

Zesium mobile d.o.o
Mičurina 8
21000 Novi Sad



Anja Bešić
Branislav Manojlović

Praktikum

Alati za testiranje mobilnih aplikacija 1

Monkey testing, Monkey Runner

Novi Sad, 2017

Sadržaj:

UVOD	2
1. Testiranje softvera	3
1.1. Faze životnog ciklusa u testiranju softvera	3
1.2. Proces testiranja	3
1.3. Mobilno testiranje	4
1.4. Podele testiranja	5
1.5. Strategije testiranja	5
2. Mobilno testiranje	7
2.1. Tipovi mobilnih aplikacija	7
2.1.1. Native mobilne aplikacije	7
2.1.2. Web aplikacije	7
2.1.3. Hibridne aplikacije	7
3. Monkey testing	8
4. Monkey Runner alat za automatizovanje procesa testiranja	11
4.1. Moduli alata	12
4.1.1. MonkeyRunner	12
4.1.2. MonkeyImage	13
4.1.3. MonkeyDevice	14
5. Primeri	15
6. Zaključak	20
LITERATURA	21

Uvod

Testiranje je proces evaluacije softvera i svih njegovih pratećih komponenata, kako bi se utvrdilo da li su kreirani u skladu sa zahtevima, da li odgovaraju svrsi za koju su namenjeni i da li postoje defekti (bagovi) u njima.

Testiranje podrazumeva puštanje sustema (softvera) u rad, sa ciljem da se identifikuju greške ili nedostaci tog sistema u odnosu na očekivane rezultate. Svako ponašanje softvera koje se ne poklapa sa originalnim zahtevima, predstavlja grešku koju je potrebno identifikovati i otkloniti.

U užem smislu, testiranje predstavlja proveru da li je softver u potpunosti implementiran prema zahtevima korisnika.

U širem smislu, testiranje predstavlja sistem kontrole kvaliteta kojim se proverava softver i sve njegove prateće komponente.

Kvalitet softvera je opisan zahtevima kojima se definiše šta se najčešće očekuje od softverskih komponenti. Cilj testiranja softvera je provera da li su navedeni zahtevi ispunjeni.

Mobilno testiranje zahteva da se većina testova izvodi manuelno. Ipak, postoje delovi procesa mobilnog testiranja koji mogu biti automatizovani. Tester i mobilnih aplikacija trebaju težiti da automatizuju deo procesa testiranja kako bi dobili brže povratne informacije o stanju aplikacije.

Namena praktikuma je da objasni osnovne pojmove u iz oblasti testiranja mobilnih aplikacija, da opiše metodu automatizovanog testiranja na principu Monkey testing-a i da prikaže alat za automatsko testiranje mobilnih Android aplikacija Monkey Runner i da opiše način njegove upotrebe.

Prvo poglavlje, opisuje faze u životnom ciklusu testiranja softvera, proces testiranja, podele testiranja i strategije testiranja, sa posebnim naglaskom na Black box testiranje.

Drugo poglavlje, prikazuje osnovne vrste mobilnih aplikacija i razlike u njima.

Treće poglavlje, definiše principe i prikazuje metod Monkey testiranja native mobilnih aplikacija.

Četvrto poglavlje, opisuje alat za kreiranje automatizovanih testova, namenjenih za testiranje native mobilnih aplikacija po imenu Monkey Runner.

Peto poglavlje, sadrži praktične primere upotrebe Monkey Runner test alata sa ispisom programskog koda, upotrebljenog u testiranju i napisanog na programskom jeziku Python.

Šesto poglavlje je zaključak u kome su izneta zapažanja o upotrebljenim alatima i navedene moguće tendencije daljeg razvoja u oblasti automatizovanja procesa testiranja native mobilnih aplikacija.

1. Testiranje softvera

Proces testiranja obuhvata veliki broj aktivnosti i usko je povezan sa procesom razvoja softvera.

Testiranje se definiše kao proces evaluacije softvera i svih njegovih pratećih komponenti, kako bi se utvrdilo da li su kreirani u skladu sa zahtevima, da li odgovaraju svrsi za koju su namenjeni i da li postoje defekti (bagovi) u njima.

Bag (engl. Bug) ili defekt, se definiše kao pogrešno ponašanje softvera. On počinje od trenutka kada se pronađe greška u softveru i prestaje onda kada se ta greška otkloni.

1.1. Faze životnog ciklusa u testiranju softvera

Svaka kompanija kreira svoj životni ciklus u testiranju softvera. U opštem slučaju, životni ciklus u testiranju softvera obuhvata sledeće faze:

- 1) Analiza zahteva – Requirements analysis,
- 2) Planiranje testiranja – Test planing,
- 3) Razvoj test slučajeva – Test case development,
- 4) Izvršavanje test slučajeva – Test case executing,
- 5) Generisanje izveštaja o rezultatima testa – Test results reporting,
- 6) Generisanje izveštaja o defektima – Defects reporting,
- 7) Regresiono testiranje – Regression testing,
- 8) Zaključivanje testova – Test closure.

1.2. Proces testiranja

Tokom procesa razvoja softvera, potrebno je prepoznati kada treba početi a kada treba završiti testiranje.

- ◆ Početak testiranja – U životnom ciklusu razvoja softvera, testiranje može da počne od faze sakupljanja zahteva. Obično, od trenutka kada se ispuni ulazni kriterijum. Ulazni kriterijum (engl. Entry criteria) je minimalan set uslova koje treba ispuniti kako bi se počelo sa testiranjem.
- ◆ Završetak testiranja – Testiranje softvera može trajati sve do isporuke softvera krajnjim korisnicima. To zavisi od toga koji se model razvoja softvera koristi. Završetak testiranja se realizuje:
 - dostizanjem roka, u smislu kraja faze testiranja ili kraja celog projekta,
 - uspešnim završavanjem svih predviđenih test aktivnosti,
 - znatnim smanjenjem broja bagova,
 - odlukom menadžmenta.

