Tartu Ülikool
Loodus- ja tehnoloogiateaduskond
Arvutitehnika

Igor Poomre

Android OS rakenduste automaattestid Robotium raamistikul

Bakalaureusetöö

Juhendaja: Marko Peterson, M.Sc.

Tartu 2013

SISUKORD

SISSEJUHATUS

Kaasaegne mobiiltelefon suudab katta peaaegu kõik lauaarvuti funktsioonid, olles samal ajal füüsiliselt väike ning seetõttu kerge kaasas kanda. Seetõttu pole üllatav, et ka mobiilirakenduste arenduse turul on toimunud hoogustumine. Rakendused muutuvad aina keerulisemaks ja mitmekülgsemaks, mille tõttu muutub järjest olulisemaks ka nende testimisprotsess ja selle kvaliteet. Testimise mahtude kasvu tõttu, ei ole üllatav, et paljud ettevõtted on hakanud järjest enam automaattestimisele tähelepanu pöörama. Just testide automatiseerimine võimaldab säästa aega ja vahendeid ning muuta testimisprotsessi paremaks, mis on eriti oluline kvaliteedi seisukohast. Kasutades automaattestimist, on võimalik automatiseerida kõik lihtsad ja pidevalt korduvad tegevused. Seda on võimalik kasutada mitte ainult funktsionaalse testimise vaid ka regressioonitestimise puhul.

Käesolevas bakalaureusetöös tehakse ülevaade Android OS rakenduste testimisel kasutatavast Robotium raamistikust, mis võimaldab kirjutada automaatiseeritud teste. Lisaks loodi testrakendus, mille puhul testiti Android OS'le arendatud loodusgiidi rakendust, mille käigus kirjutati hulk teste, kasutades Robotium'i raamistikku.

2 MOBIILTEHNOLOOGIAD

Mobiiltelefonide rakendusi hakati laialdaselt kasutama rohkem kui kümme aastat tagasi ja selle perioodi jooksul on need muutunud tootajate poolt arendatud põhifunktsioonidest rohkem kompleksete rakenduste poole. Need rakendused loodi enamasti äriprotsesside, igapäevaelu ja meelelahutuse jaoks. Mobiilirakendused hõlmavad järgmisi valdkondi:

- Siderakendused – sotsiaalvõrgud, e-post, Skype, MSN, jne.
- Inforakendused – sõnastikud, navigaatorid, infogiidid.
- Rakendustarkvara – rakendused, mis aitavad töödelda teksti, pilte, jmt.
- Meelelahutus – video ja audio failide mahamängimine, mängud, audioramaatud, jne.

Järgnevalt on välja toodud mõned põhjused, miks mobiilirakendused said nii populaarseks:

- Juurdepääs: tänapäeva mobiiltelefon võimaldab kasutajal olla Internet levalasi peaaegu kõikjal. See annab pideva võimaluse kontrollida e-posti, lugeda uudiseid, juhtida äri, jne.
- Efektiivsus: mobiilsed rakendused on suurendanud Interneti kasutajate igapäevast tegevuste hulka. Iga rakenduse peamine eesmärk on viia kasutaja otse lõppressurssini. Seetõttu säästavad mobiilsed rakendused aega ja annavad otsese juurdepääsu asjadele, mida kasutajad soovivad.
- Kohanemisvõime: üldine veendumuse põhjal kasutavad mobiilseid rakendusi enamasti lapsed ja tavakasutajatest täiskasvanud, kuid ei ole see päris tõene. Olulise osa kogu rakenduste kasutajatest moodustavad jätkuvalt ärikliendid, kes on ka paljude rakenduste esmakasutajad, õppides tihti uusi tehnoloogiaid [1].

Rakenduste kasutamisel on suur osa ka erinevatel operatsioonisüsteemidel, mis on oma omaduste ja funktsionaalsuse poolest üldjoontes sarnased, pürgides maksimaalselt hea lõppkasutaja kogemuse poole. Siiski leidub kõikidel platvormidel ka miinuseid, mis võivad tihti määrata selle, millistele meeldib arendajatel kõige rohkem rakendusi arendada.

2.1 Mobiilsed operatsioonisüsteemid

Mobiilne operatsioonisüsteem kontrollib mobiiltelefoni, tahvelarvutit või muud mobiilset seadet. Tänapäeval mobiilne operatsioonisüsteem ühendab endas lauaarvuti funktsioone koos mobiilsidevõrguga, puutetundliku ekraani, WiFi, GPS, kaamera, videokaamera jmt riistvaraga. Alljärgnevalt peatutakse levinumatel mobiilsetel operatsioonisüsteemidel.

2.1.1 Apple iOS

Apple iOS (tuntud ka kui iPhone OS) on mobiilne operatsioonisüsteem, mille töötas välja Ameerika firma Apple. See põhineb Mac OS X-il ja arendati algselt iPhone jaoks, ning seejärel oli laiendati Apple iPod Touch, iPad ja Apple TV seadmetele. Apple ei luba üldjuhul iOS süsteemi paigaldada kolmandate osapoolte tarkvara. iOSis on 4 mõttelist kihti: tuumikkiht, tuumikteenuskiht, meediakiht ja *Cocoa Touch* kiht.[5].

2.1.2 Google Android OS

Android OS on elektroonikaseadmete tarkvarasüsteem, mis hõlmab operatsioonisüsteemi vahetarkvara ja peamisi rakendusi. Algselt oli see arendatud Android Inc poolt, mille Google hiljem ostis. Koos riistvara ja tarkvara arendajadega (nagu Intel, HTC, ARM ja eBay) käivitas Google Open Handset Alliance (OHA), mis praegu tegeleb platvormi toetamisega ja arendusega.[6].

2.1.3 BlackBerry OS

BlackBerry OS töötab mobiilsetes e-postiseadmetes ja nutitelefonides, mida toodetakse Kanada ettevõtte Research In Motion(RIM) poolt alates aastast 1999. Operatsioonisüsteemi oli algselt mõeldud ettevõtete spetsialistide jaoks. BlackBerry on peamiselt tuntud tänu võimele saata ja vastu võtta e-kirju ning sõnumeid, säilitades kõrge turvalisuse tänu sisseehitatud krüpteerimise funktsioonile.[4].

2.1.4 Windows Phone

Microsoft avalikustas oma uue põlvkonna mobiiltelefoni süsteemi Windows Phone 7 2010. aastal. Windows Phone võimaldab kolmanda osapoole tarkvaraarendust. Seda operatsioonisüsteemi on kritiseeritud sellepärast, et kasutajaliides ei ole optimeeritud sisestamiseks sõrmede vajutamise teel, vaid see on rohkem kasutatav koos digitaalse pliiatsiga (*stylos*). Windows Phone kasutab Microsofti uut kasutajaliidest nimega Metro, mis ühendab endas nii Microsofti enda teenused kui ka kolmanda osapoole tarkvara nagu Zune, Xbox Live and Bing.[7].

3 TESTIDE AUTOMATISEERIMINE

Testimisprotsessid sisaldavad sageli palju rutiinset käsitööd. Eriti suur on sellise käsitöö osa näiteks regressiooni testimisel ja seda on mugavam teha automatiseeritult. Kuid automaattestide mõte ei ole asendada käsistsi testimist, vaid muuta testimisprotsessi paremaks ja kvaliteetsemaks. Automatiseerimine võib vähendada inimeste osalust või koostoimet korduvate lihtsate ülesannete puhul. Järgmisi testimistüüpe on võimalik automatiseerida:

- Funktsionaalne testimine – kas toimingud toimivad ootuspäraselt.
- Regressioonitestimine – igat tüüpi tarkvara testimine, mida kasutatakse peale koodi/süsteemi viidud muudatusi.
- Stresstestimine – eesmärk on tuvastada kriitilisemad kohad, kus võib tekkida ülekoormus ja kulutada aeg nende kohtade optimeerimiseks.
- Erandjuhtude testimine – eesmärk on süsteemis veatingimusi esile kutsuda.

All on toodud peamised automatiseerimise eelised:

- Ajasääst – võimalus testskripte öösel käivitada, et hommikuks oleks kõik tulemused olemas.
- Võimalus käivitada skripte korraga mitme telefoni peal (väga oluline mobiilitelefonide puhul)
- Automatiseeritud testid jõuavad tulemuseni kiiremini kui testija.
- Enamik funktsionaalset testimist on võimalik automatiseerida.
- Kvaliteedi parandamine – testimise täpsust on võimalik suurendada väga tehniliste testskriptide abil, mida käsitsitestimisel ei ole alati võimalik saavutada.
- Annab korduvkasutuse võimaluse iga kord, kui rakendust muudetakse (regressioonitestimine).
- Rohkem katvust – korduv läbiviimine võimaldab leida vigu, mis käsitsi testimisel ei suudata leida.
- Rakenduse erinevad versioonid võivad kasutada samu automatiseeritud teste koos väikeste muudatusega.

Testide automatiseerimisel on ka puudusi:

- Testide kirjutamiseks on vajalik eelnev kogemus.
- Testide hooldus võtab mõnel juhul liiga palju aega, isegi väikesed kasutajaliideste või funktsionaalsuse muutused vajavad palju ümberkirjutamist.

