6Д

УДК 681.3

О.В. Дерев'янченко, В.О. Плакидюк
Київський національний університет імені Тараса Шевченка, Україна
Україна, 01601, м. Київ, вул. Володимирська, 60

Система автоматичного тестування програм під ОС Android

O.V. Derevyanchenko, V.O. Plakydiuk
Taras Shevchenko National University of Kyiv, Ukraine
60, Volodymyrska Street, City of Kyiv, Ukraine, 01601

System of the Android Applications Testing

А.В. Деревянченко, В.О. Плакидюк
Киевский национальный университет имени Тараса Шевченко, Украина
Украина, 01601, г. Киев, ул. Владимирская, 60

Система автоматического тестирования программ под ОС Android

У статті розглядається розробка власного фреймворку автоматизованого тестування програмного інтерфейсу користувача під систему Android. Аналізуються існуючі підходи до автоматизованого тестування програм в інших системах та пропонується метод для розробки власного автоматизованого фреймворку для Android. Отримані результати дають можливість швидко створювати автоматизовані тести програмного інтерфейсу.
Ключові слова: автоматизоване тестування, тестування інтерфейсу користувача, фреймворк для тестування, Android.

The article discusses development of a self-made software framework for automated user interface testing for the Android OS. Analyzing existing approaches to the automated testing of applications in other systems, as well as proposing a method to develop one's own automated framework for Android. These results make it possible to easily create automated tests of application interface.
Key words: automated testing, UI testing, testing framework, Android.

В статье рассматривается создание собственного фреймворка автоматизированного тестирования программного интерфейса пользователя под систему Android. Анализируются существующие подходы к автоматизированному тестированию программ в разных системах и предлагается метод для разработки собственного автоматизированного фреймворка для Android. Полученные результаты дают возможность быстро создать автоматизированные тесты программного интерфейса.
Ключевые слова: автоматизированное тестирование, тестирование интерфейса пользователя, фреймворк для тестирования, Android.

У процесі взаємодії людина – комп'ютер, шукаючи інформацію або даючи команди з використанням елементів графічного інтерфейсу, можна встановлювати прямий візуальний контакт з ними. Важливим для цього процесу є автоматизація. Скрипти та макроси, які контролюють елементи графічного інтерфейсу, або звертаються до елементу по імені, яке може бути незнайомим або навіть недоступним для користувача, або місцем на екрані, яке може змінюватися.

Ця робота присвячена питанням автоматизованого тестування [1] користувацького інтерфейсу, а саме швидкої розробки автоматизованих тестів під платформу Android.

Розроблений фреймоворк дозволяє набагато швидше розробляти автоматизовані тести, використовуючи додатковий рівень абстракції, що дозволяє користувачам або програмістам встановлювати зв'язок об'єктів з елементами графічного інтерфейсу. Для пошуку документальних даних про елементи графічного інтерфейсу користувач може попередньо створити об'єктно-орієнтовану модель графічного інтерфейсу, після чого швидко збільшувати кількість автоматизованих тестів, використовуючи її. Аналогічним чином для автоматизації взаємодій з елементом графічного інтерфейсу програміст може використовувати вже існуючий інструмент розробки автоматизованих тестів під платформу Android – UiAutomator. Що є більш низьким рівнем розробки тестів.

Розробка фреймворку автоматизованого тестування мотивована відсутністю ефективного за швидкістю механізму створення тестів графічного інтерфейсу. Можливість швидкої побудови об'єктної моделі для довільних елементів графічного інтерфейсу має вирішальне значення, для програмістів, що створюють автоматизовані тести.

Сучасні підходи використання ОС Android [2-4] вимагають написання тестів з використанням інструментарію UiAutomator. Тобто при ручному заданні елементів графічного інтерфейсу, без необхідного рівня абстракції від конкретної реалізації. Адже розробка програмного продукту під платформу Android характеризується великою кількістю різних розмірів екранів мобільних пристроїв, що робить ручний спосіб написання тестів надто громіздким.

У роботі ми вирішували задачу розробки фреймворку, який у свою чергу, дозволить створити певний рівень абстракції між інструментом UiAutomator та графічним інтерфейсом програмного додатку під ОС Android. Система має представляти графічні елементи як певні об'єкти та надавати тестувальнику можливість швидко та якісно проводити тести і мати можливість легко їх змінювати.

Засоби автоматизованого тестування

Тестування – один з найважливіших етапів контролю якості в процесі розробки програмного забезпечення.

Автоматизоване тестування є його складовою частиною. Воно використовує програмні засоби для виконання тестів і перевірки результатів виконання, що допомагає скоротити час тестування і спростити його процес.

