PRUEBAS DE SOFTWARE PARA DISPOSITIVOS MÓVILES ANDROID

Gloriana Araya Solís, gloriana.a.s@gmail.com

Geovanny Méndez Marín guikane@gmail.com

Ronald Jiménez Segura ronalditcr@gmail.com

Instituto Tecnológico de Costa Rica Sede San Carlos

Carrera de Ingeniería de Computación Santa Clara, San Carlos, Costa Rica, julio 2014

Resumen

Los dispositivos móviles forman parte de nuestro diario vivir. Y con ello, cada día en el mercado hay nuevas aplicaciones de todo tipo y con diferentes funcionalidades. Con estas vienen problemas de funcionamiento derivados de un mal desarrollo o mala comprensión de los requisitos. Estas aplicaciones pueden generar errores de compatibilidad debido a los distintos tipos de dispositivos, sistemas operativos y versiones.

En el siguiente artículo se dará una breve descripción del trabajo realizado durante año y medio de investigación en pruebas de software en diferentes aplicaciones para dispositivos móviles android, en el cual se dará información de los procesos de aseguramiento y control de calidad de software con su debida descripción de un plan de pruebas, además la realización de pruebas con herramientas novedosas en el mercado de automatización.

Palabras Clave: Calidad de software, Pruebas de software, Automatización.

Introducción

Generalmente las empresas de software que han alcanzado cierto grado de madurez, tienen un departamento de QA (Quality assurance) conformado por ingenieros de la calidad, que deben integrarse con los equipos de desarrollo, e incluso con los clientes de la empresa. Actualmente, los mismos desarrolladores deben realizar pruebas de diferentes tipos sobre las aplicaciones que están desarrollando, por lo que no solo el departamento de QA lleva a cabo este proceso, sino los

mismos grupos de desarrollo han empezado a utilizar TDD (Test-driven development) que profundizaremos más adelante.

Uno de los principios básicos en el desarrollo de proyectos es que la calidad nunca es negociable, por lo que el aseguramiento y control de la calidad es muy importante en el desarrollo de software, para garantizar que se va a obtener como resultado final un software de calidad.

Es importante que las empresas alcancen un proceso confiable de aseguramiento de la calidad de sus productos, debido a que un software de buena calidad genera confiabilidad en los clientes. Si no se cuenta con un proceso de QA riguroso, se corre el riesgo de enfrentar el rechazo de los clientes, quienes probablemente usarán los canales de información principales para ventilar comentarios negativos acerca de los productos deficientes.

En QA son llevadas a cabo diferentes pruebas, entre las cuales se puede dividir por tipo (funcional y no funcional), por ejecuciones (automáticas y manuales) y por técnica (caja blanca y caja negra). Estas pruebas son llevadas a cabo con una guía conocida como test-plan (plan de pruebas) en el cual se elabora las normas de calidad que debe de tener el proyecto antes de la entrega al cliente.

Lo más adecuado que se debe de hacer en un proyecto de software es tener a un grupo de profesionales especializados en llevar a cabo las pruebas con técnicas que permitan alcanzar cierto grado de cobertura.

Esto debido a que nunca se debe de permitir que los desarrolladores realicen sus propias pruebas por el efecto psicológico que provoca el hecho de probar algo que ya el desarrollador sabe cómo funciona.

Existen herramientas de última generación, para aumentar la productividad en QA, entre las que se encuentran las herramientas de automatización. Estas herramientas permiten realizan una mayor cantidad de pruebas en un menor tiempo y evitando los errores humanos. Esto ha provocado que el enfoque del QA actual esté basado en las pruebas de automatización para mejorar el desempeño de los procesos y reducir el tiempo de liberación de la aplicación.

En el tiempo de investigación en el que ha trabajo el grupo con diferentes

herramientas de automatización podemos asegurar que la productividad a gran escala se ve mejorada.

En este documento no serán mostrados los resultados obtenidos de los trabajamos que se han realizado pero si se espera dar una introducción a los lectores sobre la forma de utilización de las herramientas de automatización y las ventajas de cada una de las herramientas; como también se pretende que el lector conozca la teoría básica del QA .

El proceso de investigación aun es vigente por lo que se mostraran algunas de las herramientas con las que estamos trabajando actualmente, la cuales no están estables pero son buenas propuestas para el futuro de la automatización de pruebas de software para dispositivos móviles.

Objetivos

- Definir y entender la teoría que envuelve la calidad de software.
- Determinar los tipos de pruebas de software.
- Definir las técnicas de pruebas de software.
- Exponer la guía de elaboración de un caso de pruebas.
- Exponer los lineamientos de un plan de pruebas.
- Mostrar las herramientas de automatización de software.
- Enseñar los procesos de ejecución para ejecutar pruebas con herramientas de automatización.
- Mostrar las herramientas actuales en desarrollo para el futuro de las pruebas de automatización.