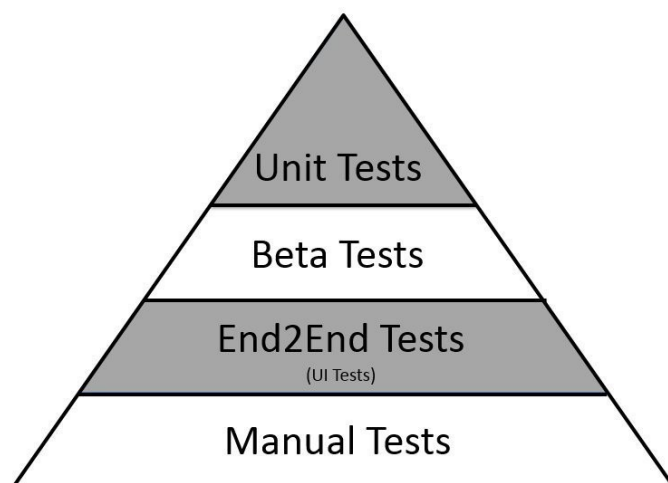
Drugim rečima, završetak testiranja, je definisan izlaznim kriterijumom. Izlazni kriterijum (engl. Exit criteria) je skup zahteva koji kada se ispune dozvoljavaju završetak testiranja.

U praksi, smatra se da je testiranje proces bez kraja, jer se ni za jedan softverski proizvod ne može reći da je u potpunosti 100% bez bagova (bug-free).

1.3. Mobilno testiranje

Mobilno testiranje zahteva da se većina testova izvodi manuelno. Ipak, postoje delovi procesa mobilnog testiranja koji mogu biti automatizovani. Testeri mobilnih aplikacija trebaju težiti da automatizuju deo procesa testiranja kako bi dobili brže povratne informacije o stanju aplikacije.

Piramida u mobilnom testiranju omogućava dobar miks manuellnog i automatizovanog testiranja i preporuka je da se on uzima u obzir pri planiranju testiranja mobilnih aplikacija.



Slika 1. Piramida u mobilnom testiranju

Najviši sloj u piramidi čini manuelno testiranje i ono predstavlja osnovu svakog projekta testiranja mobilnih aplikacija. Nakon manuellnog testiranja idu testiranja s kraja-na-kraj (end-to-end), beta testiranje i jedinično testiranje. Sloj beta testiranja je veoma važan, jer on omogućava da se ispune velika očekivanja krajnjih korisnika.

Sivi delovi piramide su testovi u kojima se mogu automatizovati test koraci. Beli delovi piramide podrazumevaju da se test koraci vrše manuelno.

1.4. Podele testiranja

Testiranje je višedimenzionalni proces koji možemo podeliti prema više aspekata I to prema: vrstama, nivoima, metodama i strategijama.

a) Vrste testiranja

- Funkcionalno testiranje,
- Testiranje opterećenja,
- Testiranje sigurnosti,
- Testiranje konfiguracije,
- Testiranje korisničkog interfejsa.

b) Nivoi testiranja

- Jedinično testiranje,
- Integraciono testiranje,
- Sistemsko testiranje.

c) Metode testiranja

- Manuelno testiranje,
- Automatsko testiranje,
- Testiranje po skriptovima,
- Regresivno testiranje,
- Istraživačko testiranje.

d) Strategije testiranja

- Black box,
- White box,
- Grey box,
- Uporedno testiranje.

1.5. Strategije testiranja

Strategije testiranja definišu na koji način će se vršiti testiranje. One određuju kako treba posmatrati sistem koji se testira I na koji način treba definisati test slučajeve. Primenjuju se na sve komponente testiranja.

- 1) Black box testiranje – je testiranje u kojoj se sistem posmatra kao jedinstvena celina koja vrši neku funkcionalnost I za neke ulazne vrednosti vraća određene rezultate. Tester u ovom pristupu nema pristup izvornom kodu (source code) same aplikacije. Ovo testiranje se vrši tako što se sistemu šalju neki ulazni podaci I proverava se da li je rezultat koji se vraća u skladu sa zahtevima, bez poznavanja interne strukture

sistema. Prednost ovog testiranja je velika efikasnost, nije potreban pristup izvornom kodu, jasno su razdvojene perspektive korisnika I developera. Mana ovog testiranja je ograničena pokrivenost po broju test scenarija, test slučajevi su komplikovani za dizajniranje.