- Kui testskriptides leidub vigu, on testimise kvaliteet väga kehv.
- Väikeste vigade vahelejätmine – skript jätab vahele väiksed vead, mida sellel pole programmeeritud kontrollida. Näiteks ebatäpsused aknede positsooneerimisel, graafilised vead.
- Ei sobi lokalisatsiooni testimiseks.

Automatiseerimine on oma plussidega ja miinusega laialdaselt kasutusel paljudelettevõtetel. Kuid käsitsi testimine on endiselt peamine testimistüüp, Automatiseerimine ainult suurendab testimisprotsessi kiirust ja lühendab testimise elutsükli [3].

Mobiilse operatsioonisüsteemi Android OS puhul on võimalik kasutada automaattestide raamistikku Robotium. Antud raamstikku on lihtne kasutada kogu tarkvaraprojekti elutsükli jooksul. Robotium võimaldab luua Android rakenduste jaoks *automated build* ja *test environment*'e. Kõik parameetrid ja konfiguratsioonid saab määrata *build.xml* failis. See annab võimaluse ehitada ja testida oma rakendusi paljudes koosseisudes, jooksutada teste realse seadmete peal või kasutada emulaatorit. Robotiumi on mõtet kasutada *Continuous Integration* puhul. See annab näiteks võimaluse pärast iga koodi *commit*'i käivitada vajalikku testide hulka öösel. Robotiumi rakendamise näitest tuleb juttu järgnevates peatükkides.

4 ANDROID OS

Android OS on uudne mobiilne operatsioonisüsteem, mis on mõeldud üsna võimsa mobiilse riistvara jaoks. Nii Windows Phone kui ka Apple iOS pakuvad mobiilrakenduste loomiseks rikkamat ja lihtsustatud arenduskeskkonda. Aga need on mõlemad ehitatud operatsioonisüsteemidel, mis mõnel juhul eelistavad kohalikke rakendusi nendele, mis on loodud kolmandate isikute poolt. Nad piiravad sidet rakenduste ja telefoni kohalike andmete vahel. Android aga pakub avatud arenduskeskkonda, mis on ehitatud kasutades avatud lähtekoodiga Linux *kernel*. Androidis on kõik rakendused võrdses seisus. Kohalikud ja kolmanda osapoole rakendused on välja arendatud kasutades sama API-t. Kasutaja saab asendada ükskõik millise kohaliku rakenduse kolmanda osapoole rakenduse vastu. See annab suurepärase võimaluse Android'i arendajatele luua liideseid ja rakendusi, nii nagu nad tahavad, ilma igasuguste piiranguteta[1].

4.1 Android OS arenduskeskkond

Android'i rakenduste loomiseks vajavad arendajad Android SDK-d ja Java Development Kit'i. Lisaks on vaja ka Java IDE-t - näiteks Eclipse, mis muudab arendamise palju lihtsamaks. Java SDK ja Eclipse viimased versioonid on kättesaadaval kõikide operatsioonisüsteemide jaoks, nii et iga arendaja saab valida arenduseks oma lemmik-platvormi. Android SDK tööriistad ja emulaator on kõikides keskkonnades hästi toimiv. Android'i rakenduse kood kirjutatakse Java süntaksit kasutades ja Android tuuma teegid sisaldavad enamikke funktsioone, mida toetab Java API. SDK sisaldab kõiki Android'i teeke, täielikku dokumentatsiooni ja näiteid. See hõlmab ka vahendeid, mis aitavad kirjutada ja siluda rakendusi, nagu Android'i emulaator ja Dalvik Debug Monitoring Service (DDMS) silumiseks (*debugging*).

4.2 Arenduskeskkonna ülesseadmine

Nagu eelnevalt mainitud töötavad Androidi rakendused Dalvik virtuaalse masina peal, mistõttu on arendajal võimalus valida ükskõik milline platvorm. Näiteks:

- Microsoft Windows (alates XP)
- Mac OS X 10.4.8 or later (Intel ainult)
- Linux

Selleks, et alustada testide kirjutamist, on vaja laadida ja paigaldada järgmised vahendid:

- *Java Development Kit (JDK) 6 or 7*
- *Eclipse IDE*
- *Android SDK*
- *Android development Tools (ADT)*
- *Robotium library*

4.2.1 Java keskkonna paigaldus

Korrektse Java versiooni kontrollimiseks tuleks avada konsool ja kävitada käsk: *java-version*, mis annab näiteks alljärgneva tulemuse:

java version "1.6.0_26"
Java(TM) SE Runtime Environment (build 1.6.0_26-b03)
Java HotSpot(TM) Server VM (build 20.1-b02, mixed mode)

Kui Java pole paigaldatud, siis saab selle alla laadida aadressilt:
http://www.oracle.com/technetwork/java/javase/downloads/index.html.

4.2.2 Eclipse IDE paigaldus

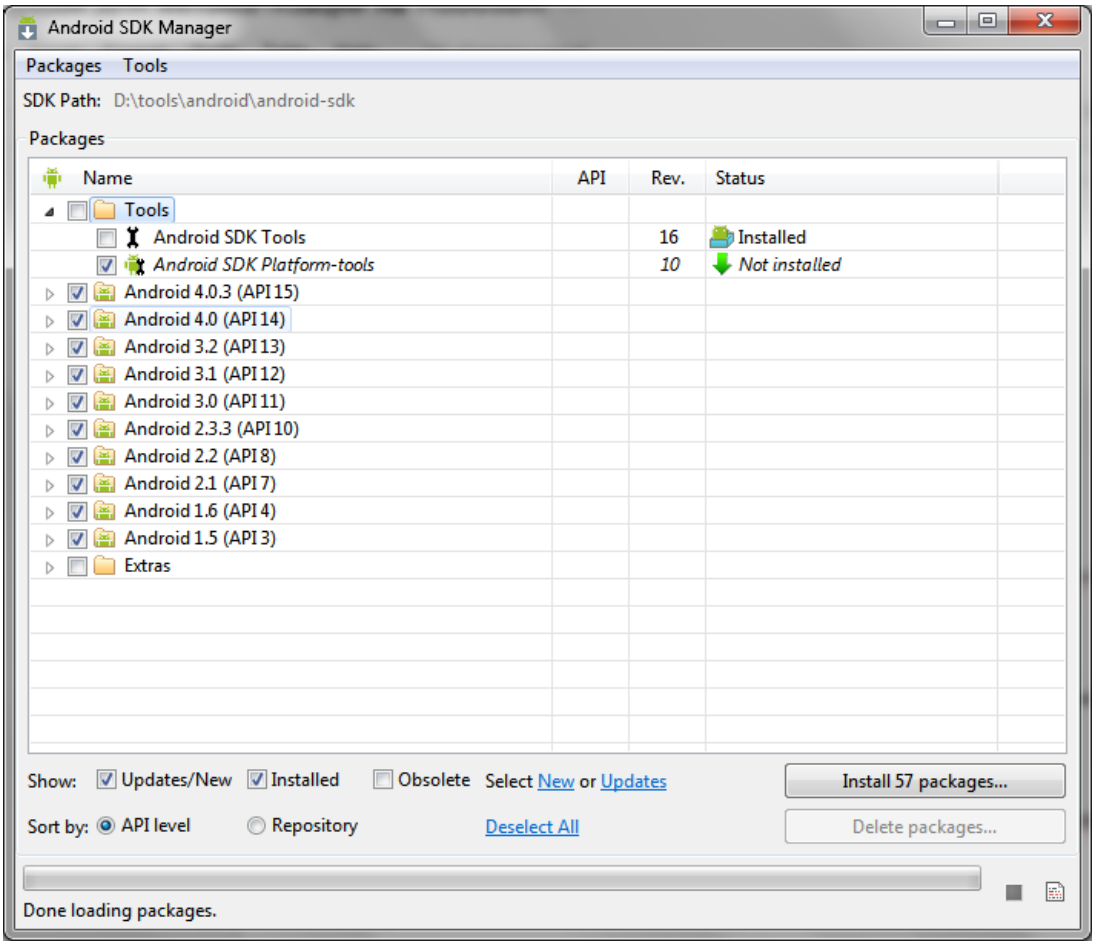
Eclipse on populaarne, avatud lähtekoodiga IDE Java keeles arendamiseks. See on saadaval allalaadimiseks enamike operatsioonisüsteemide jaoks Eclipse kodulehelt:
http://www.eclipse.org/downloads/.

4.2.3 Android SDK paigaldus

Android SDK on täiesti avatud. API laadimine on tasuta ja Google ei võta tasu ega nõua rakenduse kontrollimist enne levitamist GooglePlay's. Viimase SDK versiooni saab alla laadida *Android development* kodulehelt:
http://developer.android.com/sdk/index.html.

4.2.4 Õige platvormi valimine

Kui Android SDK on allalaetud, tuleb avada SDK Manager ja installida vajalikud platvormid (Joonis 1).



