de defectos para generar secuencias de eventos, que permitan en menor tiempo, exponer la mayor cantidad de defectos de las aplicaciones. En la figura 1 se muestra la visión general de la propuesta. En las siguientes secciones se describen cada uno de los componentes de la herramienta propuesta.

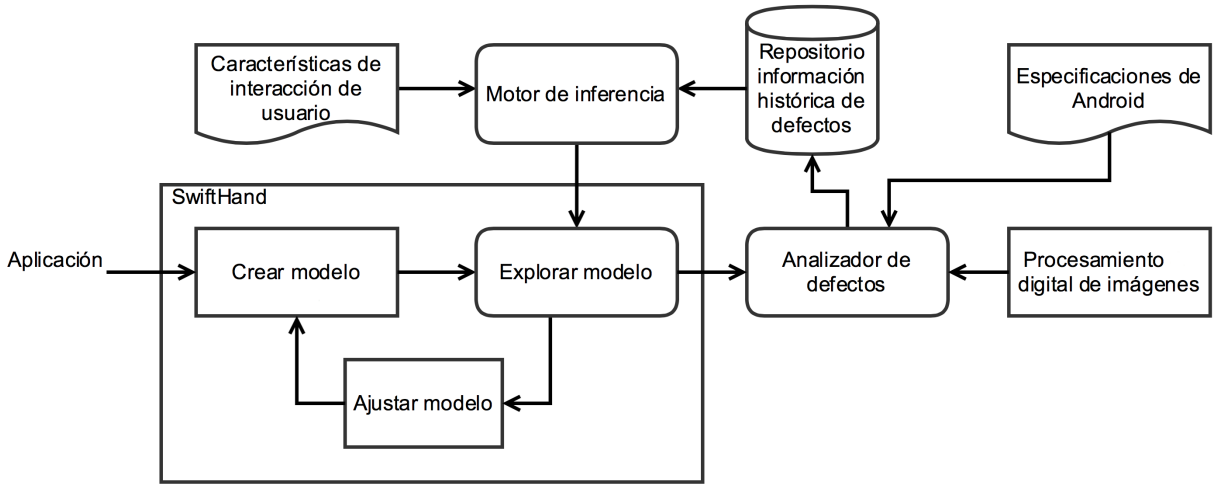

Figura 1. Visión general de la herramienta para pruebas de software automáticas.

A. Características de interacción de usuario

Las aplicaciones móviles permiten un conjunto de caractarísticas de interación del usuario que son independientes de la lógica de la aplicación. Tales características incluyen presentación o de navegación de contenido. Algunos ejemplos son girar el dispositivo (rotación horizontal o vertical del dipositivo) o el uso de diversos gestos para desplazarse o variar el tamaño en las pantallas. Zaeem et al. [16] han definido las siguientes 8 características de interacción de usuario:

- Doble rotación. Rotar un dispositivo móvil y luego retornarlo a su orientación original. Después de estas acciones la aplicación debería estar en el mismo estado que estaba antes de iniciar la acción de doble rotación.

- Finalizar y restaurar. El sistema operativo puede escoger finalizar y restaurar una aplicación por varias razones (por ejemplo, poca memoria). Después de esta acción la aplicación debería estar en su estado y vista inicial.

- Pausar y reanudar. La aplicación puede ser pausada (por ejemplo, por presionar el botón Home de Android) y reanudada. Después de estas acciones la aplicación debería estar en el mismo estado que estaba antes de iniciar la acción de pausar y reanudar.

- Funcionalidad del botón Atrás. El botón Atrás de hardware en los dispositivos Android que toma una aplicación y la lleva a la pantalla anterior.

- Abrir y cerrar menús. El botón Menú de hardware en los dispositivos Android abre y cierra los menús personalizados que cada aplicación define.

- Reducir tamaño de pantalla. Debería mostrar un subconjunto de lo que originalmente era en la pantalla.

- Aumentar tamaño de pantalla. Debería resultar en un superconjunto de la pantalla original.

- Desplazamiento. Desplazamiento hacia abajo (o hacia arriba) debe mostrar una pantalla que comparte partes de la pantalla anterior.