- 2) White box testiranje – je strategija u kojoj se vidi unutrašnj stuktura sistema a testovi se prave tako da se proveriti da li određeni algoritam ili struktura rade. Ova strategija je detaljnija I zahteva mnogo više vremena za pravljenje testova. Ovde testeri moraju donekle imati zanje o samom izvornom kodu, moraju ga pregledati kako bi pronašli koji deo koda ne radi kako bi trebalo. Prednosti ove strategije je u tome što je moguća optimizacija koda, zahvaljujući poznavanju koda testeri mogu da ostvare bolju pokrivenost aplikacije pri testiranju, pišući bolje test scenarije. Mane su povećani troškovi, potreba za obučenijim testerima, teže realizovanje ovog testiranja jer su potrebni specijalni alati za testiranje.
- 3) Grey box testiranje – je strategija u kojoj se koristi ograničen nivo znanja o internom radu same aplikacije. Ovde testeri imaju pristup dokumentima za dizajniranje aplikacije i bazama podataka. Zahvaljujući tome, testeri mogu bolje da pripreme test scenarije i test podatke. Prednost ove strategije je što ona pruža prednosti I Black box I Grey box testiranja, testeri se ne oslanjaju na izvorni kod već na dokumentaciju, testovi se vrše sa aspekta korisnika a ne dizajnera. Mane se ograničena pokrivenost, testovi mogu postati redundantni, nije pogodno za algoritamsko testiranje.
- 4) Uporedno testiranje – je strategija u kojoj se testiranje vrši tako što se rezultati porede sa etalom ili sa drugim funkcionalnostima u sistemu u kojima se očekuje da će rezultati biti isti. Etalon je funkcionalnost za koju se sigurno zna da je tačna. Uporedno testiranje se često koristi u slučaju da se neki sistem zamenjuje novim sistemom. Pošto novi sistem mora da sadrži veliki broj funkcionalnosti iz starog sistema koje su samo re-implementirane, najbolji test koji se može izvršiti je poređenje sa starim sistemom.

2. Mobilno testiranje

Mobilno testiranje buhvata testiranje mobilnih uređaja i testiranje aplikacija koe se razvijaju za rad na mobilnim uređajima (Mobilne aplikacije).

Testiranje mobilnih uređaja (engl. mobile testing), podrazumeva testiranje mobilnih uređaja, kao što su pametni telefoni (engl. smartphone), tablet uređaji, ... i sl. Ono obuhvata testiranje hardvera, baterije, povezivanja urešaja sa mrežom, softverska kompatibilnosti za dati uređaj, testiranje protokola, ... i sl.

Testiranje mobilnih aplikacija (engl. mobile application testing), je proces provere funkcionalnosti, upotrebljivosti I konzistentnosti softverskih aplikacija razvijenih za mobilne uređaje. Ovde se teži da se verifikuje da li mobilne aplikacije ispunjavaju sve funkcionalne I nefunkcionalne zahteve.

2.1. Tipovi mobilnih aplikacija

Mobilne aplikacije koje se koriste u mobilnim uređajima se dele na:

- Native aplikacije,
- Web aplikacije,
- Hibridne aplikacije.

2.1.1. Native mobilne aplikacije

Native aplikacije su aplikacije za mobilne uređaje koje se skidaju sa Play Stora-a ili App Stor-a i instaliraju se na sam uređaj. One su pisane na nekom od bazičnih programskih jezika, poput: Java, C#, Swift, Objective C, ... i sl.

Ove aplikacije imaju pun pristup svim API-jima uređaja na kome se nalaze, a izvorni kod (engl. source code) ovih aplikacija radi samo na ciljanoj platformi za koju su i napravljene.

2.1.2. Web aplikacije

Veb aplikacije su zapravo web sajtovi dizajnirani isključivo za mobilne uređaje. One nekada pokušavaju da imitiraju dizajn Native aplikacija.

Ove aplikacije su pisane u HTML, CSS i JavaScript jezicima. Za njihovo korišćenje je neophodna konekcija sa internetom i pristupa im se preko Internet pretraživača (browsera). Veb aplikacije se ne mogu naći na Play Storu i na App Storu, zato što su to obični web sajtovi, samo prilagođeni za mobilne uređaje (npr. www.mobile.twitter.com).

2.1.3. Hibridne aplikacije

Hibridne aplikacije su aplikacije koje imaju Native osnovu sa ugnježđenim HTML kodom. One imaju osnovni pristup API-jima samog uređaja sa koga se koriste. Dostupne su na Play i App Stor-u. HTML sadržaj ovih aplikacija je smešten na serveru. To znači da veb deo ovih aplikacija može da se apdejtuje na serverskoj strani. Primer ovih aplikacija su: Instagram, Wikipedija, ... i sl.

3. Monkey testing

Monkey testing podrazumeva test metodu u testiranju native mobilnih aplikacija na Android platformi u kojoj se aplikacija podvrgava nizu nasumičnih akcija u cilju provere stabilnosti same aplikacije. Ova metoda se bazira na upotrebi monkey alata, razvijenog isključivo za testiranje mobilnih aplikacija razvijenih za rad na Android platformi.

Monkey tool generiše niz nasumičnih koraka koji obuhvataju simulaciju klaktnja i dodirivanja pojedinih elemenata na ekranu aplikacije kao što su klik na dugme, dodir na link, sliku ili tekstualno polje za unos podataka. Izvođenje ovih koraka se odvija velikom brzinom i praktično nekontrolisano, čime se aplikacija testira na način koji tester ne može da reprodukuje.

Prednost primene ovake metode u testiranju je ta što se aplikacija testira na rad u stresnom radnom scenariju, čime se praktično proverava stabilnost aplikacije i utvrđuje da li aplikacija može da doživi pad (engl. crash) ukoliko bi se koristila mnogo intenzivnije nego što je to predviđeno klijentskim zahtevima.

Testiranje metodom Monkey testing, se u praksi primenjuje tek nakon što je cela aplikacija potpuno istestirana i uklonjeni i korigovani svi pronađeni bagovi i defekti. Ukoliko se nakon uspešno sprovedenog Monkey testiranja, aplikacija pokazala kao stabilna i nije padala u toku testiranja, ona se može kao završen i stabilan proizvod isporučiti klijentu.