Marco Teórico

El objetivo de esta ponencia es mostrar la información recopilada durante año y medio de investigación en los procesos de calidad de software con las herramientas de automatización para dispositivos móviles con que contamos actualmente para mejorar estos procesos.

Esta investigación se inició con la comprensión de la teoría que envuelve la calidad del software, en donde para iniciar a comprender la teoría es importante definir las diferencia entre aseguramiento de la calidad y control de calidad. El aseguramiento de calidad son un conjunto de actividades planificadas y sistemáticas que se deben de realizar antes de iniciar el desarrollo del software para garantizar la calidad en el software, mientras que

control de calidad son un conjunto de actividades a nivel operativo que garantizan tener bajo control un proceso de software.

Metodología de trabajo

Para evitar la insatisfacción del cliente y brindar un producto de calidad se deben de realizar pruebas de software, las cuales se definen como una actividad que ejecuta circunstancias previamente especificadas, donde los resultados se observan y registran para realizar una evaluación.

En los proyectos de desarrollo de software las fallas son tan variadas que se ha tenido agrupar las pruebas a aplicar en diferentes conjuntos. En donde primeramente se debe de clasificar por tipo:

No Funcionales:

Éstas se realizan para verificar que el software desarrollado cumple con los requerimientos no funcionales establecidos por el cliente.

Funcionales:

Las pruebas funcionales que se enfocan en la ejecución, revisión y retroalimentación de la aplicación. Este tipo de pruebas se aseguran de que funcionen los requisitos funcionales.

Según la forma de aplicación de las pruebas se puede dividir en:

Pruebas estáticas:

Son tipos de pruebas que se realizan sin ejecutar la aplicación, por lo cual se hace una revisión de código para verificar que cumpla con una serie de condiciones que indiquen que el código es de calidad. Las herramientas más reconocidas para este tipo de pruebas son:

- Checklist: Consiste en una herramienta que permite verificar si el código Java que generamos, cuenta con las convenciones.
- Findbugs: Es una herramienta que realiza un chequeo del código y muestra los posibles bugs por su tipo.

Pruebas dinámicas:

Las pruebas dinámicas son aquellas pruebas que no se pueden ejecutar hasta el momento en que esté disponible una versión ejecutable del software. Sin embargo, la preparación de las pruebas dinámicas puede empezar mucho antes, con el fin de comenzar con su ejecución al recibir la primera versión ejecutable del software.

Es importante mencionar que durante los procesos de desarrollo se realizan diferentes tipos de pruebas según la etapa del proyecto donde se encuentra, por lo que el marco de testeo debe de estar claro con las debidas pruebas a aplicar en la etapa de planificación de un proyecto. Uno de los modelos más utilizado es el modelo V (en cual se muestra en la Figura 1), donde el desarrollo y el testing son dos ramas que apuntan a los mismos niveles, ya que para cada nivel de desarrollo existe su correspondiente nivel de Testing.

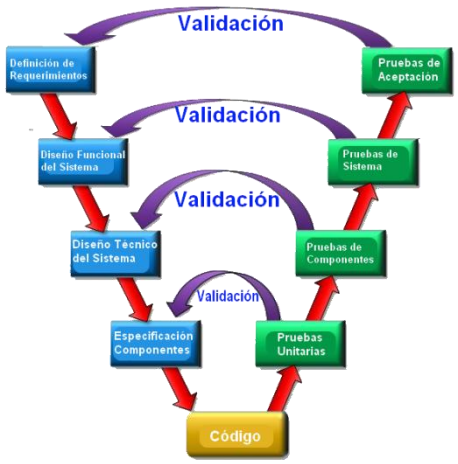

Figura 1. Ejemplo de modelo V

También se pueden clasificar por la técnica de aplicaciones de pruebas, donde se catalogan en:

Pruebas de Caja Blanca:

Son las pruebas que se realizan a lo interno de un módulo enfocándose en se los detalles procedimentales del software.

Pruebas de Caja Negra:

Las pruebas de caja negra se centran en los resultados, donde se manejan la entrada a un módulo y se da un valor esperado para la salida sin tener en cuenta el funcionamiento interno.

Pruebas de Caja Gris:

Se basan en la realización de testing de caja negra basada en casos de prueba realizados por personas que conocen el código de la aplicación . Se entiende que de esta forma las pruebas realizadas son más efectivas porque se conocen las partes del código que pueden resultar más conflictivas.