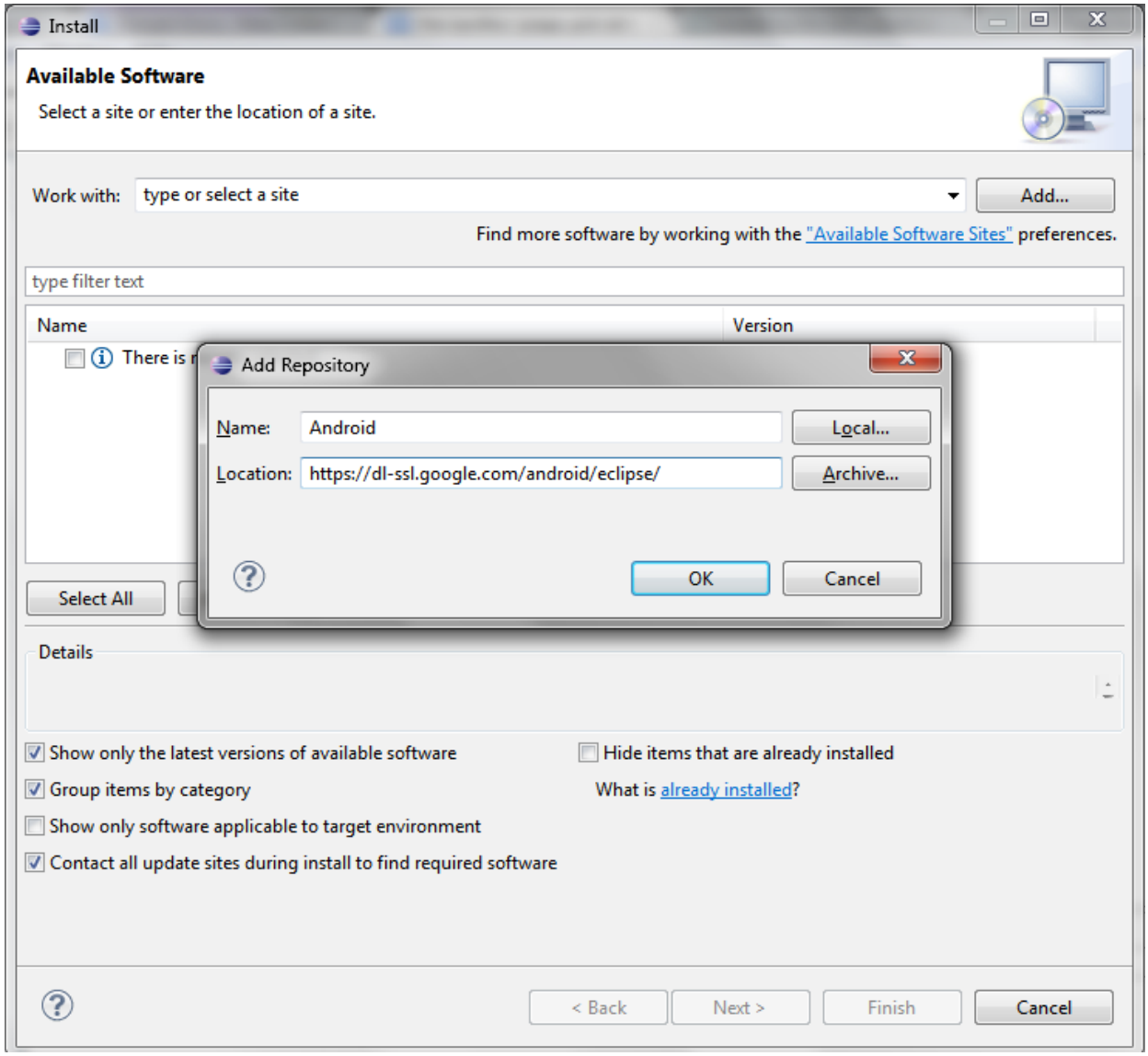
Joonis 1. Android SDK Manager.

4.2.5 ADT paigaldus

Eclipse ADT lisa lihtsustab Android'i rakenduste arendust, integreerides arendaja töövahendid otse IDE'sse, mis sisaldavad emulaatorit ja *.clas-to-.dex* konverterit. See võimaldab rakenduse arendamist, testimist ja silumist kiiremaks ja lihtsamaks teha. Eclipse'i peamenüüs tuleb valida:

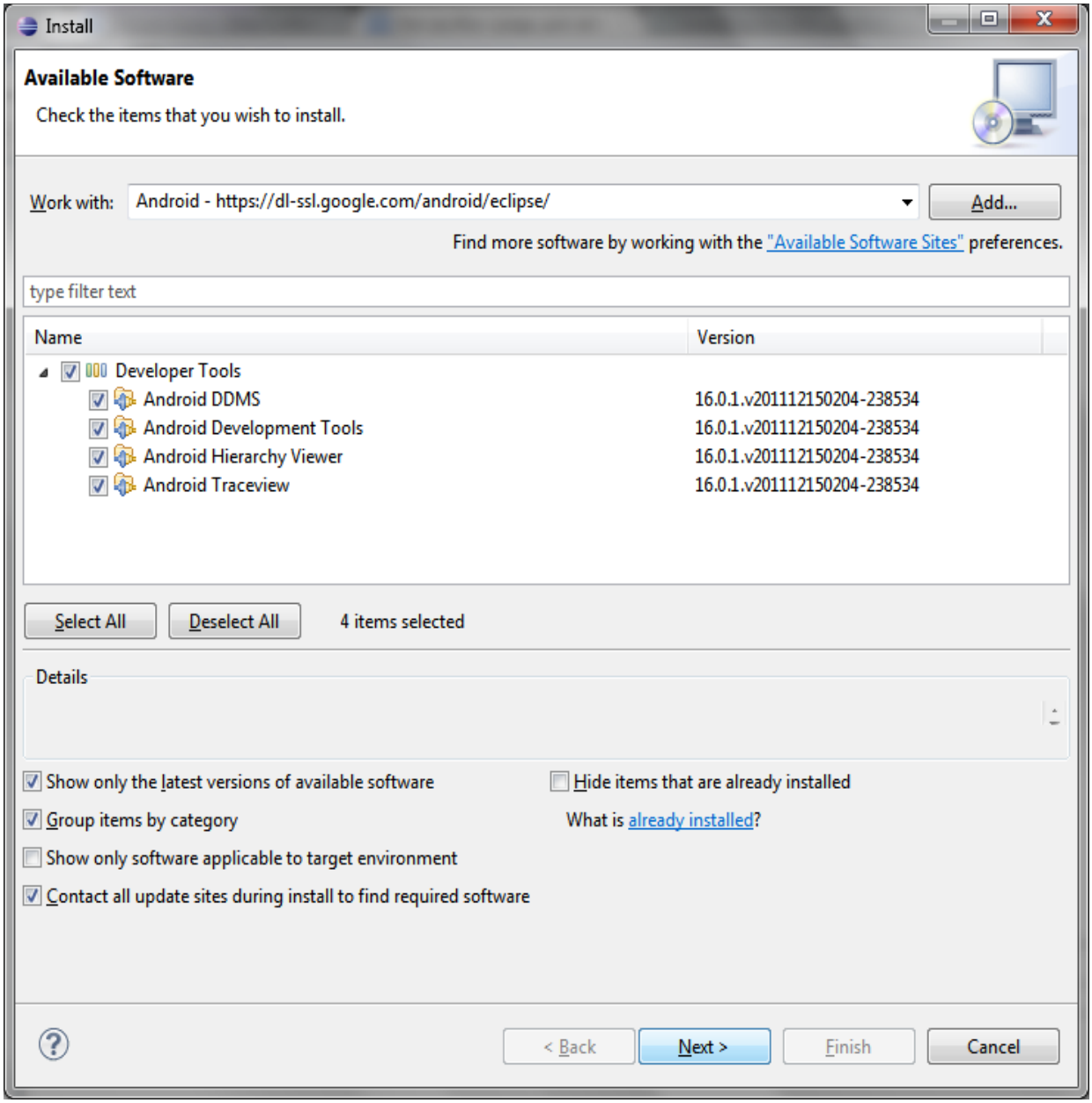
Help -> Install new software -> Add Location-> Add...

Repository asukoht on https://dl-ssl.google.com/android/eclipse/ (Joonis 2).



Joonis 2. Developer Tools installeerimine.