Prilikom Monkey testiranja obavljaju se sledeće aktivnosti:

1. Na računaru priključimo mobilni telefon ili uključimo emulator.
2. Uključimo cmd (Command prompt) i pomoću njega pokrećemo monkey testing.
3. Na terminalu pratimo ispis aktivnosti koje se odigravaju u toku testa i vodimo računa o eventualnoj pojavi pada aplikacije. Ispis je moguće i generisati odmah u tekstualni fajl.

Bolji opis aktivnosti, prikazan je u primeru broj 1.

Primer 1.

Potrebno je testirati stabilnost aplikacije ApParkigSpot, primenom Monkey testing metode.

Rešenje:

Koraci koje je potrebno izvršiti radi realizacije testiranja, obuhvataju:

1. Preko USB kabla priključimo mobilni telefon na kompjuter,
2. Uključim **cmd** (tj. terminal) na računaru,
3. Pomoću cmd-a se pozicioniram u folder u kome se nalazi **adb** aplikacija, (npr: C:\Users\Branislav\android-sdks\sdk\platform-tools)

4. Unesem komandu **adb devices**, kako bi mi komp. prepoznao priključeni telefon,
5. Ako već imamo ime .apk fajla aplikacije koju želimo da testiramo, i ako je ona već instalirana na mobilnom telefonu, onda u cmd unesemo sledeću komandu:

adb shell monkey -p ime_paketa -v 1500

6. Kliknem na **Enter**, nakon čega testiranje počinje.

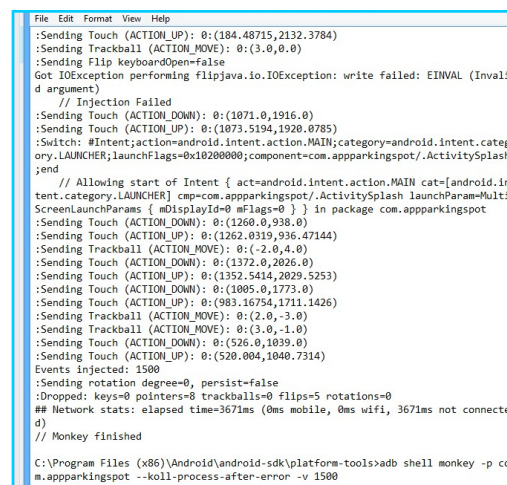
Komanda uneta u koraku 5, se sastoji od:

- **adb** – pali Android Debug Bridge, koji je deo Android SDK paketa,
- **shell** – poziv shell-a,
- **monkey** – pokretanje monkey alata,
- **-p** – je oznaka koja kaže da nakon nje sledi paket koji se testira,
- **ime_paketa** – je ime .apk fajla koji se testira, (npr. com.appparkingspot, io.selendroid.testapp, ... i sl.),
- **-v** – je oznaka koja ukazuje setovanje broja ponavljanja testa
- **1500** – je broj ponavljanja testa. Ovde može biti bilo koji ceo broj.

Ukoliko želimo da se ispis u cmd-u automatski sačuva/ispisše u nekom tekstualnom fajlu, onda moramo na kraj komande u koraku broj 5, da dodamo: > ime_fajla.txt. (npr. log01.txt). Tako da komanda iz koraka broj 5 postaje:

adb shell monkey -p ime_paketa -v 1500 > log01.txt

Primer dela ispisa iz log01.txt fajla je dat na slici 2.



```
File Edit Format View Help
:Sending Touch (ACTION_UP): 0:(184.48715,2132.3784)
:Sending Trackball (ACTION_MOVE): 0:(3.0,0.0)
:Sending Flip keyboardOpen=False
Got IOException performing Flipjava.io.IOException: write failed: EINVAL (Invalid argument)
// Injection Failed
:Sending Touch (ACTION_DOWN): 0:(1071.0,1916.0)
:Sending Touch (ACTION_UP): 0:(1073.5194,1920.0785)
:Switch: #Intent;action=android.intent.action.MAIN;category=android.intent.category.LAUNCHER;launchFlags=0x10200000;component=com.appparkingspot/.ActivitySplash;end
// Allowing start of Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.appparkingspot/.ActivitySplash launchParam=MultiScreenLaunchParams { mDisplayId=0 mFlags=0 } } in package com.appparkingspot
:Sending Touch (ACTION_DOWN): 0:(1260.0,938.0)
:Sending Touch (ACTION_UP): 0:(1262.0319,936.47144)
:Sending Trackball (ACTION_MOVE): 0:(-2.0,4.0)
:Sending Touch (ACTION_DOWN): 0:(1372.0,2026.0)
:Sending Touch (ACTION_UP): 0:(1352.5414,2029.5253)
:Sending Touch (ACTION_DOWN): 0:(1005.0,1773.0)
:Sending Touch (ACTION_UP): 0:(983.16754,1711.1426)
:Sending Trackball (ACTION_MOVE): 0:(2.0,-3.0)
:Sending Trackball (ACTION_MOVE): 0:(3.0,-1.0)
:Sending Touch (ACTION_DOWN): 0:(526.0,1039.0)
:Sending Touch (ACTION_UP): 0:(520.004,1040.7314)
Events injected: 1500
:Sending rotation degree=0, persist=false
:Dropped: keys=0 pointers=0 trackballs=0 flips=5 rotations=0
## Network stats: elapsed time=3671ms (0ms mobile, 0ms wifi, 3671ms not connected)
// Monkey finished
C:\Program Files (x86)\Android\android-sdk\platform-tools>adb shell monkey -p com.appparkingspot --kill-process-after-error -v 1500
```

Slika 2. Deo ispisa iz log fajla

Ukoliko ne znamo tačan naziv .apk fajla koji se nalazi u telefonu a želimo da pronađemo njegovo ime, u cmd-u se pozicioniramo u folder u kome se nalazi adb i unesemo komandu:

adb shell

Nakon toga će se pojaviti root staza iza znaka \$ ili # (zavisno od OS-a) i nakon toga, iza znaka \$ unesemo komandu:

pm list packages -f

Prethodna komanda daje spisak svih paketa, tj. svih .apk fajlova od svih aplikacija koje imamo instalirane na telefonu. Puno ime.pak fajla se nalazi sa desne strane znaka „=“.

