Тестирование является важной составляющей процесса разработки приложения. Для Android тестирование особенно важно, так как устройства сильно отличаются друг от друга:

* Размером и разрешением экрана.
* Версией Android.
* Форм-фактором.
* Системой команд процессора.
* Наличием фронтальной камеры, NFC, внешней клавиатуры, и т.д.

Поэтому тестировать приложение приходится на множестве устройств.  
В процесс тестирования входят различные виды тестирования. Рассмотрим, как происходит процесс функционального тестирования приложения вручную. Тестировщик устанавливает на устройство приложение, вдумчиво проверяет всю функциональность, затем возвращает устройство в первоначальное состояние. И так для каждого приложения и каждого устройства. Очевидный недостаток такого способа – большие затраты времени для регулярного тестирования.  
Очевидный плюс автоматизированного тестирования – его можно регулярно проводить без особых затрат. Например, каждую ночь тестировать свежий билд приложения на всем множестве имеющихся устройств, а утром анализировать результаты и исправлять ошибки.  
В данной заметке будут рассмотрены средства реализации автоматического тестирования. Рассматриваются только инструменты, входящие в Android SDK или распространяющиеся под Open Source лицензией.

## **Концепция автоматического тестирования**

Задача — с наибольшей точностью автоматизировать действия, которые выполняет тестировщик. Давайте их рассмотрим. В наличии есть несколько приложений и несколько Android устройств. Для каждого приложения и каждого устройства выполняются следующие шаги:

1. Установка приложения на устройство
2. Запуск приложения
3. Тестирование приложения выбранным способом
4. Удаление приложения
5. Сброс состояния устройства

На каждом шаге нужно собрать и проанализировать данные, например логи и скриншоты. Затем на основе этих данных сформировать результат тестирования.  
  
Далее рассматриваются средства, позволяющие автоматизировать перечисленные шаги.

## **Управление Android устройствами**

Для начала нужно выделить компьютер на котором будет запускаться автоматическое тестирование и настроить на нем [Android SDK](http://developer.android.com/sdk/index.html). Примеры приводятся для компьютера с установленной ОС Linux.  
  
На всех тестируемых устройствах нужно отключить экран блокировки и максимально увеличить время ожидания. Для некоторых методов тестирования нужно отключить смену ориентации экрана.  
  
В Android SDK имеются две утилиты для управления устройствами: adb и MonkeyRunner.  
  
Я постараюсь подробно описать автоматизацию действий, использующихся при тестировании. Тем, кто знаком с ADB и MonkeyRunner имеет смысл сразу переходить к разделу «[Способы автоматизированного тестирования](https://habr.com/ru/company/intel/blog/152122/#testing_methods)».

### **Управление с помощью утилиты ADB**

ADB (Android Debug Bridge) – утилита для управления Android устройствами из командной строки. Официальная документация по ADB: [developer.android.com/tools/help/adb.html](http://developer.android.com/tools/help/adb.html)  
  
Утилита adb находится в директории <android\_sdk>/platform-tools/. Путь к данной директории рекомендуется прописать в переменной окружения PATH.

### **Проверка работы ADB**

Устанавливаем и настраиваем Android SDK, подключаем к компьютеру Android устройства и выполняем команду:

adb devices

Команда выдаст список всех подключенных устройств. Если список устройств не пуст, значит ADB настроен и работает.

### **Работа с несколькими устройствами**

Чтобы указать ADB с каким устройством нужно работать, следует прописать серийный номер устройства после ключа -s:

adb -s <serial\_number> <command>

Например:

adb -s <serial\_number> logcat

Серийный номер устройства можно посмотреть командой adb devices. Ключ -s позволяет работать одновременно с несколькими подключенными устройствами. В дальнейшем ключ -s в командах я указывать не буду.

### **Основные команды ADB**

Открыть консоль на устройстве:

adb shell

Запустить команду на устройстве:

adb shell <command>

В Android присутствуют многие стандартные утилиты Linux: ls, cat, dmesg,…  
  
Установить приложение из apk файла:

adb install example.apk

Удалить приложение:

adb uninstall <package>

Название package можно получить из apk файла командой:

aapt dump badging example.apk | grep "package"

Загрузить файл с устройства на компьютер:

adb pull <path-on-device> <file>

Загрузить файл с компьютера на устройство:

adb push <file> <path-on-device>

**Примечание:**  
В большинство директорий на устройстве разрешен доступ только на чтение. Доступ на запись разрешен в директорию /sdcard (из нее нельзя запускать программы) и /data/local/tmp/.  
  