Por otra parte el test incremental es de importancia en los procesos de calidad y se divide en 2 partes:

TopDown:

Este enfoque de prueba se lleva a cabo desde el módulo principal al sub módulo. Si no se ha desarrollado el sub-módulo de un programa temporal se crea un STUB, el cual se utiliza para simular el sub-módulo.

ButtomUp:

Este enfoque de prueba se lleva a cabo a partir de sub-módulo a

módulo principal, si no se desarrolla el módulo principal, un programa temporal denominado controlador se utiliza para simular el módulo principal.

Como anteriormente mencionamos las pruebas de automatización se ha divido según el tipo de problemas que buscan, por lo que se ha tenido que generar una gran cantidad de tipos pruebas para solventar los posibles problemas que se presentan en una aplicación. La siguiente tabla muestra la descripción de las tipos de pruebas que se pueden realizar a un proyecto.

Nombre de la prueba	Descripción
Pruebas Unitarias	Se focaliza en ejecutar cada módulo, lo que provee un mejor modo de manejar la integración de las unidades en componentes mayores.
Prueba de Integración	Identificar errores introducidos por la combinación de programas probados unitariamente.
Prueba de Regresión	Determinar si los cambios recientes en una parte de la aplicación tienen efecto adverso en otras partes.
Pruebas de Humo	Detectar los errores en realeases tempranos y de manera fácil probando el sistema constantemente. Con lo que se aseguran los resultados de las pruebas unitarias y se reducen los riesgos.
Pruebas del Sistema	Asegurar la apropiada navegación dentro del sistema, ingreso de datos, procesamiento y recuperación.
Pruebas de Desempeño	Las pruebas de desempeño miden tiempos de respuesta, índices de procesamiento de transacciones y otros requisitos sensibles al tiempo
Pruebas de Carga	Verificar el tiempo de respuesta del sistema para transacciones o casos de uso de negocios, bajo diferentes condiciones de carga.
Pruebas de Stress	Verificar que el sistema funciona apropiadamente y sin errores, con memoria baja o no disponible en el servidor o con un número máximo número de clientes conectados. También con múltiples usuarios desempeñando la misma transacción con los mismos datos.
Pruebas de Volumen	Verificar que el sistema trabaja bien con un máximo número de clientes conectados y todos ejecutando la misma función por un período extendido. También verificar con un tamaño máximo de base de datos y múltiples consultas ejecutadas simultáneamente
Pruebas de tolerancia	Verificar que los procesos de recuperación restauran apropiadamente la Base de datos, aplicaciones y sistemas, y los llevan a un estado conocido o deseado.
Prueba de Múltiples Sitios	Detectar fallas en configuraciones y comunicaciones de datos entre múltiples sitios.
Prueba de Compatibilidad y Conversión	Buscar problemas de compatibilidad y conversión en los sistemas.
Pruebas de Integridad de Datos y Base de Datos	Asegurar que los métodos de acceso y procesos funcionan adecuadamente y sin ocasionar corrupción de datos.
Pruebas de Seguridad y Control de Acceso	Nivel de seguridad de la aplicación: Verifica que un actor solo pueda acceder a las funciones y datos que su usuario tiene

	permitido.
Pruebas del Ciclo del Negocio	Asegurar que el sistema funciona de acuerdo con el modelo de negocios emulando todos los eventos en el tiempo y en función del tiempo.
Pruebas de GUI	Verificar la navegación a través de los objetos de la prueba reflejando las funcionalidades del negocio y requisitos, se realiza una navegación ventana por ventana, usando los modos de acceso (tabuladores, movimientos del mouse, teclas rápidas, etc).
Pruebas de Configuración	Validar y verificar que el cliente del sistema funciona apropiadamente en las estaciones de trabajo recomendadas.
Prueba de Estilo	Comprobar que la aplicación sigue los estándares de estilo propios del cliente.
Prueba de Instalación	Verificar y validar que el sistema se instala apropiadamente en cada cliente con instalaciones nuevas y actualizaciones
Prueba de Campo	Correr el sistema en el ambiente real para encontrar errores y validar el producto contra sus especificaciones originales.
Pruebas Beta	Realizar la validación del sistema por parte del usuario.

Análisis y resultados:

A pesar de que se dice que el QA está en pañales la automatización ha estado generado un gran impacto en los procesos de calidad. Esto debido a que las pruebas manuales son propensas a errores y los procesos de ejecución son lentos.

Se recomienda enfocar la automatización para pruebas unitarias, pruebas funcionales y pruebas de interfaz de usuario.