Основні підходи до автоматизації тестування

Існує два основних підходи [1] до автоматизації тестування: тестування на рівні коду та тестування користувацького інтерфейсу (GUI-тестування). До першого типу належить, зокрема, модульне тестування. До другого – імітація дій користувача за допомогою спеціальних тестових фреймворків.

Найбільш поширеною формою автоматизації є тестування додатків через графічний користувальницький інтерфейс. В роботі ми розглянемо один з таких підходів. Популярність такого виду тестування пояснюється двома факторами: по-перше, додаток тестується тим же способом, яким його буде використовувати людина, по-друге, можна тестувати додаток, не маючи при цьому доступу до вихідного коду.

Однією з головних проблем автоматизованого тестування є його трудомісткість, незважаючи на те, що воно дозволяє усунути частину рутинних операцій і прискорити виконання тестів, великі ресурси можуть витрачатися на оновлення самих тестів. Це відноситься до обох видів автоматизації. При рефакторінгу програм часто буває необхідно оновити і модульні тести, і зміна коду тестів може зайняти стільки ж часу, скільки і зміна основного коду. З іншого боку, при зміні інтерфейсу додатку необхідно заново переписати всі тести, які пов'язані з оновленими вікнами, що при великій кількості тестів може відняти значні ресурси.

Ще зовсім недавно тестування програм проводилося вручну або самими програмістами, або користувачами, що навряд чи можна було назвати системним підходом і до того ж це не дозволяло оцінювати якість коду. Трохи пізніше тестування виділилося в окрему галузь знань у складі розробки програмного забезпечення, але швидко прийшло розуміння того, що тестування вручну неефективне, оскільки вимагає великих трудових ресурсів і багато часу. Перші засоби автоматизації тестування практично представляли собою бібліотеки, які можна було використовувати для написання тестів, що вимагало від тестувальника вміння програмувати на рівні розробника. Сучасні засоби автоматизованого тестування дозволяють створювати автоматизовані тести з мінімальною участю людини.

Класифікація засобів та план тестування

Прийнято розділяти тестування за рівнями завдань і об'єктів на різних стадіях і етапах розробки програмного забезпечення [5]:

1. *Функціональне тестування* підсистем і програмного забезпечення в цілому з метою перевірки ступеня виконання функціональних вимог до програмного забезпечення – рекомендується проводити окремою групою тестувальників, які не підпорядковані керівнику розробки;
2. *Модульне тестування* частин програмного забезпечення (компонентів) з метою перевірки правильності реалізації алгоритмів – виконується розробниками;
3. *Тестування навантаження* (у тому числі стресове) для виявлення характеристик функціонування програмного забезпечення при зміні навантаження (інтенсивності звернень до нього, наповнення бази даних і тому подібне) – для виконання цієї роботи потрібні висококваліфіковані тестувальники і дорогі засоби автоматизації експериментів.

На ринку засобів автоматизованого тестування сьогодні представлено багато продуктів [5], більшість зібрана в табл. 1. Найбільш потужними з них є: HP (QuickTest Professional, WinRunner), IBM (Robot, Functional Tester), AutomatedQA (TestComplete) та Selenium, який зараз часто використовується для тестування Web-додатків. Частина з них використовує стандартні мови програмування (наприклад, QTP використовується в якості мови розробки скриптів VB, а Functional Tester, реалізований в середовищі Eclipse, дозволяє створювати скрипти на Java), а частина застосовує свої власні спеціалізовані мови (наприклад, Robot використовує мову SQABasic для написання скриптів). Більшість інструментів орієнтовані на роботу з Web-додатками або зі звичайними додатками, написаними із застосуванням технологій Net або Java.

Розробники засобів автоматизованого функціонального тестування досить оперативно реагують на появу нових механізмів і платформ розробки програмного забезпечення. Незважаючи на те, що на ринку існує величезна різноманітність різних продуктів для автоматизованого тестування, деякі компанії-розробники програмного забезпечення створюють власні інструменти, пристосовані для тестування розроблюваних ними додатків. Причинами цього є висока вартість засобів автоматизованого тестування та унікальність програмного забезпечення, що тестується, яка не дозволяє використовувати стандартні засоби автоматизації тестування.

Крім засобів тестування існують так звані засоби підтримки процесу тестування, що дозволяють вести облік вимоги і тест-кейси, проводити аналіз покриття вимоги тестами, керувати ходом виконання тестування, вести облік виявлених дефектів і тому подібне. Лідирує в даній області Web-додаток HP Quality Center – іструмент управління процесом тестування, який, включаючи управління вимогами і дефект-менеджмент, інтегрований із засобами функціонального і навантажувального тестування HP QuickTest Professional.