Запуск приложения:

adb shell am start -n <package>/<activity>

Запускает указанную activity. Название activity, которая запускается при выборе приложения в меню можно получить из apk файла командой:

aapt dump badging example.apk | grep "launchable-activity"

### **Чтение логов**

Чтение логов в Android производится утилитой logcat.  
Домашняя страница утилиты logcat: [developer.android.com/tools/help/logcat.html](http://developer.android.com/tools/help/logcat.html)  
  
Считать логи с устройства (блокируется до нажатия Ctrl-C):

adb logcat

Очистить буфер логов на устройстве:

adb logcat -c

Считать буфер логов на устройстве (выдает текущее содержимое буфера, не блокируется):

adb logcat -d

Пример:

adb logcat -c # очищаем буфер логов

# выполняем действие

adb logcat -d > file.log # сохраняем текущее содержимое буфера логов в file.log

### **Снятие скриншотов с помощью утилиты screencap**

Утилита screencap сохраняет текущее содержимое экрана в графический файл:

adb shell screencap /sdcard/screen.png

adb pull /sdcard/screen.png screen.png

adb shell rm /sdcard/screen.png

Утилита screencap имеется на телефонах с Android 4.x и выше. На предыдущих версиях Android снятие скриншотов можно производить с помощью MonkeyRunner.

### **Пример BASH скрипта для тестирования приложения c помощью ADB**

**Скрипт: app\_test.sh**

## **Управление с помощью MonkeyRunner**

Утилита MonkeyRunner предоставляет API для написания скриптов, которые управляют Android устройстами. С помощью MonkeyRunner можно написать скрипт на языке Python, который устанавливает Android приложение, запускает его, имитирует действия пользователя, снимает скриншоты и сохраняет их на компьютер. Утилита MonkeyRunner использует [Jython](http://www.jython.org/) для выполнения скриптов.  
  
  
  
Домашняя страница утилиты MonkeyRunner и описание API: [developer.android.com/tools/help/monkeyrunner\_concepts.html](http://developer.android.com/tools/help/monkeyrunner_concepts.html)

### **Чтение логов с помощью MonkeyRunner**

Файл log.py:

# coding: utf-8

**from** com.android.monkeyrunner **import** MonkeyRunner, MonkeyDevice

**def** **log**(fn, device):

msg = device.shell('logcat -d')

f\_log = open(fn, 'at')

**if** msg **is** None:

msg = 'None'

f\_log.write(msg.encode('utf-8'))

f\_log.close()

device.shell('logcat -c')

**if** \_\_name\_\_ == '\_\_main\_\_':

device = MonkeyRunner.waitForConnection()

device.shell('logcat -c') # Очищаем буфер логов

# ...

log('example.log', device) # Записываем логи

Запуск:

monkeyrunner log.py

Скрипт запишет логи в файл example.log в текущей директории.

### **Снятие скриншотов**

Файл screen.py:

# coding: utf-8

**from** com.android.monkeyrunner **import** MonkeyRunner, MonkeyDevice

**if** \_\_name\_\_ == '\_\_main\_\_':

device = MonkeyRunner.waitForConnection()

image = device.takeSnapshot()

image.writeToFile('screenshot.png','png')

Запуск:

monkeyrunner screen.py

Скрипт снимает скриншот и сохраняет его в файл screenshot.png в текущей директории.

### **Пример управления устройством с помощью MonkeyRunner**

**Скрипт: monkeyrunner\_test.py**

Запуск:

monkeyrunner monkeyrunner\_test.py

## **Средства автоматизированного тестирования**

### **Тестирование с помощью monkey**

Представьте, что устройство попало в цепкие лапы очень активной и творческой обезьяны – утилита monkey призвана имитировать подобную ситуацию.  
  
Утилита monkey входит в состав Android SDK. Утилита отправляет на устройство поток псевдо-случайных действий пользователя. Параметры командной строки задают количество действий пользователя, соотношение их типов и имя тестируемого пакета, чтобы, например, обезьяна не вышла за пределы тестируемого приложения и не начала рассылать SMS по всем контактам из адресной книги.  
  