En esta propuesta se plantea una novedosa forma de realizar pruebas de software, para exponer defectos en aplicaciones móviles causados por las características de interacción del usuario descritas anteriormente.

B. Repositorio de información histórica de defectos

El Framework posee un repositorio para almacenar información histórica de defectos. La tabla 1 representa la forma en que es almacenada la información referente a los defectos reportados. Con esta información se puede determinar los eventos que se habían ejecutado previo al evento que originó el defecto.

El objetivo de almacenar la información histórica de defectos es para que en futuras pruebas de software pueda ser utilizada esa información por el motor de inferencia.

El motor de inferencia mediante aprendizaje de máquina consultará el repositorio de información histórica de defectos y hará inferencias sobre las características de interacción del usuario que aplicará en las pruebas de software y la forma en que aplicará dichas características.

TABLA 1. BASE DE DATOS DE DEFECTOS

Base de datos de defectos	
Ítem	***Descripción***
Sistema operativo	Versión
Secuencia de eventos	Secuencia de eventos que se estaba probando cuando apareció el defecto
Evento que generó el defecto	Descripción del evento que generó el defecto
Tipo de defecto	Clasificación automática del tipo de defecto
Tiempo transcurrido	Tiempo desde que se inició las pruebas hasta que el defecto fue encontrado
Contexto	Características del contexto y descripción de los recursos GUI que intervinieron en la generación del defecto

C. Motor de inferencia

El motor de inferencia es el encargado de seleccionar las características de interacción del usuario que se desean probar en las aplicaciones y el orden en que éstas serán probadas. Para la primera versión del Framework, se está empleando una selección aleatoria de las características de interacción de usuario que se desean probar en las aplicaciones.

En la figura 2 se muestra un diagrama de estados para una funcionalidad de inicio de sesión de una aplicación. Los estados que se encuentran entre líneas disconinuas (doble rotación, pausar y reanudar, botón atrás) representan las características de interacción del usuario que se desean ejecutar a la aplicación bajo prueba.