Estas pruebas son realizadas sobre aplicaciones dirigidas por eventos (Event-Driven) que son un subconjunto de los Sistemas Reactivos. Dichas aplicaciones reciben eventos tanto por parte del usuario, como de la plataforma, y sus sensores.

Los componentes que conforman la estructura de la GUI de este tipo de aplicaciones, aceptan secuencias de eventos de usuario, que alteran el estado de la aplicación. Esta alteración puede o no, incluir un cambio que altera a la GUI.

Las pruebas para el software con GUI, consisten en la ejecución de eventos pertenecientes a esos componentes y el monitorio del resultado al cambiar el estado del programa.

Los casos de prueba para aplicaciones con interfaces graficas de usuario tienen secuencias de eventos tales como inputs y alguna indicación del estado del programa (Estado de la GUI, Estado de la Memoria, Log de errores, u otros indicadores en tiempo de ejecución del estado de la aplicación) como el output esperado.

En muchos casos (a menos que una aplicación incluya dos tipos de interfaces, con GUI y sin GUI) las pruebas de GUI (el GUI Testing) es la única forma posible de realizar pruebas a nivel de sistema para este tipo de aplicaciones. Esto hace que el testing de este tipo sea una parte critica de las prueba para cualquier software GUI.

El tamaño y complejidad de las GUI modernas, en términos de los componentes GUI, y los eventos que deben ser ejecutados en ellos, exceden los límites prácticos de la exhaustiva y analítica aproximación para el testing.

El número de posibles casos de test para una GUI incrementa exponencialmente dependiendo del número de eventos por caso de prueba.

Desafortunadamente, los testers no pueden ignorar secuencias de eventos largas y complejas, debido a que esas secuencias frecuentemente revelan bugs de estados específicos.
En la práctica, las herramientas para el GUI testing usan dos aproximaciones principales.

1. Emplear un lenguaje basado en scripts. Ejemplos son:
   - JFCUnit
   - Selenium Web-Driver
   - Robotium
   - Abbot
   - SOAtest

Este lenguaje se utilizaría para crear manualmente unit test cases para GUIs.

Un unit test case, o test de unidad, consiste de llamadas a métodos que programáticamente invocan eventos de la GUI.

Los tests graban y verifican el output usando afirmaciones tester-specified.

2. Ya que la codificación manual de casos de test puede ser tediosa, una

alternativa se llama capture/replay es utilizada.

Ejemplos de herramientas que realizan capture/replay son

- Test Automation FX
- Selenium IDE
- Quick Test Pro

La herramienta primero captura secuencias de eventos que los testers realizan manualmente en la GUI.

Todo esto proceso implica que es necesario utilizar herramientas que permitan automatizar la mayor cantidad de procedimientos posibles. Aunque cuando hoy en dia no sea posible automatizar todas las etapas del testing, hay una buena parte, sobre todo la parte combinatoria en la que recae la buena generación de casos de prueba, con la que los equipos de QA se pueden apoyar.

Entre las herramientas de automatización que se han investigado en este proyecto están AndroidGUITAR[X], A^3E [X], ATG[X] y Orbit.

En el caso de ATG y Orbit, no fue posible obtener la aplicación para realizar las pruebas, pero se realizaron las lecturas de sus artículos de investigación, en donde se explicaba su funcionamiento. Por lo que solo fueron probadas AndroidGUITAR y A^3E, que seran explicadas a continuacion.

GUITAR[X] es un framework de automatización que cumple las siguientes características

- Tiene una arquitectura basada en plugins.
- Automatización de las actividades claves para el GUI Testing.
- Permite modificar los algoritmos para adaptarlo a diferentes plataformas
- Basado de forma nativa en modelos (Pero se pueden usar otro tipo de estructuras).

Esta herramienta esta compuesta por 4 módulos:

1. **GUI RIPPER:** Se encarga de crear el archivo .GUI
2. **MODEL CONSTRUCTOR:** Genera un EFG a partir del .GUI
3. **TEST CASE GENERATOR:** Genera los casos de testing
4. **REPLAYER:** Realiza el testing de los casos de prueba generados

El EFG (Event-Flow Graph) es una estructura de datos generada por la aplicación.

El GUI-Ripper hace una revisión del ejecutable, y comienza a ejecutar eventos sobre todos los Widgets, y a partir de ello, extrae información relativa con los siguientes elementos

- Event Handlers
- Listeners
- Events

Esta información es usada para crear un archivo de estructura, que luego debe ser transformado en un modelo que simplifique de manera eficiente esa estructura, la cual es el EFG

Prácticamente todas las etapas con GUITAR se pueden automatizar hasta cierto punto.