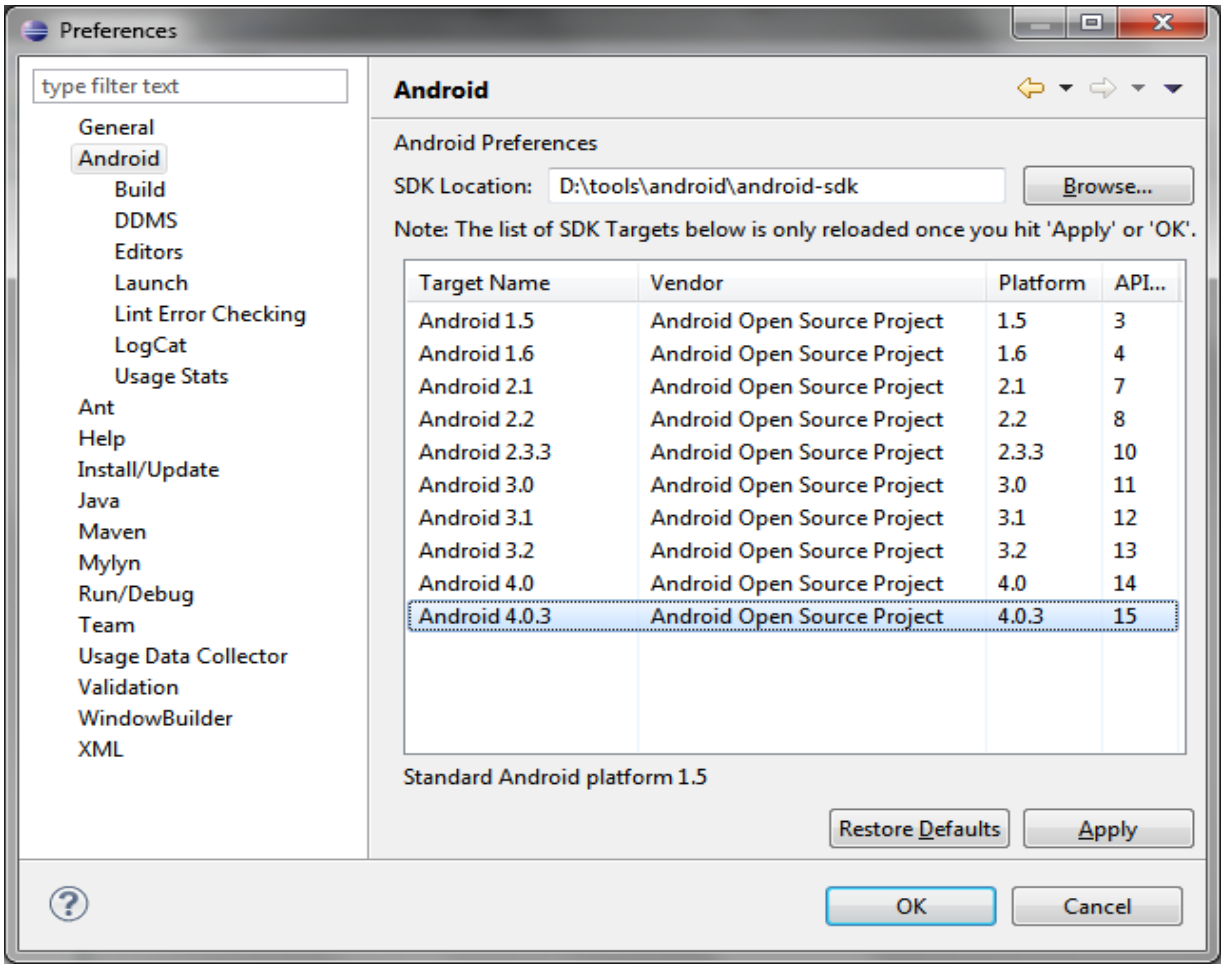
Edasi tuleb installida *Developer tools* (Joonis 3).



Joonis 3. Developer Tools installeerimine.

Pärast installeerimist tuleb Eclipse restartida.

Android SDK asukoha täpsustamine: *Windows -> Preferences ->Android* (Joonis 4).



Joonis 4. Android SDK asukoha valik.

5 ROBOTIUM

Robotium on testimise raamistik, mis on loodud võimsate ja jõuliste kasutajaliideste automaattestide kirjutamiseks Android'i rakenduste jaoks. Kasutades Robotium teste, saavad arendajad kirjutada funktsionaalseid, acceptance ja system teststsenaariume, mis hõlmavad mitut Androidi Activity't [9].
Tihtipeale on Android'i rakendusi vaja testida mitmete seadmete pea või käivitada kiiresti regressioonitestimist. Muidugi on võimalik kasutada inimressursse, kuid need ülesanded on korduvad, ajakulukad ning võivad tekitada hooletusvigu. Sel juhul oleks parem kasutada automatiseerimist, mille juures saab olla suureks abiks Robotium.

Robotium'i peamised eelised on:

- Vajatakse minimaalseid teadmisi rakendusest, mis testitakse.
- Lihtne ja kiire testide valmistamise protseduur.
- Testjuhtude kerge loetavus.
- Automaatne ajamõõtmine ja viivitused.
- Ekraanipilt katsetamise käigus, et näha, kuidas rakendus reaalses seadmes välja näeb ning käitub.
- *Screenshot*'ide võimalus kasutajaliidese testimise jooksul.

Lisaks on Robotium'i suur pluss , et sellega saab kasutada kahte tüüpi testimist. Esimesetüübi puhul on lähtekood saadaval, teine puhul on ainult *.apk* fail saadaval. Testimise põhiklass on *Solo*, mis käivitub koos *instrumentation*'ga testimise alustamisel. See klassi kasutatakse, et testide arendust lihtsamaks muuta. *Solo* toetab teste, mis kestavad mitme *activity* vahel. Testide kirtjutamisel ei ole vaja planeerida või oodata uusi *activity* teste. Kõik testid käideldatakse automaatselt, kasutades Robotium *Solo't*. Lisaks sisaldab *Solo* klass palju valmismeetodeid, mis on orienteerunud Android'i rakenduste automaattestide kirjutamiseks nii, et arendajal on võimalus kiiresti koodi kirjutada. Ja kui vajaliku meetodit ei leidu, on võimalus see kiiresti lisada. Robotium *Solo*'t tuleb kasutada koos *ActivityInstrumentationTestCase2*'ga.

ActivityInstrumentationTestCase2 testklass on mõeldud ühe rakenduse *activity* funktsionaalse testimise jaoks, mis kasutab süsteemi tavapärast infrastruktuuri. Testitav *activity* luuakse kasutades süsteemi

infrastruktuuri (kutsudes välja *InstrumentationTestCase.launchActivity()*). Pärast seda on võimalik välja kutsutud *activity*'ga otseselt manipuleerida - sellega tegeleb Robotium.

Nnende kahe klassi kasutamine võimaldab kiiresti ja paindlikult automattesteteste kirjutada ilma, et peaks muretsema testimise rakenduse *activity*'de või vajalikke meetodite puudumise üle. Üldiselt võib antud ajahetkel pidada Robotium'it üheks parimaks test raamistikuks.

6 TESTMISRAKENDUSE ÜLESSEADMINE

Käesoleva lõputöö raames luuakse üks Robotium'i testprojekt, mille lähtekood on toodud lisas Lisa 1. Antud testimisprojektis kasutatakse ainult *.apk* faili. Enne paigaldamist on mõned olulised asjad, mida tuleb kontrollida. *.apk* fail tuleb signeerida silumisrežiimis (*debug*) ning fail peab kasutama sama võtit, mida kasutab testprojekt. See võti identifitseerib rakenduse autori. Tööriistasid nagu Keytool ja Jarsigner kasutatakse kasutatavate võtmete signeerimiseks [3]. Siin on mõned sammud, mida tuleb järgida:

- Kui sertifikaadi võti on teada,siis tuleb kasutada sama võtit testprojektis.
- Kui sertifikaadi võtit pole teada, siis tuleb kustuta võtit ja kasutada sama silumise võtme test projekti ja rakenduse signeerimiseks.
- Kui rakendus on allkirjastamata, siis tuleb kasutada Androidi silumise võtit rakenduse signeerimiseks.

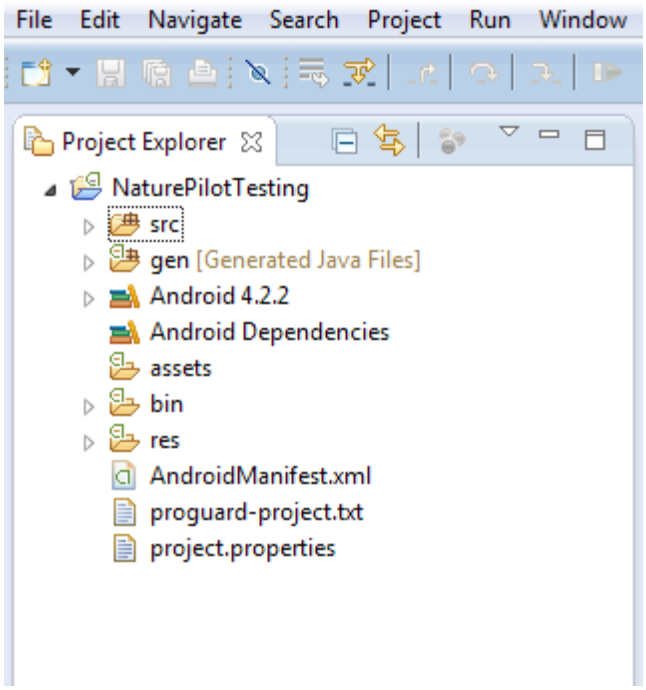
Kui rakendus on juba signeeritud, võib seda mitte-signeerituks muuta kasutades Java programmi: http://www.troido.de/re-sign.jar, käivitades programmi ja lisades sinna *.apk* faili. Kui rakendus on signeeritud silumisrežiimis, saab selle kohe paigaldada .

Paigaldamiseks kasutatakse ADB käsurea käsku:
adb install ApplicationToTest.apk

Et luua Eclipse' uus testprojekt on vaja valida *File ->New ->Project*. Uues aknas tuleb valida *Android ->Android Test Project* ja vajutada *Next*. Täidetud peavad olema ka järgmised väljad:

- *Project name: NaturePilotTesting*
- *Test target: This project*
- *Build target*: tuleb valida viimane SDK versioon.

Peale eelnevate punktide läbimist on Android'i testprojekt *NaturePilotTesting* loodud (Joonis 5).



Joonis 5. Loodud testprojekt.

AndroidManifest.xml fail on kõige olulisem fail igas Androidi projektis. Androidi süsteem kasutab seda faili mitmeteks eesmärkideks:

- Rakenduse pakettide paigaldamine ja uuendamine ning informatsiooni edastamine.
- Rakenduse *Activity'*te käivitamine.
- Rakenduste õiguste haldamine.