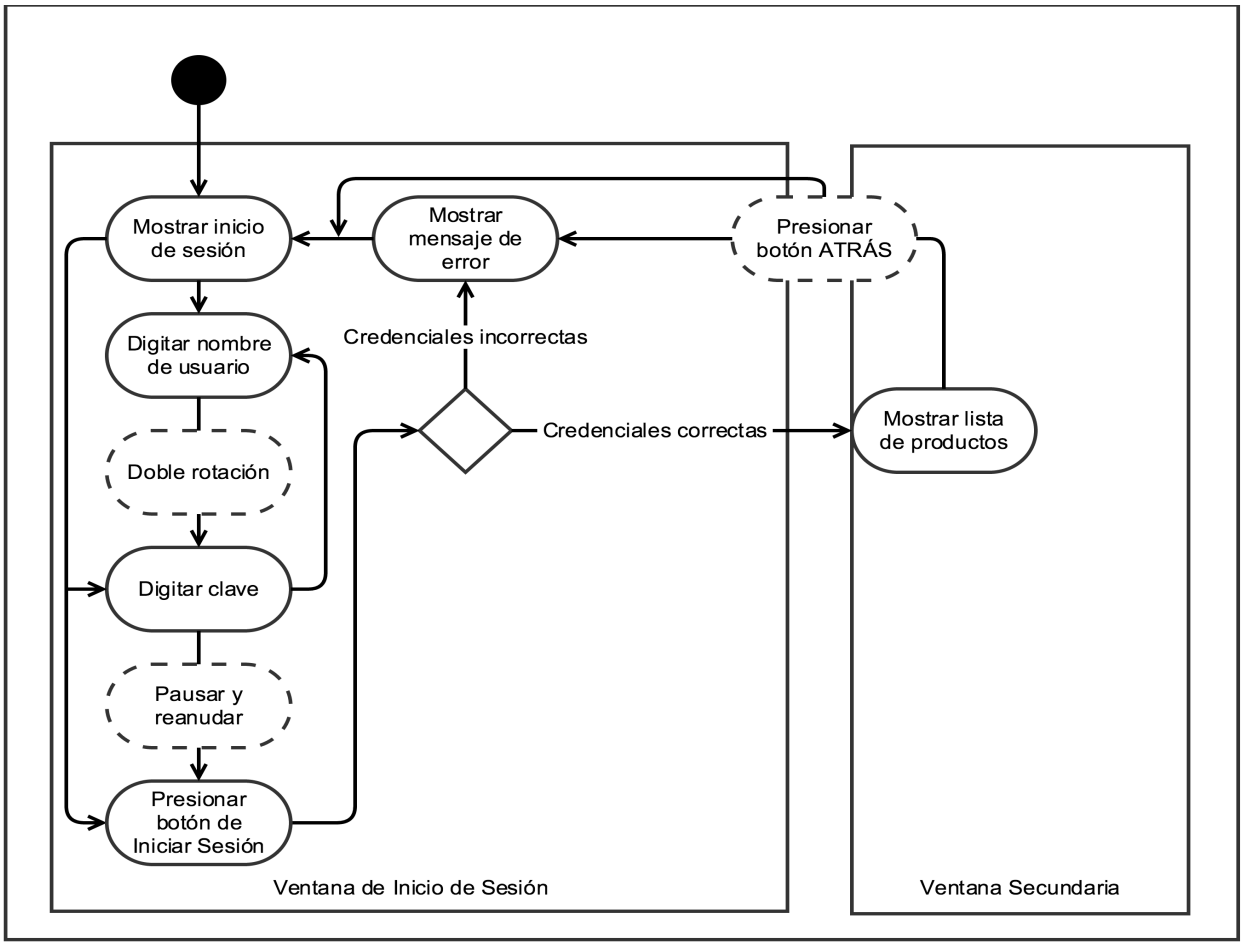

Figura 2. Ejemplo de inserción de características de interacción del usuario en un inicio sesión.

Se selecciona de forma arbitraría la siguiente secuencia de estados que se desea probar: digitar nombre de usuario, digitar clave y presionar el botón de inicio de sesión. Para pasar del estado "Digitar nombre de usuario" al estado "Digitar clave" se debe procesar el estado "Doble rotación". En este estado se realiza la acción de cambiar la orientación del dispositivo móvil y retornarlo a su orientación original para verificar si se generan defectos en la aplicación bajo prueba. Similar situación sucede cuando se desea pasar del estado "Digitar clave" al estado "Presionar botón de Iniciar Sesión". Antes de cambiar de estado se debe pausar y reanudar la aplicación bajo prueba para verificar si se generan defectos.

Una vez que se logre almacenar información histórica de defectos en el repositorio creado para tal fin, el motor de inferencia empleará aprendizaje de máquina para seleccionar las características de intereacción de usuario y el orden en que éstas serán probadas en las aplicaciones.

D. Analizador de defectos

El analizador de defectos se encarga de procesar la información que se genera cada vez que se desea procesar una característica de interacción del usuario en una aplicación bajo prueba. Para cada característica de interacción del usuario el proceso es el siguiente:

1. Se captura una imagen de la pantalla actual de la aplicación.
2. Se obtiene la información sobre la GUI de la pantalla actual.
3. Se ejecuta la característica de interacción del usuario que se desea probar. Por ejemplo, se ejecuta una doble rotación del dispositivo móvil.
4. Se captura una imagen de la pantalla actual después de haber ejecutado la característica de interacción del usuario que se deseaba probar.
5. Se captura la información de la GUI de la pantalla actual después de haber ejecutado la característica de interacción del usuario que se deseaba probar.
6. La información recolectada en los pasos anteriores es enviada al analizador de defectos.

Mediante la utilización de un detector y descriptor de puntos de interés se obtiene los puntos de interés de cada imagen. Con la ayuda de las especificaciones de la GUI de la plataforma Android, se determina si hay defectos al aplicar la característica de interacción del usuario. En caso de encontrar defectos, estos son almacenados en el repositorio de información histórica de defectos.

E. Especificaciones de la GUI de la aplicación

Las especificaciones de la GUI de la aplicación son utilizadas para crear un modelo de la actividad que está siendo explorada. El modelo es utilizado para identificar el o los recursos que se podrían ver afectados por la característica de interacción del usuario que está siendo aplicada.