Tiene algunas funcionalidades que el grupo de este proyecto considera bastante relevantes, tal como la posibilidad de especificar oráculos, y el uso de credenciales.

Por su parte, A^3E (Automatic Android App Explorer) es una herramienta para generar casos de prueba, mediante una aplicación de instrumentación, que se instala junto con la AUT, en un dispositivo real.

Dado a que A^3E no requiere instrumentación a nivel de kernel o framework, es posible realizar las pruebas sobre dispositivos reales, lo cual presenta una ventaja para el desarrollador.

A diferencia de GUITAR, Orbit y ATG, A^3E no sirve para ejecutar las pruebas, sino que solamente permite generarlas de forma automática. Se pueden tomar estas pruebas y completar un test suite.

Las estructuras de datos son generadas recorriendo la aplicación en tiempo real, con técnicas de exploración dirigida y en profundidad.

La técnica “primero en profundidad” fue desarrollada para simular las acciones del usuario, explorando las actividades y los elementos de la GUI de forma similar a como lo haría un usuario (haciendo tap sobre los objetos gráficos, editando los cuadros de texto, o entrando a nuevas actividades y luego retornando a las previas con el boton back, entre otras.). Tiene un funcionamiento lento, debido a su

comportamiento mayormente sistemático.

La técnica “Exploracion dirigida” realiza un análisis estático del bytecode para extraer una estructura de datos en forma de grafo disconexo llamada SATG (Static Activity Transition Graph) para lugeo explorarla sistemáticamente mientras la aplicación está siendo ejecutada. Se encarga de enlistar todas las actividades que puedan ser llamadas desde otras aplicaciones o desde servicios en segundo plano sin intervención del usuario. Además, genera llamadas para invocar esas actividades directamente. Esta estrategia es requerida debido a que no todas las actividades son invocadas a través de la interacción del usuario, por lo que el análisis en profundidad no las lograría alcanzar.

Para esta tarea también es generada otra estructura llamada DATG, encargada de capturar la interacción del usuario con la aplicación en tiempo de ejecución (la cual es simulada de manera automática).

La construcción del SATG puede se reducida a un problema de taint-tracking, porque, por ejemplo, si la actividad actual es A y una transición a la Actividad B es posible como resultado de la interacción del usuario, los métodos asociados con A no invocaran directamente a B. En vez de eso, la transición esta basada en un Intent genérico en el cual se pasa la lógica. Consecuentemente la construcción del SATG puede ser alcanzada a través de un análisis de flujos de datos, más específicamente realizando un seguimiento por marcas.

Conclusiones:

En un mercado abarrotado por aplicaciones, donde el usuario final tiene una gran cantidad de opciones que prometen cumplen su expectativa y sus requerimientos es importante, si se desea asegurar el éxito de nuestro producto, que se asegura ofrecer algo de calidad que haya sido probado con anterioridad. Con el paso del tiempo, el avance en el desarrollo tecnológico, y el acceso cada vez mayor que se tiene a diferentes tecnologías especialmente en el área móvil, los usuarios se han vuelto cada vez más exigentes y desarrollan un mayor criterio al momento de evaluar las mismas en la

App Store donde se publican, estos mismos se han vuelto aún más expertos y se guían por los comentarios y evaluación de otros usuarios; es por esto que antes de publicar una aplicación se debe estar seguros de cumplir con la calidad necesaria ya que también se juega la reputación de la empresa que se representa. En este artículo el grupo presenta las diferentes herramientas en las que se ha trabajado, obteniendo resultados positivos, tanto en la evaluación del código como tal, con las herramientas CheckList y FindBucks, que permite verificar que en el desarrollo del código se cumplan estándares y convenciones de código de Java, y por otro lado, se trabajó con herramientas que permiten evaluar la funcionalidad del proyecto, tanto los aspectos no funcionales como los funcionales, que se integran y permiten de generar una buena calidad en la aplicación. El grupo ha presentado los resultados exitosos de una investigación realizada, se sabe que el tema de Q.A es algo relativamente nuevo aplicado a la tecnología móvil, pero que se ha ido desarrollando rápidamente con un gran futuro, además que con el transcurso del tiempo aparecerán nuevas aplicaciones que cumplan funcionalidades diferentes o vengan a complementar o mejorar las que se han mencionado, por lo tanto para el grupo de investigación es una motivación presentar exitosamente los resultados que se obtuvieron, y se tiene como fin continuar posteriormente la investigación ampliando los conocimientos que cada miembro ha generado.

Bibliografía:

[1].http://www.slideshare.net/hally20191/casos-de-pruebas

[2].http://www.calidadysoftware.com/testing/pruebas_funcionales.php