Peale testprojekti loomist vajab *AndroidManifest.xml* fail muutmist. Selleks on vaja teada testitava rakenduse *targetPackage* ID'd. *TargetPackage* väärtusena tuleb lisada testitava rakenduse paketi nimi:

<instrumentation
android:targetPackage="com.nature.pilot.testing"
järgmiseks:
<instrumentation
android:targetPackage="testitava rakenduse paketi nimi"

Nüüd suudab Robotium leida rakenduse, mida on vaja testida (Joonis 6).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.nature.pilot.testing"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="4" />

    <instrumentation
        android:name="android.test.InstrumentationTestRunner"
        android:targetPackage="com.nature.pilot.testing" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <uses-library android:name="android.test.runner" />
    </application>

</manifest>
```

Joonis 6. Muudetud AndroidManifest.xml sisu.

Samuti on vaja teada *launcher Activity* nime, kuid kui seda pole teada või puudub *targetPackage* ID tuleb teha järgmist:

- Käivitada käsureal *adb -d logcat -v time | findstr ActivityManager* käsk (Windows).
- Käivitada käsureal *adb -d logcat -v time | grep ActivityManager* käsk (Linux).
- Käivitada testimisrakendust.

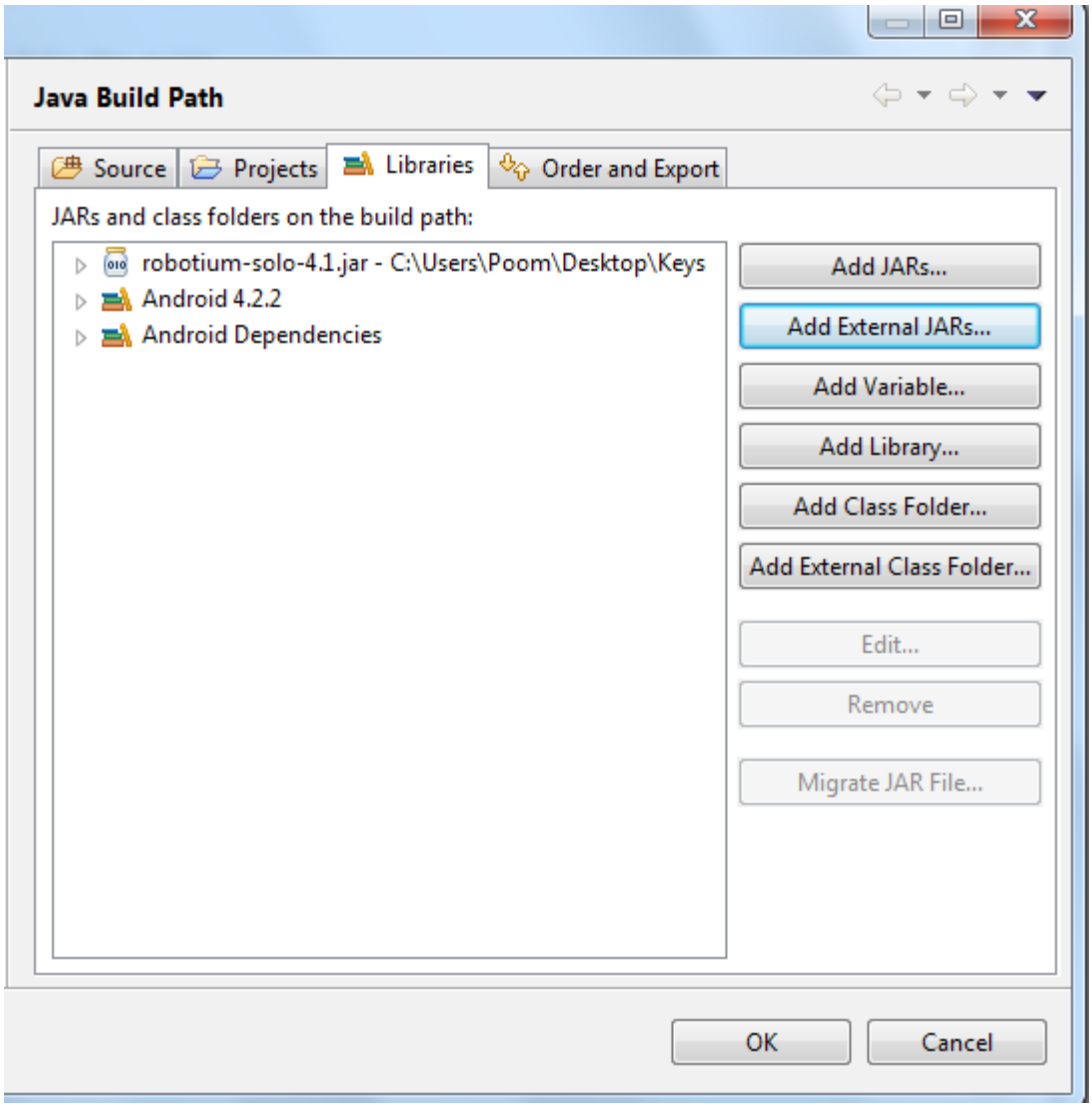
Logidest on näha:
- *org.naturepilot.on* ehk siis *targetPackage* ID
- *org.naturepilot.on.activity.MainKeyListView* ehk siis *launcher activity* nimi.

Alljärgnevalt on toodud logist kopeeritud näidis:
I/ActivityManager(519):STARTu0{act=android.intent.action.MAINcat=[android.intent.category.LAUNCHER] flg=0x10200000 cmp=org.naturepilot.on/.activity.MainKeyListView} from pid 826

Viimaseks tuleks lisada testprojekti Robotium'i viimase versiooni *.jar* fail. Viimase versiooni saab alla laadida Robotium kodulehelt: https://code.google.com/p/robotium/downloads/list

Lisamiseks projektil: *->Build path-> Configure build path.* Üleval menüüs tuleb valida *Libraries-*

>*Add External Jars* ja lisada allalaaditud Robotium jar fail projekti (Joonis 7). Peale seda on arenduskeskkond automaattestide kirjutamiseks valmis.



Joonis 7. Robotium'i .jar'i projekti lisamine.

6.1 Testide kirjutamine

Tavaliselt kasutatatakse Robotium'i testide kirjutamiseks *ActivitityInstrumentationTestCase2* testklassi. Kuid Robotium sobib kõikidele Android'i testklassidele. Lisaks kasutatakse *Solo* klassi, mis käivitub koos testi *instrumentation*'iga ja esimese tegevusega.

```
abstract public class BaseTestCase extends
            ActivityInstrumentationTestCase2<Activity> {

      private static final String TESTDATA_FILE_NAME = "testdata.properties";
      protected Properties testDataProperties;
      protected String tag = this.getClass().getSimpleName();
      protected Solo solo;

      protected static final String TARGET_PACKAGE_ID = "org.naturepilot.on";
      protected static final String LAUNCHER_ACTIVITY_FULL_CLASSNAME =
"org.naturepilot.on.activity.MainKeyListView";
      protected static Class<?> launcherActivityClass;

      // For debugging and timeouts
      protected static final int ONE_SECOND = 1000;
      protected static final int FIVE_SECONDS = 5 * ONE_SECOND;
      protected static final int ONE_MINUTE = 60 * ONE_SECOND;
      protected static final int TWO_MINUTE = 120 * ONE_SECOND;
      protected static final int FIVE_MINUTE = 300 * ONE_SECOND;

      static {
            try {
                  launcherActivityClass = Class
                              .forName(LAUNCHER_ACTIVITY_FULL_CLASSNAME);
            } catch (ClassNotFoundException e) {
                  throw new RuntimeException(e);
            }
      }

      @SuppressWarnings("deprecation")
      public BaseTestCase(String pkg, Class<Activity> activityClass) {
            super(pkg, activityClass);
      }

      @Override
      public void setUp() throws Exception {
            debug("setUp()");

            Instrumentation instrumentation = getInstrumentation();
            debug("Instrumentation loaded: " + instrumentation.getComponentName());
```

```
            solo = new Solo(getInstrumentation(), getActivity());
            debug("Solo iniated");

            InputStream inputStream = getInstrumentation().getContext()
                        .getResources().getAssets().open("conf/" +
TESTDATA_FILE_NAME);
            testDataProperties = new Properties();
            testDataProperties.load(inputStream);
            debug("Testdata loaded");

      }
```

Koodis on vaja määrata:

- *TARGET_PACKAGE_ID*
- *LAUNCHER_ACTIVITY_FULL_CLASSNAME*

Testide kävitamisel kasutab *Solo* oma meetodeid, mis *click*'ivad, *scroll*'ivad, sisestavad tekste, jne. Meetodite loetelu on üsna suur ja lisaks on võimalus kirjutada ja lisada oma meetodeid [9].

Alljärgnevalt on toodud automaattestide näidised:

```
      public void testDownloadFullGuide() {
            debug("testDownloadFullGuide()");

            // Get guide name from testdata
            String guideFull = getVariable("guideNameFull");

            // Check if there are already downloaded guide
            checkIfGuideDownloaded(guideFull);

            // Go to load guides page
            solo.clickOnText("Load guides");
            assertTrue("Load guide page not opened",
      solo.waitForActivity("MainNewKeysDownList"));

            // Click on button Full guide for downloading
            solo.clickOnText(guideFull);