Таблиця 1 – Класифікація засобів автоматичного тестування

Назва	Розповсюдження, Ціна	Мова тестів	Технології
Функціональне тестування			
Canoo WebTest	Безкоштовний	HTML, Groovy	Web-технології
Codeception	Безкоштовний	PHP	Web-технології
Coded UI Test	Коштовний	C#	.NET 2.0, 3.0, 3.5, 4.
GUIdancer	Коштовний	Keyword-driven	Swing, SWT/RCP, GEF, HTML
HP QuickTest Professional	Коштовний, (~$10000)	VBScript	Web, Java, .Net, WPF, SAP
IBM Rational Functional Tester	Коштовний, $6000	Java, Visual Basic	Java,.NET,Win32,HTML,Terminal
M-eux	Коштовний	Java, C#, VBScript	Java, C#, VBScript
Ranorex	Коштовний, 290-1190 EUR	C#, VB.NET, Python (IronPython)	.NET (C#, VB.NET), WPF (XAML
Robot Framework	Безкоштовний	Python, Java	Java, .NET
RoutineBot	Коштовний, $495	Pascal, JavaScript, Basic	Windows Forms, Flex
Selenium	Безкоштовний	HTML, Java, C#, Perl, PHP, Python, Ruby	DHML, JavaScript, Ajax
Sikuli	Open Source	Sikuli Script	Jython, Windows, Linux, MacOS
SWAPY	Open Source	Python	Windows, Python
T-Plan Robot	Коштовний	Proprietary language Java	VNC, Windows, Linux, Java, C
TestComplete	Коштовний, $1000-5000	VBScript, JScript, DelphiScript, C++Script, C#Script	IE, Firefox, Chrome, Flash,
Testdroid	Коштовний,$595	Java	Java
Testing Anywhere	Коштовний, $7000	візуальне проектування	VB.NET, C#, C++, Win32, VB6
TestPlan	Безкоштовний	власна скрипкова мова	HTML, DHTML, JavaScript
Twist	Коштовний	Java, Groovy	Selenium, Java, Swing, Groovy.
Watir	Open Source	Ruby, Java, .NET, Perl	HTML, JavaScript
Модульне тестування			
Codeception	Безкоштовний	PHP	Web -технології
Selenium	Безкоштовний	HTML, Java, C#, Perl, PHP, Python, Ruby	DHML, JavaScript, Ajax
Testing Anywhere	Коштовний, $7000	візуальне проектування	VB.NET, C#, C++, Win32, VB6
Тестування навантаження			
AgileLoad	Коштовний, $70	SCL	Web -технології
Apache JMeter	Open Source	визуальное проектирование	HTTP, HTTPS, SOAP, JDBC, LDAP
BrowserMob	Коштовний	Selenium Record&Playback	AJAX, Flash
Testing Anywhere	Коштовний, $7000	візуальне проектування	VB.NET, C#, C++, Win32, VB6

Важливим моментом при створенні тестових систем та написанні тестів є наявність *плану тестування*, що визначений міжнародним стандартом IEEE 829-1983 [6]. В ньому повинні бути наступні розділи:

– що буде тестуватися (тестові вимоги, варіанти тестів);

– якими методами буде тестуватися система та критерії, за якими буде визначатися успіх тестів;

– план робіт та ресурси (тестувальники, техніка).

Також слід визначити певні критерії вдалого/невдалого завершення тестів, ризики і плани керування та конфігурації середовища та керування та т.п.

Коли зрозуміло, що і навіщо потрібно тестувати, і є план дій, саме час задуматися про те, як це зробити ефективніше, швидше і якісніше. Сучасне програмне забезпечення – це складний об'єкт, і вручну з ним справлятися важко і дорого. До того ж при «ручному» тестуванні результати кожного виконання тестів пропадають, і їх важко повторити. Для того щоб збільшити обсяг перевірок і підвищити якість тестування, забезпечити можливість повторного використання тестів при внесенні змін у програмне забезпечення застосовують засоби автоматизації тестування.

Сьогодні Україна відстає від решти комп'ютерного світу щодо застосування засобів автоматизації тестування, і для цього, на наш погляд, є декілька причин:

1. Неправильне ставлення до тестування як такого – багато керівників вважають, що розробник може написати програму, яка не містить помилок.
2. Висока вартість інструментів автоматизації тестування.
3. Бажання заощадити на кваліфікованих кадрах – робота спеціаліста з засобів автоматизації тестування обходиться дорожче, ніж праця звичайного тестувальника.
4. Обмеження по термінах – автоматизація тестування вимагає значних тимчасових витрат і має сенс тільки в тому випадку, якщо проект як мінімум середньостроковий.
5. Невдалий досвід застосування таких засобів і очікування миттєвого ефекту від їх впровадження. Адже результати впровадження таких продуктів помітні далеко не відразу і витрати на автоматизацію окупаються за тривалий період часу, що не дозволяє повністю відмовитися від тестування вручну.