Iz shell-a izlazimo pomoću komande: **exit**.

Ako želimo da nam se test proces monkey testinga prekine odmah u trenutku kada se pojavi greška, tj. padne aplikacija, potrebno je u okviru komande za pokretanje testa napisati sledeću komandu:

--kill-process-after-error

Tako da nam komanda iz koraka 5 izgleda ovako:

adb shell monkey --kill-process-after-error -p ime_paketa -v 1500

U suprotnom, ako želimo da nam se test proces ne prekine u toku njegovog izvršavanja, unutar komande za pokretanje, napisaćemo komandu:

--ignore-crashes

Tada nam komanda iz koraka 5 izgleda, ovako:

adb shell monkey --ignore-crashes -p ime_paketa -v 1500

Potrebno je napomenuti da je difoltno stanje po pitanji test procesa da se sam test proces neće prekinuti ukoliko padne aplikacija, čak iako se ne napiše komanda **--ignore-crashes** tako ta to treba imati na umu, jer monkey testing proces je teško kontrolisati I teško je njime upravljati, pa je moguće da u slučaju pada aplikacije, test proces počne da se izvršava ne nekoj drugoj od postojećih instaliranih aplikacija na samom mob. Uređaju.

Alat monkey tool, se razlikuje od alata Monkey Runner po tome što monkey tool generiše sekvencu potpuno nasumičnih I nekontrolisanih koraka, kojima se testira rad aplikacije, dok Monkey Runner zahteva pisanje Python skripte i s toga omogućuje punu kontrolu nad tokom aktivnosti pri testiranju aplikacije.

4. Monkey Runner alat za automatizovanje procesa testiranja

Monkey Runner je alat koji omogućava pisanje programa koji upravlja android uređajem i aplikacijama instaliranim na mobilnom uređaju.

Monkey Runner poseduje API koji omogućava upravljanje aplikacijom instaliranom na telefonu, bez korišćenja ili izmene programskog koda same aplikacije.

Monkey Runner omogućava sledeće:

- pisanje Python koda za testiranje aplikacija na jednom ili više uređaja ili emulatora,
- instaliranje aplikacije direktno na telefon,
- puštanje aplikacije u rad,
- slanje događaja, tj. pokretanje aktivnosti kao što su: klik na dugme, dodir (engl. touch event) ili povlačenje preko ekrana (engl. swipe event),
- slikanje ekrana (engl. screenshot),
- čuvanje slikanih screenshot-ova u željenom fajlu na ekranu,
- automatsko kreiranje i čuvanje dnevnika (engl. log) sa izveštajima o izvršenim testovima.

Monkey Runner je pre svega namenjen za testiranje aplikacija na funkcionalnom nivou, tj. za izvođenje Jediničnih testova, Funkcionalnih testova i za Regresiono testiranje.

Prednosti Monkey Runner-a:

- mogućnost izvođenja testova na više uređaja istovremeno,
- izvođenje testova uz istovremeno slikanje ekrana,
- jednostavno regresiono testiranje,
- mogućnost proširenja i nadogradnje već postojećih automatskih testova (tj. skripti).

Monkey Runner se bazira na korišćenju Jython-a, hibridnog jezika koji predstavlja implementaciju programskog jezika Python u kombinaciji sa Java programskim jezikom.

4.1. Moduli alata

Moduli sadrže metode koje omogućuju rad i upravljanje sa mobilnim uređajem i aplikacijom instaliranom na njega. Moduli su praktično biblioteke koje sadrže metode namenjene za izvršavanje određenih aktivnosti.

Postoje tri osnovna modula koji omogućavaju rad Monkey Runner alata:

1. MonkeyRunner modul,
2. MonkeyImage modul,
3. MonkeyDevice modul.

4.1.1 MonkeyRunner

Monkey Runner je modul, koji sadrži statičke metode za prikaz poruke, dijaloga, za pauziranje izvršavanja programa i tome slično. Pored toga, poseduje i metode za kreiranje korisničkog interfejsa (UI) kojim korisnik komunicira sa napisanim test programom i metode za uspostavljanje komunikacije sa mobilnim uređajem ili emulatorom.

Metode u modulu MonkeyRunner su:

- **alert()** - je metoda koja omogućava da Monkey Runner otvori mali alert dijalog prozor. Koriti se u formi: **MonkeyRunner.alert("Aplikacija pokrenuta...")**,
- **sleep()** - je metoda koja pauzira izvršenje programa (test skripte) određeni broj sekundi. Koristi se u formi: **MonkeyRunner.sleep(8)**,
- **input()** - je metoda koja otvori mali alert dijalog prozor i omogućava korisniku da unese neki podatak u vidu string-a. Potom program to dalje upotrebi u skladu sa metodama napisanim u test skripti. Koristi se u formi:
Y=MonkeyRunner.input("Unesi broj ponavljanja: ", "0"),
- **waitForConnection()** - je metoda kojom MonkeyRunner uspostavlja konekciju sa mobilnim uređajem ili sa emulatorom. Obično se smešta u premenljivu device. Foma upotrebe je: **device=MonkeyRunner.waitForConnection()**,
- **help()** - je metoda za davanje API reference za pomoć u vezi neke komande i sl.,
- **choice()** - je metoda koja omogućava korisniku da zada neke opcije, koje potom udi korisniku u vidu padajućeg menija. Može se koristiti za zadavanje broja iteracija.