Примеры использования и перечень параметров приведены на домашней странице: [developer.android.com/tools/help/monkey.html](http://developer.android.com/tools/help/monkey.html)  
  
Главное достоинство monkey – отсутствие затрат на поддержку. Кроме того, стресс-тестирование приложения потоком произвольных событий может обнаружить нетривиальные ошибки.  
  
Недостатки тестирования утилитой monkey:

* Неэффективно для тестирования функционала, вызываемого сложной последовательностью действий. Например, monkey не сможет пройти аутентификацию и основной функционал приложения останется без внимания.
* Игры со сложным управлением, требующим быстрой реакции и сложных жестов, будут завершатся в самом начале, либо вообще не начнутся.
* Ошибки, найденные с помощью monkey, очень сложно воспроизвести.
* Нет проверки состояния приложения.

С помощью утилиты monkey можно без усилий протестировать любое приложение – это неплохая отправная точка. Возможно, что этот способ покажет адекватные результаты для конкретного приложения. Если же качество тестирования неудовлетворительное, то следует воспользоваться другими способами тестирования.

### **Тестирование с помощью MonkeyRunner**

При помощи скриптов использующих MonkeyRunner API можно не только разработать основу для тестирующей системы, но и написать скрипты для тестирования конкретного приложения на конкретном устройстве.  
  
**Достоинства:**

* Гибкость – реализовать можно практически все, что угодно.

**Недостатки:**

* Сложность написания скриптов даже в простых случаях.

Как правило, этот способ не оправдан – написание скриптов занимает много времени. Однако в частных случаях этот способ может сработать.

### **Тестирование с помощью getevent/sendevent**

Утилиты getevent и sendevent позволяют записать последовательность действий пользователя, а затем воспроизвести эту последовательность. Утилиты находятся на самом Android-устройстве и не требуют для работы root-доступа.  
**Достоинства:**

* Последовательности действий могут быть записаны без дополнительных затрат в ходе ручного тестирования, если оно уже проводится.
* Для записи сценариев не требуются навыки программирования.

**Недостатки:**

* Последовательности действий необходимо записывать отдельно для каждого приложения и для каждого устройства. При изменении интерфейса приложения все записанные действия необходимо проделать заново.
* Отсутствует проверка состояния приложения. Например, при тестировании браузера открывается страница. Если она открывается дольше, чем в момент записи, то дальнейшие действия будут выполнены до полной загрузки страницы и результат будет некорректный. Иногда возможно записать скрипт таким образом, что во всех подобных случаях ожидание превышает максимально возможное.
* Быстрая и сложная последовательность действий будет воспроизводиться дольше, чем записывалась – поэтому способ не всегда подойдет для тестирования динамичных игр, где критично время реакции и своевременность действия.

Запись последовательности действий:

# Записываем последовательность событий

# выполняем действия на устройстве, по окончанию нажимаем Ctrl-C

adb shell getevent -t > events.txt

# Преобразуем последовательность в исполняемый скрипт

./decode\_events.py events.txt > events.sh

# Загружаем скрипт на устройство

adb push events.sh /data/local/tmp/

# Устанавливаем права на запуск

adb shell chmod 755 /data/local/tmp/events.sh

# Запускаем скрипт

adb shell sh /data/local/tmp/events.sh

**Скрипт: decode\_events.py**

На устройстве должны воспроизвестись записанные действия.

### **Тестирование с помощью Robotium**

В отличии от рассмотренных ранее способов Robotium не входит в состав Android SDK, а распространяется под Open Source лицензией.  
  
Домашняя страница Robotium: [code.google.com/p/robotium](http://code.google.com/p/robotium/)  
  
Главное отличие Robotium в том, что тестовые действия описываются на уровне интерфейса приложения. В рассмотренных ранее способах тестовые действия явно или неявно описывались на уровне устройств ввода.  
  
Например, в приложении нужно нажать кнопку «OK». С помощью скрипта MonkeyRunner нажатие на кнопку реализуется как: «Коснуться точки экрана с координатами (x0, y0)». С помощью Robotium это реализуется как: «Нажать кнопку с текстом «OK»».  
  
Когда действия описываются на уровне интерфейса приложения их можно сделать независимыми от расположения элементов интерфейса, разрешения экрана и положения устройства.  
  
Кроме того, Robotium позволяет проверять реакцию приложения на действие.  
  
Например, после нажатия на кнопку «OK» в приложении должен появиться список с элементом «Item 1». С помощью Robotium можно проверить, появился ли список с таким элементом.  
  