Розробка фреймворку автоматизованого тестування під Android

Розглянемо існуючий підхід до автоматизованого тестування користувацького інтерфейсу, а також детальний розгляд розробленого фреймворку.

UI тестування під Android

На додаток до модульного тестування окремих компонентів, з чого складаються Android-додатки [2] (наприклад, активіті, сервіси і контент-провайдери), не менш важливо перевірити поведінку інтерфейсу користувача програмного додатка (UI), під час його виконання на пристрої. Тестування користувальницького інтерфейсу гарантує, що ваш додаток повертає правильний UI інтерфейс у відповідь на послідовність дій користувача на пристрої, таких як введення з клавіатури або натискання панелі інструментів, меню, діалогів, зображень та інших елементів управління користувальницького інтерфейсу.

Функціональне тестування або, так званий «чорний ящик» – тестування (без знання будови модуля та його взаємодії з іншими) користувальницького інтерфейсу не вимагає від тестувальників знати внутрішні деталі реалізації програми, тільки її очікуваних результатів при виконанні, коли користувач виконує певні дії або вводить конкретні дані. Такий підхід дозволяє краще підійти до питання розподілу ролей тестування в організації.

Загальний підхід до тестування UI – це провести тести вручну і переконатися, що додаток працює так, як від нього очікується. Тим не менш, цей підхід може бути трудомістким, стомлюючим і є більш схильним до генерації помилок. Більш ефективний і

надійний підхід полягає в автоматизації тестування користувацького інтерфейсу в рамках тестування програмного забезпечення. Автоматизоване тестування включає в себе створення програм для виконання тестових завдань (тестів) для покриття конкретних сценаріїв використання, а потім за допомогою тестувальних засобів (або фреймворків) для запуску тестів автоматично перевірка, що все було відтворено та виконано в правильному порядку та правильним чином.

На даний час у поставці Android SDK існують наступні засоби для автоматизованого тестування програмного забезпечення, написаного з використанням користувацького інтерфейсу під платформу Android:

– UiAutomatorViewer – інструмент, що дозволяє сканувати і аналізувати елементи графічного інтерфейсу програмного додатка під Android. Працює на базі прошарку API в системі, що відповідає за доступність кожного UI елементу (accessibility).

– UiAutomator – бібліотека, написана мовою Java, для створення кастомізованих функціональних UI тестів, що можуть бути виконані за допомогою запускача тестів, вбудованого в платформу Android [4].

На даний момент для використання даних засобів потрібно задовольнити наступні вимоги: Android ADK Tools ревізії 21 або вище, а також Android SDK API 16 або вище.

Послідовність виконання дій при UI тестуванні

Наведемо *методику* послідовності виконання дій при автоматизованому тестуванні графічного користувацького інтерфейсу:

1. Підготовка до тестування. До неї входить встановлення тестованого програмного додатка на пристрій, аналіз графічного інтерфейсу додатка, а також перевірка того, що графічні елементи додатку доступні для автоматизованого тестування на базі даного тестуваьного засобу.
2. Створення автоматизованих тестів для симуляції специфічних користувацьких дій над програмним додатком.
3. Компіляція тестових сценаріїв у JAR-архів, що виконується, та інсталяція його на пристрій, разом з тестованим додатком.
4. Запуск автоматизованого тестування та перегляд отриманих результатів.
5. Виправлення та корекція помилок та програмних дефектів, виявлених під час тестування.

Перед тим, як приступити до написання тестів, майже необхідно ознайомитися з компонентами користувацького інтерфейсу (у тому числі бути ознайомленим з видами елементів представлення графічного інтерфейсу в системі Android) тестованого програмного додатка. Для цього і стане в пригоді інструмент для візуалізації графічного інтерфейсу з точки зору програмного представлення. UiAutomatorViewer робить знімок екрана користувальницького інтерфейсу на будь-якому пристрої Android, підключеному до комп'ютера, на якому запущений даний інструмент. UiAutomatorViewer забезпечує зручний візуальний інтерфейс для перевірки макета ієрархії візуальних елементів UI і перегляду властивостей окремих компонентів користувацького інтерфейсу, які відображаються на тестованому пристрої. Використовуючи цю інформацію, можна пізніше створити UiAutomator тести з об'єктами селектор, які орієнтовані на конкретні компоненти для користувацького інтерфейсу для тестування.