4.1.2. MonkeyImage

MonkeyImage je modul koji sadrži metode za snimanje screenshot-ova ekrana, njihovo čuvanje i rad sa njima.

MonkeyImage modul omogućava:

- slikanje screenshot-ova,
- čuvanje slika (screenshot) u željenom formatu i naznačenom fajlu,
- poređenje snimljne slike sa drugim objektima istog tipa, tj. slikakam istog formata.

Metode u MonkeyImage modulu su:

- **takeSnapshot()** - je metoda koja se koristi za snimanje screenshot-ova ekrana, tj. za njegovo slikanje. Koristi se u formi: **slika1=device.takeSnapshot()**, i potom, **slika1.writeToFile('C:\Users\manojlovic\Desktop\slika1.png', 'png')**,
- **convertToBytes()** - je metoda koja primljenu promenljivu u kojoj je smestena slika (.png, .jpeg) konvertuje u niz bajtova. Koristi se u formi: **m=MonkeyImage.convertToBytes(slika1)**,
- **getRawPixel()** - je metoda koja vraća pojedinačan piksel sa određene slike pomoću (x, y) koordinata u formi (a, R, G, B),
- **getRawPixelInt()** - je metoda koja kao rezultat daje poziciju određenog pixel-a sa slike pomoću (x, y) koordinate. Metode **getRawPixel()** i **getRawPixelInt()** se koriste kada se pri testiranju kreira jako mnogo slika (screenshot-ova), pri čemu je potrebno da se one sačuvaju a istovremeno i da se uštedi na memorijskom prostoru.
- **getSubImage()** - kreira nov MonkeyImage objekat, tj. novu sliku na osnovu kvadratne selekcije sa postojeće slike,
- **sameAs()** - je metoda za poređenje dve slike, tj. dva MonkeyImage objekta. Ona vraća kao odgovor TRUE ili FALSE boolean vrednost,
- **writeToFile()** - je metoda za upisivanje, tj. čuvanje postojeće (već kreirane) slike u fajl preciziran u datoj metodi. Koristi se u formi: **slika1.writeToFile('C:\Users\manojlovic\Desktop\slika1.png', 'png')**.

4.1.3. MonkeyDevice

MonkeyDevice je modul koji se odnosi na uređaj na kome se vrši testiranje. On sadrži metode za instaliranje i deinstaliranje aplikacije na mobilni uređaj (.apk fajla), startovanje preciziranog aktivitija (željene aktivnosti u aplikaciji), slanja aktivnosti kao što su klik na dugme, dodir na ekran i sl. Pomoću ovog modula se praktično puštaju u rad testovi koji se napišu i kontrolišu se uređaju ili emulatori koji se koriste.

Metode u modulu MonkeyDevice su:

- **waitForConnection()** - za kreiranje novog objekta koji će predstavljati uređaj ili emulator. Koristi se u formi: **device=MonkeyRunner.waitForConnection()**,
- **touch()** - za simuliranje dodira na određeni element na ekranu. Koristi se u formi: **device.touch(200, 390, "DOWN_AND_UP")**, gde prva dva broja predstavljaju koordinate tačke (elementa) na ekranu koji treba da se dodirne, a string "DOWN_AND_UP" je konstanta kojom se definiše da će se element prvo pritisnuti a potom otpustiti.
- **press()** - simulira klik na određeni element na ekranu. Kao atribut mu se šalje ime elementa. Primer forme je: **device.press('KEYCODE_MENU', MonkeyDevice.DOWN_AND_UP)**.
- **reboot()** - je metoda koja restartuje telefon.
- **removePackage()** - je metoda koja deinstalira .apk fajl, sa telefona, tj. Deinstalira aplikaciju,
- **shell()** - je metoda koja pali i izvršava adb shell komande,
- **startActivity()** - je metoda koja startuje onu aktivnost koja se navede u njoj. Ta aktivnost može biti na primer splash screen ili tome slično. Primer forme upotrebe je: **device.startActivity(component='io.selendroid.testapp/.HomeScreenActivity')**,
- **wake()** - je metoda koja pobuđuje uređaj, tj. budi ga iz Stand by režima,
- **type()** - je metoda koja omogućava unošenje/upis string-a karaktera u tekstualne forme na ekranu. Veoma je dobra za testiranje unosa lozinke ili email-a i sličnih podataka na "Log in" ili "Register" ekranima aplikacija. Primer upotrebe je dat u formi: **device.type("nekimejl@gmail.com")**, **device.type("lozinka")**.
- **takeSnapshot()** - uzima, tj. snima screenshot ekrana,
- **installPackage()** - instalira .apk fajl na uređaj ili emulator.
- **drag()** - je metoda koja kreira "swipe" događaj. Forma je: **device.drag((110, 70), (250, 50), 1.0, 10)**. Gde su (110, 70) i (250, 50) početna i završna koordinata a 1.0 je trajanje "swipe" događaja.