F. Procesamiento digital de imágenes

La herramienta propuesta posee un módulo de procesamiento de imágenes para detectar y describir puntos de interés, que permitan detectar defectos en las aplicaciones móviles. Para ello se utiliza el algoritmo SURF (Speeded Up Robust Features) [17]. SURF es un detector y descriptor de puntos de interés invariante a la rotación y a la escala.

Los puntos de interés también conocidos como "key points" ó "features" son ubicaciones o puntos en una imagen susceptibles de caracterización y consecuente referencia, entre estos se encuentran esquinas, bordes e intercepciones. Deben cumplir diversos aspectos o condiciones:

1. Contar con una posición claramente definida dentro de la imagen.
2. Una definición matemática bien fundamentada que permita establecer la identificación del punto de interés como tal.
3. Mantenerse estables ante cambios o perturbaciones en la imagen, por ejemplo cambios en el brillo o iluminación, rotación, escala, entre otros.

Los puntos de interés son identificados mediantes descriptores, los cuales consisten en vectores de características calculados sobre cada uno de los puntos de interés. De esta manera un punto de interés puede contar con más de un descriptor.

Los puntos de interés son identificados en la imagen a través de círculos cuya dimensión está relacionada con el método MSER (Maximally Stable Extremal Regions) o mecanismo utilizado para establecer la región sobre la cual el descriptor podrá garantizar mayor solidez en términos de invarianza ante potenciales alteraciones en la imagen. Esto despeja la interrogante en términos del tamaño de los círculos en ciertos puntos de interés; un círculo con dimensiones amplias implica que el o los descriptores asociados necesitan dicha área para establecer una identificación adecuada del punto de interés.

G. SwiftHand modificado

La herramienta SwiftHand [11] permite crear, explorar y ajustar un modelo de la aplicación durante las pruebas de software. Nosotros estamos modificando SwiftHand para que permita la inclusión de características de interacción de usuario mientras explora el modelo de la aplicación. Además, estamos agregando la funcionalidad de capturar imágenes antes y despúes de la ejecución de las características de interacción del usuario. Estás imágenes serán enviadas al analizador de defectos para buscar posibles defectos en la aplicación.

IV. RESULTADOS PRELIMINARES

Se ha seleccionado la característica de interacción de usuario "Doble rotación", para mostrar los primeros resultados obtenidos con la propuesta planteada. Las acciones de rotar un

dispositivo móvil y luego volverlo a su orientación original debería dejar la aplicación en el mismo estado. Cambiar de orientación el dispositivo provoca que la actividad de la pantalla activa sea pausada y reanudada. Durante este proceso se pueden presentar algunos defectos con la fijación de datos a los diferentes recursos de la aplicación. Los campos de texto son propensos a perder los datos o modificarlos durante la doble rotación. Vamos a simular una modificación de datos sobre un campo de texto y cambio de color de los botónes mientras se realiza una doble rotación.

Se utiliza la captura de dos pantallas de la aplicación RotaTEC -aplicación desarrollada por los autores- para mostrar el ejemplo de doble rotación. La doble rotación aplicada consistió de los siguientes pasos:

- Estado inicial. Dispositivo en orientación vertical.
- Segundo estado. Dispositivo es cambiando a orientación horizontal.
- Tercer estado. Dispositivo es regresado a su estado inicial (orientación vertical).

En la figura 3 (a) se aprecia la captura de la pantalla antes de realizar la doble rotación. Los campos de texto tienen información personal de un usuario. En la figura 3 (b) se aprecia la captura de la pantalla después de realizar la doble rotación. Los campos de texto han perdido la información del usuario. Además, el color de los botones *Rotar* y *Salvar* ha cambiado. Los datos de ambas capturas de pantalla son diferentes. Visualmente, se puede apreciar los defectos que surgieron debido a la acción de doble rotación Automáticamente deben ser detectada estas diferencias y reportar los defectos de doble rotación.

La primera tarea realizada fue aplicar el algoritmo de detección de puntos de interés SURF a las imágenes capturadas antes y después de la doble rotación. Los circulos de color rojo en las imágenes de la figura 3(c) y la figura 3(d) representan los puntos de interés detectados por SURF. Se puede apreciar que la mayor cantidad de puntos de interés se centran donde hay letras o números.