Перед тим як використовувати тестувальний інструмент UiAutomator потрібно виконати наступні етапи попередньої підготовки:

– завантажити програмний додаток на пристрій. Переконатись, що тестований додаток встановлений на пристрої (з відповідною версією);

– дослідити користувацький інтерфейс тестованої програми. Використовуючи раніше описаний UiAutomatorViewer для отримання представлення ієрархії макетів

та графічних елементів. Найчастіше всього елементи, з якими взаємодіє людина, є елементами з певними візуальними характеристиками, такими як текст, або специфічне графічне зображення, яке легко можна буде розрізнити, використовуючи селекторні об'єкти тестувального інструменту. Якщо є доступ до вихідного коду тестованого додатку, то досить корисним може бути додавання до всіх графічних елементів опису (а саме поля контент-опису) для полегшення його доступності при автоматизованому тестуванні;

– необхідно зробити налаштовування середовища розробки. Цей пункт є не менш важливим серед решти. Адже навіть при готовності всіх попередніх, відсутність налаштованого середовища може призвести до затримок.

Створення автоматизованих тестів на UiAutomator інструменті

API тестового інструменту UiAutomator представлений наступними класами:

– UiDevice – представляє собою стан мобільного пристрою, що надає уніфікований доступ до різних його характеристик. А також надає легкий інтерфейс для виконання дій над графічним інтерфесом, такими як натискання кнопок, дотик до екрана та ін.;

– UiSelector – представляє собою критерій пошуку графічних елементів, що представлені в певний момент часу на екрані пристрою. Якщо на екрані знаходиться декілька елементів, що підходять під пошуковий критерій – то повертається перщий з них у ієрархії графічних елементів;

– UiObject – представляє собою об'єкт-елемент графічного інтерфейсу. Створюється на базі UiSelectoroб'єкта;

– UiCollection – наслідник класу UiObject. Представляє собою колекцію елементів графічного інтерфейсу.

– UiScrollable – наслідник класу UiCollection. Представляє собою колекцію елементів графічного інтерфейсу, що може бути проскролена [4].

Основні компоненти фреймворку для тестування

Актівіті або вид

Дуже часто при написанні тестів програміст зосереджується лише на конкретному моменті часу, та на певній множині дій. Не думаючи про майбутнє можливе розширення функціонала програми. Що призводить до подальших проблем при впровадженні нового функціонала в програму, адже тоді доводиться майже повністю переписувати тести.

В першу чергу графічний інтерфейс програм на платформі Android – це актівіті. Таким чином легко передбачити, що першою основною компонентою нашого фреймворку будуть саме абстрактні об'єкти, що представляють собою актівіті або вид. А саме об'єкти, які будуть являти собою візуальне представлення для користувача. В свою чергу такі об'єкти мають свої параметри та набір методів. А також надають легкий та зручний інтерфейс доступу до вкладених у актівіті графічних елементів. Такий підхід у першу чергу дозволяє легко будувати структуру графічних елементів, а також застосувати певний алгоритм локації їх у рамках обраної абстракції. Ми можемо легко розділити логіку тестів та конкретну специфікацію актівіті, що тестується, в рамках цілого додатка.

На цьому ж рівні абстракції ми описуємо всі елементи актівіті. Тут можна додати і характерні абстрактні класи елементів, інтерфейси доступу до них. Тобто створити абстракцію для елементів керування користувацьким інтерфейсом.

Атомарні дії

Мета даного рівня абстракції – організувати дії над елементами актівіті, обгорнути їх в атомарні кроки. Де під поняттям крок ми розуміємо найменшу логічну дію. Розмір цих атомарних дій залежатиме вже від конкретної області програмного додатка та побудови тестів. Ці ж найменші атомарні дії і будуть компонентами для створення складених дій.

Можна виділити два основних класи атомарних дій: звичайна дія та дія з верифікацією. До перших можна віднести всі дії, що можуть бути виконані користувачем, а також абстрактні дії, які є лише загальними для своїх наслідників. До класу ж дій з верифікацією всі такі, що крім самого виконання дії ще і верифікують певні речі, такі як результат виконання дії або проміжні результати.

Складені дії

Складені дії будуються з кількох частин. А саме: підготовчих дій, основних кроків тесту з верифікацією і кроків, що закінчують. В цілому задача складених дій полягає у групуванні атомарних дій за сутностями для повторного їх використання. Це надаватиме змогу розробляти тести, притримуючись основних правил написання об'єктно-орієнтованого коду.

Основою для складених дій є насамперед підготовчі кроки. Адже дані операції потрібно якомога більше оптимізувати з питань повторного використання коду для споріднених сутностей.

Після підготовчих кроків йдуть основні кроки. Вони будуть формуватись на більш високому рівні абстракції, де саме і будуть створюватись сценарії виконання тестів. Але чим краще буде розроблена гнучка система завдання атомарних та складених дій, тим легше і швидше можна буде створювати тести.