5. Primeri

1) Primer za uspostavljanje konekcije sa uređajem ili emulatorom.

```
# Imports - importuje potrebne module
from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice, MonkeyImage

# Connection - uspostavlja konekciju sa mobilnim uređajem ili emulatorom
device = MonkeyRunner.waitForConnection()
print("Test zapocet")

# definise koji paket instaliran na mobilnom uređaju se pokrece
package = 'com.appparkingspot'
activity = '.ActivitySplash'
runComponent = package + '/' + activity

# ispis kratkih poruka na konzoli
print('Launching app')
print(runComponent)

# pokrece, tj. lansira aplikaciju
device.startActivity(component = runComponent )

# pauzira izvršenje test skripte na 6 sekundi
MonkeyRunner.sleep(6)
# Deploy your app, add your intent, if needed
```

Slika 3. Test skripta u primeru 1

2) Primer upotrebe metoda za uzimanje screenshot-a i cuvanje istog u fajl.

```
# imports
from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice, MonkeyImage
# Connection
device = MonkeyRunner.waitForConnection()

# Setuje/podesava path varijablu sa punim imenom .apk fajla
# i pokrece odabranu komponentu
device.startActivity(component='io.selendroid.testapp/.HomeScreenActivity')
MonkeyRunner.sleep(8)

# Uzima/slika screenshot 1
slika1 = device.takeSnapshot()
# Memorise screenshot u preciziran fajl
slika1.writeToFile('C:\Users\manojlovic\Desktop\slika1.png','png')
# poruka
print "Slika napravljena i snimljena"
MonkeyRunner.sleep(5)
# while petlja koja zadaje koliko puta treba ponoviti test
x = 1
while (x <= 5):
    print "Krug: ", x
    x = x + 1
    #klik na EN Button
    device.touch(362, 136, "DOWN_AND_UP")
    #klik na No, no
    device.touch(352, 450, "DOWN_AND_UP")
    MonkeyRunner.sleep(5)
    #uzimam screenshot 2
    slika2 = device.takeSnapshot()
    slika2.writeToFile('C:\Users\manojlovic\Desktop\slika2.png','png')
    MonkeyRunner.sleep(5)

print('Kraj testa!')
```

Slika 4. Test skripta u primeru 2

3) Primer za touch() i drag() metode.

```
# Imports
from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice, MonkeyImage
# Connection
device = MonkeyRunner.waitForConnection()
print("Test04 zapocet\n")
# Setuje/podesava path varijablu sa punim imenom .apk fajla
package = 'com.appparkingspot'
activity = '.ActivitySplash'
runComponent = package + '/' + activity
# Pokrece (Run) komponentu
device.startActivity(component=runComponent)
# Pauzira izvršenje programa na 3 sekunde da bi se aktivnost ucitala
MonkeyRunner.sleep(8)

x = 1
while (x <= 2):
    print "Krug: ", x
    x = x + 1
    print('Klik na meni')
    device.touch(144, 180, "DOWN")
    MonkeyRunner.sleep(8)
    print('Klik na Event')
    device.touch(414, 636, "DOWN")
    MonkeyRunner.sleep(6)
    print('Klik na meni')
    device.touch(144, 180, "UP")
    MonkeyRunner.sleep(6)
    print('Klik na My spaces')
    device.touch(432, 810, "UP")
    MonkeyRunner.sleep(8)
    print('Skrolovanje')
    device.drag((360, 2352), (360, 696), 1.0, 10)
    device.drag((360, 696), (360, 2352), 1.0, 10)
    device.drag((360, 2352), (360, 696), 1.0, 10)
    device.drag((360, 696), (360, 2352), 1.0, 10)
    device.drag((360, 2352), (360, 696), 1.0, 10)
    device.drag((360, 696), (360, 2352), 1.0, 10)
    MonkeyRunner.sleep(8)

print('\nKraj testa')
```

Slika 5. Test skripta u primeru 3

4) Primer za swipe() evente

```
from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice, MonkeyImage
# Connection
device = MonkeyRunner.waitForConnection()
print('\nTest - Testiranje tutorijala')0
# Pokrece komponentu
device.startActivity(component='com.appparkingspot/.ActivitySplash')
MonkeyRunner.sleep(8)
x = 1
while (x <= 5):
    print "Krug: ", x
    x = x + 1
    print('Klik na meni')
    device.touch(144, 180, "DOWN")
    MonkeyRunner.sleep(8)
    print('Klik na tutorijal')
    device.touch(420, 2262, "DOWN_AND_UP")
    MonkeyRunner.sleep(8)
    print('Swipe s desna na levo tri puta')
    device.touch(198, 2340, "DOWN_AND_UP")
    MonkeyRunner.sleep(3)
    device.drag((1302, 1674), (144, 1668), 0.1, 10)
    MonkeyRunner.sleep(1)
    device.drag((1302, 1674), (144, 1668), 0.1, 10)
    MonkeyRunner.sleep(1)
    device.drag((1302, 1674), (144, 1668), 0.1, 10)
    MonkeyRunner.sleep(1)
    print('Swipe s leva na desno tri puta')
    device.drag((144, 1668), (1302, 1674), 0.1, 10)
    MonkeyRunner.sleep(1)
    device.drag((144, 1668), (1302, 1674), 0.1, 10)
    MonkeyRunner.sleep(1)
    device.drag((144, 1668), (1302, 1674), 0.1, 10)
    MonkeyRunner.sleep(1)
    print('Swipe s desna na levo tri puta')
    device.touch(1236, 2340, "DOWN_AND_UP")
    MonkeyRunner.sleep(3)
    device.drag((1302, 1674), (144, 1668), 0.1, 10)
    MonkeyRunner.sleep(1)
    device.drag((1302, 1674), (144, 1668), 0.1, 10)
    MonkeyRunner.sleep(1)
    device.drag((1302, 1674), (144, 1668), 0.1, 10)
    MonkeyRunner.sleep(3)
    print('klik na dugme')
    device.touch(702, 2316, "DOWN_AND_UP")
    MonkeyRunner.sleep(8)
    device.touch(144, 180, "DOWN")
    MonkeyRunner.sleep(3)

print('Kraj testa!')
```