            // Download guide
            solo.clickOnButton("Full guide");
```

```
        solo.waitForText("Download completed");
        solo.clickOnButton("OK");

        // Check that guides appears on your main page
        solo.goBackToActivity("MainKeyListView");
        assertTrue("Full Guide not appeared on main page",
solo.waitForText(getVariable("guideNameFull")));

}

public void testDeleteGuideFromMainPage() {
        debug("testDeleteGuideFromMainPage()");

        // Get guide name from testdata
        String guideName = getVariable("guideName");

        // If guide not downloaded then download
        addGuide(guideName);

        // If guide already downloaded then delete
        solo.clickLongOnText(guideName);
        solo.clickOnText("Delete");
        solo.clickOnButton("OK");
        solo.sleep(FIVE_SECONDS);
        assertFalse("Guide not deleted from Main page",
solo.waitForText(guideName));

}

public void testCloseDialogWindow() {
        debug("testCloseDialogWindow()");

        // Get guide name from testdata
        String guideName = getVariable("guideName2");

        // Find application textView "Load guides" by ID
        TextView loadGuide = (TextView) findTextViewById("title");

        // Go to Load guide page
        solo.clickOnText("Load guides");
        assertTrue("Load guide page not opened",
solo.waitForActivity("MainNewKeysDownList"));

        // Select guide
        solo.scrollToTop();
        solo.clickOnText(guideName);

        // Check that dialog window appears
        assertTrue("Dialog window not opened",
solo.waitForDialogToOpen(ONE_SECOND));
```

```
        // Find close button by ID
        Button close = (Button) findButtonById("close_but");

        // Close dialog windows
        solo.clickOnView(close);

        // Check that dialog window is closed
        assertTrue("Dialog windows not closed",
    solo.waitForDialogToClose(ONE_SECOND));

    }
```

Nagu eelnevast koodist näha, on Robotium'il palju meetodeid, mis kasutada.

Meetodid kasutajaliidese komponentidega manipuleerimiseks:

- *clickOnScreen*
- *clickOnMenuItem*
- *clickOnImageButton*
- *clickOnRadioButton*
- *clickOnText*
- *clickOnToggleButton*
- *clickOnView*
- *clickOnImage*
- *clickOnCheckBox*
- *clickOnButton*
- *clickOnEditText*
- jt

***Scroll'*imine ja lohistamine:**

- *scrollDown*
- *scrollDownList*
- *scrollToSide*

- *scrollUp*
- *scrollUpList*
- *drag*
- jt

Ootamine või lihtsalt paus mõneks ajaks:

- *waitForDialogToClose*
- *waitForText*
- *waitForView*
- *sleep*
- jt

Robotium leiab elemente ID järgi. Kindlasti tuleb pöörata tähelepanu klassile *R.java.* See sisaldab nimeruumi nimega *id.R.java*, mis sisaldab rakenduse graafilise kasutajaliidese kõiki ID'sid. Kui rakenduse lähtekood on saadaval (w*hitebox* testimine), võib arendaja viidata: *solo.getView (R.id.TextView01)*. Lisaks on võimalus otsida objekte, kasutades musta kasti testimist, näiteks kui lähtekood ei ole kättesaadav. *Solo* määratleb kasutajaliidese komponente nagu tavalisi numbreid, alates 0. See tähendab, et kui arendajal ekraanil on kaks nuppu, näeb nuppude vajutamine välja nii:

```
// Vajuta ülevalt esimine nupp
solo.clickOnButton(0);

// Vajuta ülevalt teine nupp
solo.clickOnButton(1);
```

Iga testi lõpetamiseks kutsub Robotium *tearDown()* meetodit, kus *Solo* lõpetab kõik avatud *activity*'d.

```
@Override
public void tearDown() throws Exception {
	debug("tearDown()");
	solo.finishOpenedActivities();
	try {
```