Останньою складовою є закінчуючі кроки. Вони є не менш важливими, адже після них ми маємо отримати певний результат (після верифікації), а також при можливості повернути стан системи до такого, щоб ми могли успішно продовжити тестування. Тобто не створювати побічні ефекти після виконання даного тесту.

Послідовність дій тестувальника по застосуванню нашого фреймворку для тестування додатка (калькулятор)

Загальна послідовність дій UI тестування була наведена вище, зараз ми розглянемо дії, які треба зробити тестувальнику програмного додатка під Android з використанням нашого фреймворку:

1. Підготовка до тестування.

Під'єднуємо Android мобільний пристрій до комп'ютера за допомогою USB-з'єднання та інстальованої програми Android Debug Bridge.

На комп'ютері запускаємо UiAutomatorViewer, який надасть можливість зробити скриншоти та проаналізувати графічний інтерфейс тестованого додатка (калькулятор).

Запускаємо на мобільному пристрої обраний додаток та виконуємо тест «вручну». За допомогою UiAutomatorViewer, ми бачимо ієрархічну структуру візуальних об'єктів графічного інтерфейсу та метадані цих об'єктів. Для нашого тестованого додатка тестувальнику легко побачити значення в полі text після виконання ручного набору. Наприклад, рядок, що фактично задає набор метаданих, які повинен використовувати тестувальник при створенні автоматичних тестів, виглядає: *«Left parenthesis2 point 5plus1right parenthesismultiplied byleft arenthesis7minus3right parenthesisdivided by2»*.

2. Створення автоматизованих тестів для симуляції специфічних користувацьких дій над програмним додатком.

Порівняємо фрагменти коду табл. 2 для нашого тестового прикладу (додаток-калькулятор), який тестувальник повинен був написати в загальному випадку, використовуючи UiAutomator, та використовуючи наш фреймворк.

Як бачимо при роботі з нашим фреймворком код більш простий, легко підтримуваний та зручний для написання, який може легко написати тестувальник,

3. Компіляція тестових сценаріїв у JAR-архів, що виконується, та інсталяція його на пристрій разом з тестованим додатком.

Після того як ми маємо готовий набір тестів за допомогою Android SDK виконуємо процедуру компіляції тестових класів у JAR-архів, що виконується на DVM (Dalvik Virtual Machine), та переносимо його на мобільний пристрій для подальшого тестування.

4. Запуск автоматизованого тестування та перегляд отриманих результатів.

Запускаємо автоматизовані тести з командного рядку на комп'ютері, для обраного тестованого додатка:

«adb shell uiautomator runtest CalculatorTest.jar -c calc.tests.MainTest».

5. Виправлення та корекція помилок, виявлених під час тестування.

При виникненні помилок при тестуванні, повідомляємо розробника та починаємо тестування нової версії додатка.

Приклад роботи тесту, написаного з використанням нашого фреймворку, можна побачити на рис. 1.

Таблиця 2 – Фрагмент коду для тесту додатка калькулятор

Код з використанням UiAutomator	Код з використанням нашого фреймворку
public void testComplexExpressionD() throws UiObjectNotFoundException, IllegalArgumentException { UiObject leftParenthesis = new UiObject(new UiSelector().description("left parenthesis")); leftParenthesis.click(); UiObject buttonTwo = new UiObject(new UiSelector().text("2")); buttonTwo.click(); UiObject pointButton = new UiObject(new UiSelector().description("point")); pointButton.click(); UiObject buttonFive = new UiObject(new UiSelector().text("5")); buttonFive.click(); UiObject plusButton = new UiObject(new UiSelector().description("plus")); plusButton.click(); … buttonTwo.click(); UiObject editText = new UiObject(new UiSelector().className("android.widget.EditText")); *assertEquals("left parenthesis2 point 5plus1right parenthesismultiplied byleft parenthesis7minus3right parenthesisdivided by2", editText.getText());* UiObject equalsButton = new UiObject(new UiSelector().description("equals")); equalsButton.click(); assertEquals("7", editText.getText()); }	*public void testComplexExpression() throws UiObjectNotFoundException, IllegalArgumentException {* calc.pressParenthesisButton(Parenthesis.Left); calc.pressNumberButton(2); calc.pressPointButton(); calc.pressNumberButton(5); calc.pressOperation(Operation.Plus); … calc.pressNumberButton(2); *assertEquals("left parenthesis2 point 5plus1right parenthesismultiplied byleft parenthesis7minus3right parenthesisdivided by2", calc.getFormulaText());* calc.pressOperation(Operation.Equals); assertEquals("7", calc.getFormulaText());}