En la tabla 2 se aprecia un resumen de los datos obtenidos al aplicar el algoritmo SURF. Se puede conocer la cantidad de puntos de interés antes de la doble rotación y después de la doble rotación. Además, el tiempo de extración de los puntos de interés. También la cantidad similar de puntos de interés entre ambas imágenes capturadas durante la doble rotación. Por último, el porcentaje de similitud entre la imagen capturada antes de la doble rotación y la imagen captura después de la doble rotación. Con esta información se puede decidir si se debe hacer una revisión más minusiosa de las imágenes, para buscar posibles defectos por la doble rotación.

TABLA 2. RESUMEN DE RESULTADOS DE LA DETECCIÓN Y DESCRIPCIÓN DE LOS PUNTOS DE INTERÉS

Información sobre los puntos de interés	
Ítem	***Valor***
Puntos de interés en figura 3 (c)	448
Puntos de interés en figura 3 (d)	464
Tiempo de extracción en milisegundos	226
Cantidad similar de puntos de interés	292
Porcentaje de similitud	64

El porcentaje de similitud entre imágenes de la figura 3(c) y la figura 3(d) es de 64,035%. Este valor hace suponer que deben existir diferencias importantes entre ambas imágenes y por lo tanto defectos a causa de la doble rotación. El enfoque ahora debe centrarse en los puntos de interés diferentes entre ambas imágenes. Los puntos de interés diferentes entre la imagen capturada antes de la doble rotación y la imagen captura después de la doble rotación son mostrados en figura 3(e) y la figura 3(f). Se puede apreciar claramente que los puntos de interés diferentes se concentran mayormente sobre los caracteres de los campos de texto y sobre los botones *Rotar* y *Salvar*. Teniendo ubicados los puntos de interés diferentes entre ambas imágenes, el siguiente paso es determinar que controles son los que se encuentran en esas coordenadas para identificar los posibles cambios. Para realizar estas tareas se emplea las especificaciones de GUI de la plataforma Android. Con las especificaciones de Android se logra determinar los objetos de tipo EditText y los objetos de tipo Button. Se analizan las propiedades de estos objetos para determinar si hay variaciones en sus configuraciones.

Una vez identificados los defectos, se procede a almacenar el reporte en el repositorio de información histórica de defectos. Se guarda la secuencia de eventos que se estaba ejecutando, el evento en que se presento el defecto, los controles que presentaron el problema y el tiempo transcurrido hasta que se originó el defecto. Esta información es utilizada por el motor de inferencia para determinar secuencias de eventos y el tipo de caracerísticas de interacción que se desea probar.

Para identificar defectos asociados a las restantes características de interacción de usuario, se debe emplear la misma técnica descrita en esta sección. Lo que varía es la captura de las imágenes antes y después de aplicar la caracterísitca de interacción del usuario. Las características de interacción de usuario que afectan el tamaño o realizan desplazamientos en la pantalla parecen ofrecer mayor dificultad para la técnica planteada es este trabajo.