Если выполнять проверки после каждого действия, то легко обнаружить, на каком шаге произошла ошибка.  
  
**Недостатки:**

* Для каждого приложения необходимо разработать сценарий тестирования на языке Java. Это требует навыков программирования и временных затрат.
* При изменении интерфейса приложения сценарий тестирования придется модифицировать.
* Написать сценарий Robotium сложнее, чем записать действия с помощью getevent/sendevent.

В целом, Robotium позволяет разрабатывать наиболее качественные сценарии тестирования при адекватных затратах.

### **Сравнение способов тестирования**

|  |  |  |
| --- | --- | --- |
| **Способ тестирования** | **Достоинства** | **Недостатки** |
| Monkey – поток случайных действий пользователя. | Отсутствуют затраты на сопровождение. Не зависит от устройства. Стресс-тестирование позволяет обнаружить нетривиальные ошибки. | Качество тестирования варьируется от приложения к приложению. Найденные ошибки сложно воспроизвести. Нет проверки состояния приложения. |
| MonkeyRunner – скрипт управления устройством. | Гибкость. | Сложность написания и поддержки скриптов даже для простых приложений. |
| getevent/sendevent – запись/воспроизведение действий пользователя. | Для записи последовательности действий не требуются навыки программирования. | Записанная последовательность действий подходит только к одному устройству при фиксированной ориентации. При изменении интерфейса приложения необходимо заново записать последовательность действий. Нет проверки состояния приложения. |
| Robotium – сценарий тестирования интерфейса приложения с проверкой состояния. | Действия описываются на уровне интерфейса приложения. Сценарий может быть независимым от разрешения экрана и ориентации устройства. После совершения действия можно проверять состояние приложения. | Сложность написания сценариев на языке Java. При изменении интерфейса приложения сценарий придется модифицировать. |

## **Анализ результатов**

В результате тестирования приложения перечисленными выше способами мы получили логи и скриншоты. Теперь их нужно проанализировать на наличие ошибок.

### **Анализ логов**

Для начала можно сделать поиск по подстрокам:

* I/DEBUG
* FATAL EXCEPTION
* WIN DEATH

Список можно дополнять по мере выявления ошибок в ходе ручного тестирования.

### **Анализ скриншотов**

В процессе тестирования вручную можно подготовить серию скриншотов в ключевых моментах тестирования, а затем сравнивать их с содержимым экрана в процессе автоматизированного тестирования. Это позволит определить, правильно ли идет процесс автоматизированного тестирования и выявлять ошибки.  
  
Также полезно сравнивать скриншот до и после запуска приложения – это позволяет определять случаи, когда приложение аварийно завершается без сообщений на экране и в логах.  
  
MonkeyRunner позволяет сравнить два скриншота с заданным допуском в процентах:

image1 = device.takeSnapshot()

# ...

image2 = device.takeSnapshot()

**if** image2.sameAs(image1, 0.1):

**print** 'image1 and image2 are the same (10%)'

К сожалению, в API MonkeyImage не предусмотрена функция загрузки из файла. Поэтому для сравнения сохраненных скриншотов придется писать свою функцию, например с помощью [Python Imaging Library](http://www.pythonware.com/products/pil/).

## **Сброс состояния устройства после тестирования**

После тестирования приложения устройство нужно вернуть в первоначальное состояние.  
  
Этого можно достичь несколькими путями:

* Многократное нажатие кнопки «Назад».
* Перезагрузка устройства.
* Перезапуск процесса zygote.

Рассмотрим первый вариант, как наиболее адекватный.

### **Многократное нажатие кнопки «Назад»**

Нажимаем кнопку «Назад» используя MonkeyRunner:

**for** i **in** xrange(0, 10):

device.press('KEYCODE\_BACK', MonkeyDevice.DOWN\_AND\_UP)

time.sleep(0.5)

На практике этот вариант оптимален, так как имитирует поведение реального пользователя.

## **Заключение**

В заметке были рассмотрены некоторые способы автоматического тестирования Android приложений, их достоинства и недостатки. Кроме того, рассмотрены инструменты, входящие в Android SDK или распространяющиеся под Open Source лицензией.  
  
Хочется отметить, что автоматическое тестирование не является панацеей и не заменяет другие виды тестирования. Качественный продукт получается при грамотно построенном процессе тестирования, сочетающем различные способы.