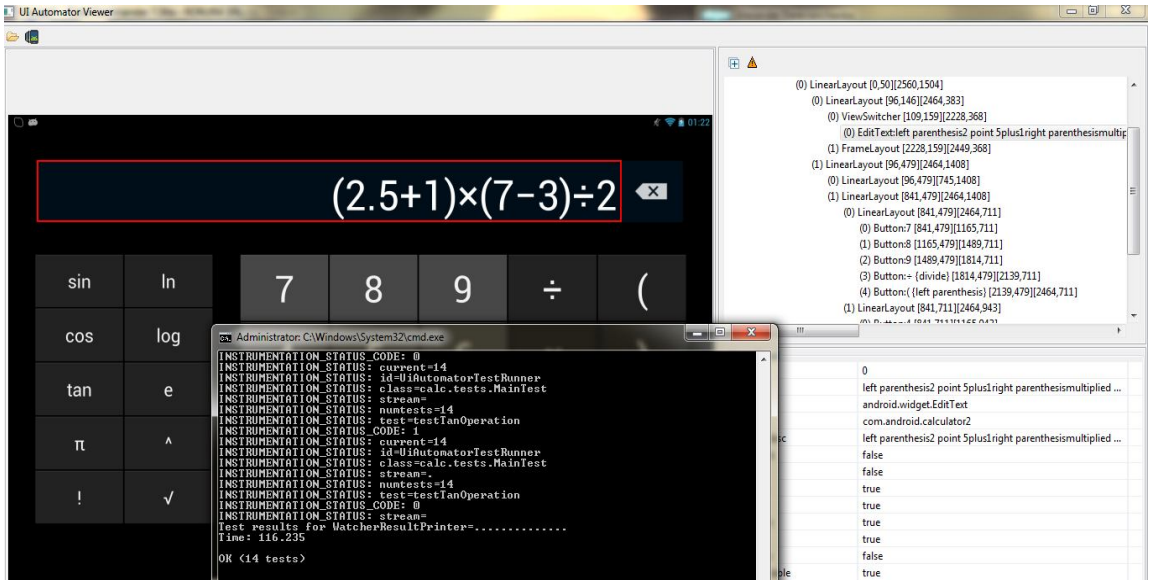

Рисунок 1 – Приклад роботи тестового фреймворку з додатком під Android

Висновки по створеному фреймворку

Таким чином створений нами фреймворк надає достатній рівень абстракції для того, щоб створення автоматизованих тестів графічного інтерфейсу проходило достатньо швидко і при цьому на якісно високому рівні.

Для кожного активіті ми маємо свій об’єкт, який має свої структурні елементи, методи для взаємодії з ними, а також легкої локації їх в контексті даного активіті.

У свою чергу елементи описуються на тому ж рівні абстракції, що і активіті, або види. Адже в системі Android всі графічні елементи легко представити у вигляді об’єктів. Що надає нам додаткових мотивацій до використання саме такого підходу.

Але створення абстракції для одних лишень елементів було б недостатньо, тому наш фреймворк надає змогу оголошувати атомарні або мінімальні логічні дії, а також складені дії. Що надає нам змогу таким чином будь-який сценарій тесту описати, використовуючи даний фреймворк, а отже і розробити тести на базі даного фреймворку.

Висновки

У роботі проаналізовані існуючі підходи до автоматизації тестування користувацького інтерфейсу в загальному, було проаналізовано структуру та основні компоненти системи Android, а також описаний основний існуючий підхід до автоматизованого тестування графічного інтерфейсу в ній.

У даній роботі був розроблений фреймворк для швидкого написання автоматизованих тестів користувацького інтерфейсу під Android. На основі цього фреймворку були написані автоматизовані тести для стандартного для Android систем додатка (калькулятор) – рис. 1. Слід зазначити, що реалізований фреймворк набагато спростив задачу написання цих тестів у швидкий час, та дотримуючись правил написання об’єктно-орієнтованого коду. А також дані тести можуть бути підтримувані та змінені за досить короткий час при впровадженні нового функціонала у будь-який додаток.

У подальшому можна вдосконалювати розроблену систему в різних напрямах. Можна розробляти додаткові рівні абстракції, характерні для певних класів програмного забезпечення під Android [7], [8]. За рахунок будови фреймворку, створені нові

рівні абстракції можуть бути легко впроваджені та адаптовані. Також можна розробляти конкретні реалізації певних класів атомарних та складених дій з метою ще більшого пришвидшення написання автоматизованих тестів.