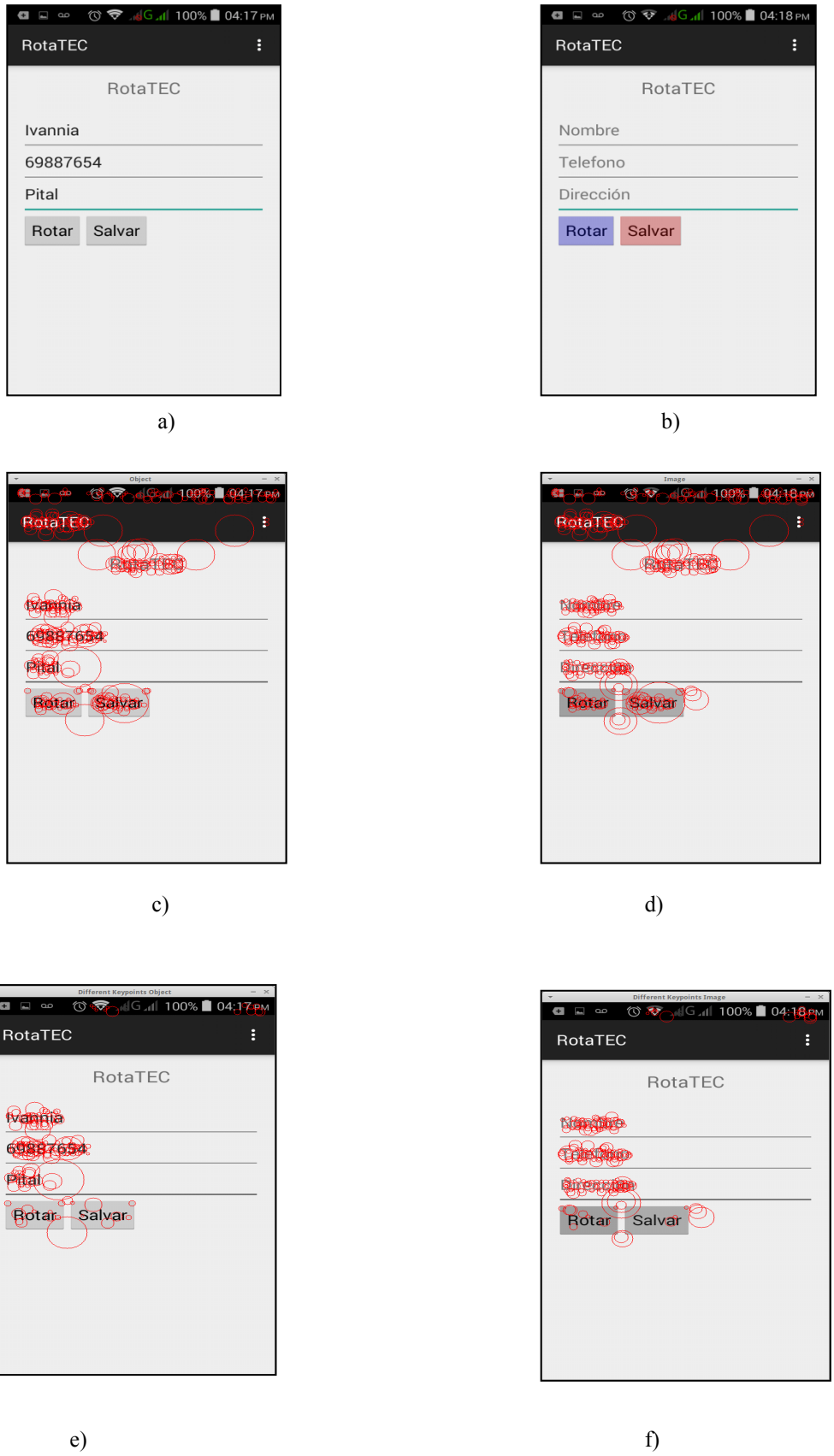

Figura 3. Ejemplo de la característica de interacción de usuario "doble rotación".

V. Conclusiones y trabajos futuros

En este artículo se propuso un novedoso Framework para la deteteción de defectos en aplicaciones móviles para la plataforma Android. El Framework propuesto está basado en características de interacción del usuario e información histórica de defectos. Mientras explora la aplicación móvil introduce características de interacción del usuario. Captura una imagen antes y otra imagen después de cada característica de interacción del usuario introducida. Utiliza el detector y descritor de puntos de interés SURF para comparar los puntos de interés entre las imágenes y basado en las diferencias entre los puntos de interés se determina las regiones donde se sospecha que hay defectos en la aplicación móvil.

La detección y descripción de puntos de interés en imágenes capturadas durante la interacción del usuario, apoyado con las especificaciones de la GUI, permitió la identificación de defectos de forma efectiva.

Una solución para detectar defectos atribuidos a características de interacción del usuario puede ser aplicada a cualquier aplicación móvil independiente del contexto o temática que ésta aborde. Por ejemplo, las 8 características de interacción del usuario abordadas se comportan igual en aplicaciones móviles para redes sociales que en aplicaciones móviles para análisis financiero. Otros tipos de gestos o eventos son específicos del contexto y temática de cada aplicación, no permiten ser generalizados y por esa razón no han sido considerados en este trabajo.