```
            solo.finalize();
        } catch (Throwable e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
}
```

6.2 Testimisprogrammi kirjeldus

Töö käigus oli kirjutatud testprojekt (Lisa 1), mis kajastab enamikku rakenduse funktsionaalsust. Testid on jagatud eraldi pakettidena. Projekti nimi on *com.nature.pilot.testing,* seal asub ainult üks klass – *BaseTestcase*, milles sisalduvad:

- korduvad meetodid, mis kasutatakse iga testide kävitamisel (*setUp, tearDown*);
- korduvad meetodid, mis kasutatakse mõnede testide kävitamisel puhul (*checkIfguideDownloaded, addGuide*);
- abi meetodid, mis võimaldavad seada vajaliku teadet konsooli testide käivitamisel (*void debug, void error, void warning*);
- meetodid, mis aitavad leida testelemente ID järgi (f*indTextViewById, findButtonById*);
- meetodid, mis aitavad kasutada testandmeid ja asuvad eraldi failis (*String getVariable*).

6.2.1 Esimene pakett

Esimine pakett on *com.nature.pilot.testing.add_guides,* kus asuvad kõik testid, mis testivad giidi lisamise funktsionaalsust. Siin kontrollitakse, kas rakenduse toimingud toimivad ootuspäraselt ehk need on funktsionaalse testimise näited. Lisaks tuleb neid kasutada ka regressioonitestimise puhul ning neid soovitav käivitada pärast iga rakenduse funktsionaalsuse muutmist. Tavaliselt vajavad testid enne regressioonitestimist uuendust. Sõltuvalt rakenduse funktsionaalsuse muutmisest, on need muudatused suured või väiksed, kuid kui automaatteste regulaarselt hooldada, on need muudatused väiksed. Kõik testid on kirjeldatud ühes Java klassis – *AddGuidesTestcase.*

Testid on järgmised:

Esimene test on ***testDownloadFullGuide,*** mille jooksul laaditakse alla giidi täisversioon ja pärast

kontrollitakse, et giid on allalaetud, kohalik nimikiri on uuenenud ja giid on ilmunud. Juhul, kui giidi lisamine ei õnnestu, trükitakse logides selle kohta veateade. Järgnevalt on toodud antud testi sammud:

1. Võtab giidi nime failist *testdata.properties.*

Tegevus: Võetakse giidi nimi failist *testdata.properties*.
Eeldatav tulemus: Esineb giidinimi, mis on rakenduses olemas või allalaetav.

2. Kontrollitakse, kas ettemääratud giid on laetud, kui jah siis kustutakse ettemääratud giidi.

Tegevus: Kui giid on allalaetud siis kustutab.
Eeldatav tulemus: Ettemääratud giid on kustutatud.

3. Läheb *Load guides* lehele.

Tegevus: Läheb *Load guide* lehele.
Eeldatav tulemus: *Load guide* leht on avanud.

4. Valib ettemääratud giidi ja laeb olla giidi täisversiooni.

Tegevus: Allalaetakse ettemääratud giidi täisversiooni.
Eeldatav tulemus: Giidi täisversiooni on allalaetud.

5. Ootab kuni ilmub teade, et giid on alla laetud ja vajutab *OK* nuppu.

Tegevus: Oodatakse kuni ilmub teade ja vajutab OK nuppu.
Eeldatav tulemus: Ilmub teade giidi laadimise kohta.

6. Läheb tagasi pealehele ja kontrollib, et giid on ilmunud.

Tegevus: Minnakse tagasi pealehele.
Eeldatav tulemus: Giid on allalaaditud.

Teine test on ***testDownloadTextGuide,*** mille jooksul laaditakse giidi õhem versiooni ja pärast kontrollitakse, et giid on allalaetud, kohalik nimikiri on uuenenud ja giid on ilmunud. Juhul, kui

giidi lisamine ei õnnestu, trükitakse logides selle kohta veateade.

Järgnevalt on toodud antud testi sammud:

1. Võtab giidi nime failist *testdata.properties.*

Tegevus: Võetakse giidi nimi failist *testdata.properties*.
Eeldatav tulemus: Esineb giidinimi, mis on rakenduses olemas või allalaetav.

2. Kontrollitakse, kas ettemääratud giid on laetud, kui jah, siis kustutakse ettemääratud giidi.

Tegevus: Kui giid on allalaetud siis kustutab.
Eeldatav tulemus: Ettemääratud giid on kustutatud.

3. Liigub *Load guides* lehele.

Tegevus: Liigub *Load guides* lehele.
Eeldatav tulemus: *Load guides* leht on avanud.

4. Valib ettemääratud giidi ja laeb olla giidi minimaalse versiooni.

Tegevus: Valitakse ettemääratud giidi ja allalaetakse giidi minimaalse versiooni.
Eeldatav tulemus: Giidi minimaalne versioon on allalaetud.

5. Ootab kuni ilmub teade, et giid on alla laetud ja vajutab *OK* nuppu.

Tegevus: Oodatakse kuni ilmub teade ja vajutab *OK* nuppu.
Eeldatav tulemus: Ilmub teade giidi laadimise kohta.

6. Läheb tagasi pealehele ja kontrollib, et giid on ilmunud.

Tegevus: Minnakse tagasi pea lehele
Eeldatav tulemus: Giid on allalaaditud.

Kolmas test on ***testDownloadGuideBigSize****, mille* jooksul laaditakse täis giidi, suure mahtuvusega ja pärast kontrollitakse, et giid on allalaetud, kohalik nimikiri on uuenenud ja giid

on ilmunud. Juhul, kui giidi lisamine ei õnnestu, trükitakse logides selle kohta veateade.

Järgnevalt on toodud antud testi sammud:

1. Võtab giidi nime failist *testdata.properties.*

Tegevus: Võetakse giidi nimi failist *testdata.properties*.
Eeldatav tulemus: Esineb giidinimi, mis on rakenduses olemas või allalaetav.

2. Kontrollib, kas ettemääratud giid on laetud, kui jah, siis kustutakse ettemääratud giid.

Tegevus: Kui giid on allalaetud siis kustutab.
Eeldatav tulemus: Ettemääratud giid on kustutatud.

3. Liigub *Load guides* lehele.

Tegevus: Liigub *Load guides* lehele.
Eeldatav tulemus: *Load guides* leht on avanud.

4. Valib ettemääratud suure giidi (rohkem kui 45MB) ja laeb alla giidi täisversiooni.

Tegevus: Valitakse ettemääratud giidi ja allalaetakse giidi suure mahuga versiooni.
Eeldatav tulemus: Giidi suure mahuga versioon on allalaetud.

5. Ootab kuni ilmub teade, et giid on alla laetud ja vajutab *OK* nuppu.

Tegevus: Oodatakse kuni ilmub teade ja vajutab *OK* nuppu.
Eeldatav tulemus: Ilmub teade giidi laadimise kohta.

6. Läheb tagasi pealehele ja kontrollib, et giid on ilmunud.

Tegevus: Minnakse tagasi pea lehele
Eeldatav tulemus: Giid on allalaaditud.

6.2.2 Teine pakett

Teine pakett on ***com.nature.pilot.testing.buttons,*** kus asuvad testid, mis testivad etteantud nuppe

ja nende funktsionaalsust. Need testid on funktsionaalse ja regressiooni testimise näide. Selle test klassi käivitamisel otsitakse nupud ainult ID järgi. Tavaliselt määratleb *Solo* ise kasutajaliidese komponente nagu tavalisi numbreid, alates 0st, aga need võivad segadusse minna ekraani *scroll*'imise puhul. See on näidis, kuidas Robotium'is saab otsida ja kasutada elemente kasutades nende ID-sid. Juhul elemente ei õnnestu leida, trükitakse logides selle kohta veateade. ID väljauurimiseks oli kasutatud Android SDK poolt pakutud programm nimega Hierarchy Viewer [11]. Kõik testid on kirjeldatud ühes klassis – ***FindButtonsByIdTestCase.***

Antud testid on järgmised:

Esimine test on ***testCloseDialogWindow****, mille* jooksul otsitakse nuppe ID järgi, kasutades neid, navigeeritatakse *Load guide* lehele, avatakse *Pop up* aken ja pannakse see kinni. Juhul kui *dialog window* ei ava ega sulgu, trükitakse logides selle kohta veateade.

Järgnevalt on toodud testi sammud:

1. Võtab giidi nime failist *testdata.properties.*

Tegevus: Võetakse giidi nimi failist testdata.properties.
Eeldatav tulemus: Esineb giidinimi, mis on rakenduses olemas või allalaetav.

2. Otsib *textView* elemendi „*Load page*“ kasutades ID*'d.*

Tegevus: Otsib „Load page“ elemendi.
Eeldatav tulemus: Element on leitud.

3. Vajutab *textView* elemendui peale ja läheb Load guides lehele.

Tegevus: Vajutab elemendi peale.
Eeldatav tulemus: *Load guide* leht on avanud.

4. Valib ettemääratud giidi.

Tegevus: Klõpsab ettemääratud giide peale.
Eeldatav tulemus: *Dialog window* on avatud.

5. Otsib nuppu , kasutades ID*'d.*

Tegevus: Otsib nuppu *ID* järgi.

Eeldatav tulemus: Nupp on leitud.

6. Vajutab nuppu peale ja paneb *dialog window* kinni.

Tegevus: Vajutab nuppu.

Eeldatav tulemus: *Dialog window* on kinni pandud.

Teine test ***testBackToGuidesButton****, mille* jooksul otsitakse nuppe ID järgi, kasutades neid, navigeeritatakse *Load guide* lehele ja minnakse tagasi pealehele, vajutades „*back to my guides*„ nuppu. Järgnevalt on toodud testi sammud:

1. Otsitakse *textView* elementi „Load page“, kasutades *ID'd.*

Tegevus: Otsib „Load page“ elemendi.

Eeldatav tulemus: Element on leitud.

2. Vajutab *textView* elemendi peale ja läheb *Load guides* lehele.

Tegevus: Vajautab elemendi peale.

Eeldatav tulemus: *Load guide* leht on avanud.

3. Otsitakse nuppu „Back to my guides“ kasutades *ID'd.*.

Tegevus: Otsib „Back to my guides“ elemendi.

Eeldatav tulemus: Element on leitud.

4. Vajutakse nupu peale ja minnakse tagasi *Main page* lehele.

Tegevus: Vajutab elemendi peale.

Eeldatav tulemus: *Main page* leht on avanud.

6.2.3 Kolmas pakett

Kolmas pakett on ***com.nature.pilot.testing.delete_guide***. *S*eal asuvad kõik testid, mis testivad giidi kustutamise funktsionaalsust. Need testid on funktsionaalse ja regressioonitestimise näide. Neid tuleb kasutada regressiooni testimise puhul. Kõik testid on kirjeldatud ühes klassis – ***DeleteGuidesTestCase***.

Antud testid on järgmised:

Esimine test on ***testDeleteGuideFromMainPage***. Testi jooksul kontrollitakse giidi olemasolu - kui giid pole allalaetud, siis see esmalt lisatakse, kuid kui giid on juba olemas, siis see kustutakse pealehelt. Pärast kontrollitakse, kas giid on kustutatud, kohalik nimekiri on uuenenud ja giid on nimekirjast kadunud. Juhul, kui giidi kustutamine ei õnnestu, trükitakse logides selle kohta veateade.

Järgnevalt on toodud testi sammud:

1. Võtab giidi nime failist *testdata.properties.*

Tegevus: Võetakse giidi nimi failist *testdata.properties*.
Eeldatav tulemus: Esineb giidinimi, mis on rakenduses olemas või allalaetav.