Для демонстрації ефективності фреймворку, як приклад, було реалізовано набір автоматизованих тестів для тестування графічного інтерфейсу, стандартного в системі Android додатка (калькулятору). На практиці було показано ефективність тестів, легкість їх написання, а також якість програмного коду, який представляє цей набір тестів.

Підсумовуючи те, що було зроблено, слід зазначити, що даний напрямок – автоматизації тестування графічного інтерфейсу під Android [9], дуже актуальний на сьогодні. Розробка програмного забезпечення під мобільні пристрої під керуванням системи Android є дуже швидким процесом, тому дуже гостро постає питання тестування графічного інтерфейсу розроблюваних додатків, особливо в разі впровадження нового функціонала. Тому написання тестів повинно відбуватись швидко, надійно та на якісно високому рівні з метою подальшого підтримання програмного коду тестів. Тому дана робота робить досить суттєвий внесок у розвиток напрямку створення автоматизованих фреймворків тестування графічного інтерфейсу.

Література

1. Fewster M. Software Test Automation / M. Fewster, D. Graham. – ACM Press, 1999. – 600 p.
2. Murphy M.L. The Busy Coder's Guide to Android Development / Murphy M.L. – Commonsware LLC, 2008. – 400 p.
3. Komatinemi S. Pro Android 3 / S. Komatinemi, MacLean D., S.Y. Hashimi. – Apress, 2011. – 1200 p.
4. [Електронний ресурс]. – Режим доступу : http://developer.android.com/tools/testing/testing_ui.html - UI Testingfr Android.
5. [Електронний ресурс]. – Режим доступу : http://automated-testing.info/tools - Web site about Tests Automation.
6. [Електронний ресурс]. – Режим доступу : http://standards.ieee.org/findstds/standard/829-1983.html - 829-1983 - IEEE Standard for Software Test Documentation.
7. GUI Testing Using Computer Vision / [Tsung-Hsiang Chang, Tom Yeh, Robert C. Miller]. – ACM, CHI, 2010. – P. 1535-1544.
8. Sikuli: using GUI screenshots for search and automation / [Tom Yeh, Tsung-Hsiang Chang, Robert C. Miller] // In Proceedings of the 22nd annual ACM symposium on User interface software and technology (UIST '09). – ACM, NY, USA. – P. 183-192.
9. [Електронний ресурс]. – Режим доступу : http://www.vogella.com/articles/AndroidTesting/article.html#roboelectric_overview – Android application testing with the Android test framework.
10. Binder R.V. Testing Object-Oriented Software: Models, Patterns, and tools / Binder. R.V. – Addison-Wesley, 1999. – 1203 p.

Literatura

1. Fewster M. Software Test Automation / M. Fewster, D. Graham. – ACM Press, 1999. – 600 p.
2. Murphy M.L. The Busy Coder's Guide to Android Development / Murphy M.L. – Commonsware LLC, 2008. – 400 p.
3. Komatinemi S. Pro Android 3 / S. Komatinemi, MacLean D., S.Y. Hashimi. – Apress, 2011. – 1200 p.
4. [Електронний ресурс]. – Режим доступу : http://developer.android.com/tools/testing/testing_ui.html - UI Testingfr Android.
5. http://automated-testing.info/tools - Web site about Tests Automation.
6. http://standards.ieee.org/findstds/standard/829-1983.html - 829-1983 - IEEE Standard for Software Test Documentation.
7. GUI Testing Using Computer Vision / [Tsung-Hsiang Chang, Tom Yeh, Robert C. Miller]. – ACM, CHI, 2010. – P. 1535-1544.

8. Sikuli: using GUI screenshots for search and automation / [Tom Yeh, Tsung-Hsiang Chang, Robert C. Miller] // In Proceedings of the 22nd annual ACM symposium on User interface software and technology (UIST '09). – ACM, NY, USA. – P. 183-192.
9. [Електронний ресурс]. – Режим доступу : http://www.vogella.com/articles/AndroidTesting/article.html#roboelectric_overview – Android application testing with the Android test framework.
10. Binder R.V. Testing Object-Oriented Software: Models, Patterns, and tools / Binder. R.V. – Addison-Wesley, 1999. – 1203 p.

RESUME

O.V. Derevyanchenko, V.O. Plakydiuk

System of the Android Applications Testing

This paper analyzes existing approaches to automated user interface testing in general, a structure and main components of the Android system, and also describes basic existing nowadays approach to automated GUI testing in this system.

In this paper was developed a framework for fast writing of automated user interface tests for Android. Based on this framework it was written automated tests for standard Android system application (calculator). It should be noted, that the developed framework simplified a task of writing these tests in short time, and writing them following the rules of object-oriented code style. And these tests can be supported and modified in a quick manner, if a new functionality was introduced into application.

Стаття надійшла до редакції 08.06.2013.