En trabajos futuros se incluirán otras características de interacción del usuario y del entorno para buscar otro tipo de defectos en las aplicaciones móviles. Además, se desea aumentar el repositorio de defectos reportados, para hacer experimentación con la generación de secuencias de eventos. El objetivo es minimizar la cantidad de secuencias de eventos que se deben generar y maximizar la cantidad de defectos encontrados.

La detección y descripción de puntos de interés es una tarea que introduce costo computacional. Para solventar esta problemática en el futuro utilizaremos recursos en la nube y paralelismo para la detección de los defectos.

Agradecimientos

Esta investigación es apoyada por el Ministerio de Ciencia, Tecnología y Telecomunicaciones (MICITT) de Costa Rica. Nuestro agradecimiento al Grupo de Ingeniería de Software Empírica de la Universidad de Costa Rica. A Giovanni Méndez Marín y Alberto Tablada Rojas, asistentes de este proyecto, gracias por su entrega y dedicación.

References

[1] D. Amalfitano, A. Fasolino and P. Tramontana, “A gui crawling-based technique for android mobile application testing,” in Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference, 2011, pp. 252–261.

[2] A.I. Wasserman, “Software engineering issues for mobile application development,” in Proc. FSE/SDP workshop on Future of software engineering research. FoSER ’10, New York, NY, 2010, pp. 397– 400.

[3] H. Muccini, A. Di Francesco and P. Esposito, “Software testing of mobile applications: Challenges and future research directions,” in Automation of Software Test (AST), 2012 7th International Workshop, 2012, pp. 29–35.

[4] Z. Wang, S. Elbaum and D. Rosenblum, “Automated generation of context-aware tests," in Software Engineering, ICSE 2007. 29th International Conference, 2007, pp. 406–415.

[5] D. Amalfitano et al., “Considering context events in event-based testing of mobile applications,” in Software Testing, IEEE 6th International Conference, Verification and Validation Workshops, 2013, pp. 126–133.

[6] H. Ham and Y. Park, “Mobile application compatibility test system design for android fragmentation,” Communications in Computer and Information Science 257 CCIS, 2011, pp. 314–320.

[7] Z. Liu, X. Gao and X. Long, “Adaptive random testing of mobile application,” in Computer Engineering and Technology (ICCET), 2010 2nd International Conference, vol 2, 2010.

[8] J. Kaasila et al., “Testdroid: Automated remote ui testing on android,” in Proc. 11th International Conference on Mobile and Ubiquitous Multimedia, MUM 2012, 2012.

[9] L. Lu et al., “Activity page based functional test automation for android application,” 2012 Third World Congress on Software Engineering, 2012, pp. 37–40.

[10] W. Yang, M.R. Prasad and T. Xie, “A grey-box approach for automated gui-model generation of mobile applications,” in Fundamental Approaches to Software Engineering, 16th International Conference, Berlin, Heidelberg, Springer-Verlag, 2013, pp. 250–265.

[11] W. Choi, G. Necula and K. Sen, “Guided gui testing of android apps with minimal restart and approximate learning,” in ACM SIGPLAN Notices, vol. 48(10), 2013, pp. 623–639.

[12] A. Machiry, R. Tahiliani and M. Naik, “Dynodroid: An input generation system for android apps,” in Proc. 2013 9th Joint Meeting on Foundations of Software Engineering, 2013, pp. 224–234.

[13] B. Nguyen et al., “Guitar: an innovative tool for automated testing of gui-driven software,” Automated Software Engineering, 2013, pp. 1–41.

[14] D. Amalfitano et al., “Mobiguitar – a tool for automated model-based testing of mobile apps,” in Software, IEEE PP(99), 2014.

[15] T. Azim and I. Neamtiu, “Targeted and depth-first exploration for systematic testing of android apps,” ACM SIGPLAN Notices, vol. 48(10), 2013, pp. 641–660.

[16] R. Zaeem, M. Prasad and S. Khurshid, “Automated generation of oracles for testing user-interaction features of mobile apps,” in Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference, 2014, pp. 183–192.

[17] H, Bay, T. Tuytelaars and L. Van Gool, “Surf: Speeded up robust features,” in Computer Vision, 9th European Conference on Computer Vision, vol. 3951, 2006, pp. 404–417.