Slika 6. Test skripta u primeru 4

5) Primer za komunikaciju sa korisnikom i koriscenje while petlje za ponavljanje testa.

```
# Imports
import sys
import os
from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice, MonkeyImage

# Connection
device = MonkeyRunner.waitForConnection()

print('\nTest - Rad sa modulima')
print("Test zapocet\n")

# Setuje/podesava path varijablu sa punim imenom .apk fajla
package = 'com.appparkingspot'
activity = '.ActivitySplash'
runComponent = package + '/' + activity

# Pokrece (Run) komponentu
device.startActivity(component=runComponent)

#ispisuje poruku o pokrenutoj aplikaciji
MonkeyRunner.alert("Aplikacija je pokrenuta")

# Pauzira izvršenje programa na 3 sekunde da bi se aktivnost ucitala
MonkeyRunner.sleep(8)

y = MonkeyRunner.input("Unesite broj ponavljanja testa: ", "0")
print "Broj ponavljanja: ", y
#konvertujem String vrednost y =1 u int vrednost y = 1
y = int(y)

x = 1
while (x <= y):
    print "Krug: ", x
    x = x + 1
    print('Klik na meni')
    device.touch(144, 180, "DOWN")
    MonkeyRunner.sleep(8)
    device.takeSnapshot()
    print('Klik na Event')
    device.touch(414, 636, "DOWN")
    MonkeyRunner.sleep(6)
    print('Klik na meni')
    device.touch(144, 180, "UP")
    MonkeyRunner.sleep(6)

#ispisuje poruku o zavrшеноj aplikaciji
MonkeyRunner.alert("Cestitamo uspesno ste završili Monkey Runner Test!!!")
print("Test završen")
```

Slika 7. Test skripta u primeru 5

6) Primer za testiranje Log in forme i popunjavanje text field-ova

```
# Imports
from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice, MonkeyImage
# Connection
device = MonkeyRunner.waitForConnection()
# pobudjuje uredjaj
device.wake()
MonkeyRunner.sleep(8)
print('\nTest - Testiranje Type Metode')
MonkeyRunner.sleep(8)
# Setuje/podesava path varijablu sa punim imenom .apk fajla
package = 'com.appparkingspot'
activity = '.ActivitySplash'
runComponent = package + '/' + activity
# Pokrece (Run) komponentu
device.startActivity(component=runComponent)
#ispisuje poruku o pokrenutoj aplikaciji
MonkeyRunner.alert("Aplikacija je pokrenuta")
# Pauzira izvršenje programa na 3 sekunde da bi se aktivnost ucitala
MonkeyRunner.sleep(5)
x = 1
while (x <= 1):
    print "Krug: ", x
    x = x + 1
    # kliknuo sam na Email address
    device.touch(354, 1092, "DOWN_AND_UP")
    device.type("banelmanojlovic@gmail.com")
    MonkeyRunner.sleep(3)
    # klik na password field
    device.touch(336, 1320, "DOWN_AND_UP")
    device.type("BakiMaki1")
    MonkeyRunner.sleep(3)
    # klik na done da spusti tastaturu
    device.touch(1284, 2364, "DOWN_AND_UP")
    MonkeyRunner.sleep(3)
    # klik na Log in dugme
    device.touch(414, 1614, "DOWN_AND_UP")
    MonkeyRunner.sleep(8)
    print('Klik na meni')
    device.touch(144, 180, "DOWN")
    MonkeyRunner.sleep(8)
    #klik na settings
    device.touch(414, 1896, "DOWN_AND_UP")
    MonkeyRunner.sleep(5)
    #klik na LogOut
    device.touch(702, 2268, "DOWN_AND_UP")
    MonkeyRunner.sleep(5)

print('Kraj testa!')
```

Slika 8. Test skripta u primeru 6

6. Zaključak

Prikazani alat Monkey Runner, koji se upotrebljava za automatsko testiranje mobilnih aplikacija razvijenih na Android platformi, predstavlja dopunsko sredstvo kojim se može upotpuniti testiranje native mobilnih aplikacija i kao takvog ga treba i koristiti.

U praksi je slučaj da se Monkey testing metoda i konkretna primena Monkey Runner alata vrši na samom kraju procesa testiranja aplikacije, kada su već svi ozbiljniji bagovi otkriveni i korigovani. Pa se onda monkey testingu, pristupa kao poslednjoj grupi tesova koji će imati za cilj da provere stabilnost aplikacije.

Native mobilne aplikacije se pre svega moraju testirati manuelno i zbog toga je primena Monkey Runner alata, u njihovom testiranju veoma ograničena.

Tendencije u oblasti testiranja native mobilnih aplikacija se svakako kreću u pravcu delimičnog automatizovanja delova test slučajeva, kako bi se olakšao proces testiranja, ali još uvek kao primarni način testiranja native mobilnih aplikacija ostaje manuelno testiranje.

Literatura

- [1] Popović J., “Testiranje softvera u praksi”, Računarski fakultet, Beograd, 2012.
- [2] Knott D.,: “Hands-on Mobile App Testing”, eBook, Indiana USA, 2015.
- [3] <https://developer.android.com/studio/test/index.html> sajt “Android Developers”.
- [4] <http://blogs.wittwer.fr/whiler/2011/07/01/editeur-monkeyrunner/> blog Wittwer William