2. Kontrollitakse, kas giid on juba laaditud, kui ei, siis laetakse giid.

Tegevus: Kui giid pole allalaetud siis lisab.
Eeldatav tulemus: Giid on lisatud.

3. Giid kustutakse *Main page* lehelt.

Tegevus: Kustutab giid.
Eeldatav tulemus: Giid on kustutatud.

Teine test on ***testDeleteGuideFromLoadPage****, mille* jooksul kontrollitakse giidi olemasolu - kui giid pole allalaetud siis see esmalt laaditakse, kuid kui giid on juba allalaetud, siis see kustutakse pealehelt. Pärast kontrollitakse, kas giid on kustutatud, kohalik nimekiri on uuenenud ja giid on nimekirjast kadunud. Juhul, kui giidi kustutamine ei õnnestu, trükitakse logides selle kohta veateade.

Järgnevalt on toodud testi sammud:

1. Võtab giidi nime failist *testdata.properties.*

Tegevus: Võetakse giidi nimi failist *testdata.properties*.
Eeldatav tulemus: Esineb giidinimi, mis on rakenduses olemas või allalaetav.

2. Kontrollitakse, kas giid on juba laaditud, kui ei, siis laetakse giid alla.

Tegevus: Kui giid pole allalaetud siis lisab.
Eeldatav tulemus: Giid on lisatud.

3. Läheb *Load guides* lehele.

Tegevus: Läheb *Load* guides lehele.
Eeldatav tulemus: *Load guides* leht on avanud.

4. Giid kustutakse *Load page* lehelt.

Tegevus: Kustutab giid.
Eeldatav tulemus: Giid on kustutatud.

5. Läheb tagasi pealehele.

Tegevus: Laheb *Main page* lehele.
Eeldatav tulemus: Giid on kustutatud.

6.2.4 Neljas pakett

Neljas pakett on **c*om.nature.pilot.testing.reject_options***. *S*eal asuvad kõik testid, mis testivad

rakenduse keeldumise võimalusi. On tähtis, et läbi vaadatakse mitte ainult positiivsed stsenaariumeid, kus toimingud viiakse lõpuni, vaid ka kontrollida keeldumise võimalusi. Keeldumise puhul peab rakendus peatama toiminguid ilma , et see piiraks rakenduse tööd. Need testid on funktsionaalse ja regressiooni testimise näide. Kõik testid on kirjeldatud ühes klassis ***RejectActionsTestCase***.

Antud testid on järgmised:

Esimine test on ***testRejectDeletingFromMainPage****, mille* jooksul kontrollitakse giidi olemasolu - kui giid pole allalaetud, siis see laaditakse, kuid kui giid on juba allalaetud, siis üritakse seda pealehelt kustutada. Pärast kontrollitakse, et giid poleks kustutatud ja ei oleks nimekirjast kadunud. Juhul, kui giid kaob nimekirjast, trükitakse logides selle kohta veateade.

Järgnevalt on toodud testi sammud:

1. Võtab giidi nime failist *testdata.properties.*

Tegevus: Võetakse giidi nimi failist *testdata.properties*.
Eeldatav tulemus: Esineb giidinimi, mis on rakenduses olemas või allalaetav.

2. Kontrollitakse, kas giid on juba laaditud, kui ei, siis laetakse giid.

Tegevus: Kui giid pole allalaetud siis lisab.
Eeldatav tulemus: Giid on lisatud.

3. Keeldutatakse giidi kustutamiseset vajutades *Cancel* nuppu.

Tegevus: Vali kustutamist ja keeldu.
Eeldatav tulemus: Giid ei ole kustutanud.

Teine test on ***testRejectDeletingFromLoadPage****, mille* jooksul kontrollitakse giidi olemasolu - kui giid pole allalaetud, siis see laaditakse, kuid kui giid on juba allalaetud, siis keeldutatakse giidi *Load page* lehelt kustutamisest. Pärast kontrollitakse, et giid poleks kustutatud ja ei oleks nimekirjast kadunud. Juhul kui giid kaob nimekirjast, trükitakse logides selle kohta veateade.

Järgnevalt on toodud testi sammud:

1. Võtab giidi nime failist *testdata.properties.*

Tegevus: Võetakse giidi nimi failist *testdata.properties*.
Eeldatav tulemus: Esineb giidinimi, mis on rakenduses olemas või allalaetav.

2. Kontrollitakse, kas giid on juba laaditud, kui ei, siis laetakse giid.

Tegevus: Kui giid pole allalaetud siis lisab.
Eeldatav tulemus: Giid on lisatud.

3. Läheb *Load guides* lehe.

Tegevus: Läheb *Load guide* lehele.
Eeldatav tulemus: *Load guides* leht on avanud.

4. Keeldutatakse giidi kustutamisest vajutades *Cancel* nuppu.

Tegevus: Vali kustutamine ja keeldu.
Eeldatav tulemus: Giid ei ole kustutanud.

Kolmas test on ***testClosePopUpOnMainPage,*** mille jooksul kontrollitakse giidi olemasolu, kui giid pole allalaetud, siis see laaditakse, kuid kui giid on juba allalaetud, avatakse *Pop up* aken ja pannakse see kinni.Järgnevalt on toodud testi sammud:

1. Võtab giidi nime failist *testdata.properties.*

Tegevus: Võetakse giidi nimi failist testdata.properties.
Eeldatav tulemus: Esineb giidinimi, mis on rakenduses olemas või allalaetav.

2. Kontrollitakse, kas giid on juba laaditud, kui ei, siis laetakse giid.

Tegevus: Kui giid pole allalaetud siis lisab.
Eeldatav tulemus: Giid on lisatud.

3. Pikk „klõps“ allalaetud giidi peal.

Tegevus: Pikk „klõps“ allalaetud giidi peal.

Eeldatav tulemus: P*op up* on ilmunud.

4. Vajutab *Cancel* nuppu.

Tegevus: Vajutab *Cancel* nuppu.

Eeldatav tulemus: *Pop up* aken on kadunud.

Neljas test on ***testRejectAddingGuide***, mille jooksul keeldutakse giidi Load page lehelt lisamisest. Pärast kontrollitakse, et giid pole lisatud ja ei ole nimekirjast ilmunud. Juhul, kui giid ilmub nimekirjast, trükitakse logides selle kohta veateade.

Järgnevalt on toodud testi sammud:

1. Võtab giidi nime failist *testdata.properties.*

Tegevus: Võetakse giidi nimi failist *testdata.properties*.
Eeldatav tulemus: Esineb giidinimi, mis on rakenduses olemas või allalaetav.

2. Kontrollitakse, kas ettemääratud giid on laetud, kui jah, siis kui jah siis kustutakse ettemääratud giidi.

Tegevus: Kui giid on allalaetud siis kustutab.
Eeldatav tulemus: Ettemääratud giid on kustutatud.

3. Läheb *Load guides* lehe.

Tegevus: Läheb *Load guides* lehele.
Eeldatav tulemus: *Load guides* leht on avanud.

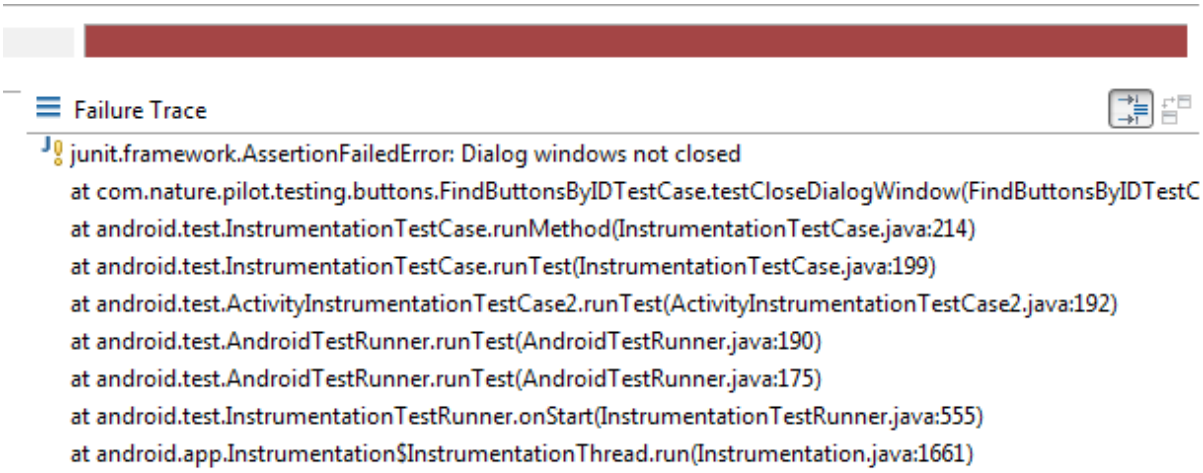
4. Keeldutatakse lisamisest vajutades *Cancel* nuppu.

Tegevus: Vali lisamist ja keeldu.
Eeldatav tulemus: Giid ei ole lisatud.

Testide kirjutamisel ja käivitamisel oli kasutatud logide vaatamiseks konsooli:

- *Windows* puhul käsk on: adb -d logcat -v time | findstr < Teie TestCase>.
- *Linux* puhul käsk on: logcat -v time | grep < Teie TestCase>.

Lisaks on võimalus jälgida logisi Android'i poolt pakutud Logcat'iga. Testide tulemused ja jooksmise ajad vaadati Eclipse's. Joonisel 8 on toodud näide, kuidas näeb välja ebaõnnestunud test logi Eclipse's.



Joonis 8. Ebaõnnestunud testi logi Eclipse's.

KOKKUVÕTE

Käesolevas töös on kirjeldatud automaattestide kirjutamist ja süsteemi üleseadmist Androidi rakendustes, kasutades Robotiumi testimise raamistikku. Mobiilseadmete turul on mitmeid operatsioonisüsteeme, kuid töös on käsitletud just Androidi platvormi selle populaarsuse ja kättesaadavuse tõttu. Nende omaduste tõttu on valitud ka Robotiumi testimise raamistik. Robotium'it on lihtne kasutada ning sobib ka keeruliste testide loomiseks.

Töös on käsitletud järgnevaid printsiipe: selgitada, kuidas Androidi arenduskeskkonda ja Robotiumi algusest peale üles seada. Seejuures on detailselt selgitatud iga protsessi etappi. Lisaks on loodi töö käigus näitena testrakendus, mille juures testiti Android'i rakendust „NaturePilot-Field guide“.

Antud töö peamine eesmärk on selgitada Androidi automaattestimist ja Robotium'i tööprintsiipe. Teema paremaks mõistmiseks on esitatud illustratsioonid ja lisana Java kood. Töö näitab, kuidas manipuleerida kasutajaliidese elemente ja koodi, mis on vajalikud, et teha lihtne Robotium'i testimisrakendus.

ABSTRACT

AUTOMATED TESTING OF ANDROID OS APPLICATIONS USING ROBOTIUM

Bachelor's thesis

Igor Poomre

Current thesis examined mobile platforms and focused on the testing automation process and more specific how it is accomplished in the Android mobile platform. Presenting information about the most popular operating systems. Android was chosen because of its popularity and the availability of the developer tools. Android is offering an open development environment built on an open-source Linux kernel. Robotium is a widely used Android testing framework. It is also open-source and has a lot of methods to manipulate an application's UI. It is very easy to use and can help to keep very high quality of software throughout the project lifecycle. The creators of Robotium have created an automated build and test environment for Android application developers who can build and test their applications on many configurations, real devices and targets that emulate real world devices. In any Continuous Integration process for Android applications the developer can automate his testing process for multiple emulators of the real android devices. In describing the Robotium framework the main goal was to understand how to set up the environment for all the tools that are required to develop and test an application. Through examples main use cases and how they work were shown.

It is a very robust and can be used in very big projects as it is very configurable and powerful. On the other hand it can become difficult to maintain unless proper care is taken. The result of the work is a project which tests an android application „NaturePilot- Field guide“.

KASUTATUD KIRJANDUS

[1] R. Meier, Professional Android 2 Application Development (Wiley Publishing, Inc., Indianapolis, 2010).

[2] J. McWherter, S. Gowell, Professional Mobile Application Development (John Wiley & Sons, Inc., Indianapolis, 2010).

[3] E. Dustin, T. Garrett, B. Gauf, Implementing Automated Software Testing. How to Save Time and Lower Costs While Raising Quality (Pearson Education, Inc., Boston, 2009).

[4] Blackberry-Company: http://ca.blackberry.com/company.html (May, 2013).

[5] iOS Technology Overview: goo.gl/hV5jP (May, 2013).

[6] Android Application Fundamentals: goo.gl/aj6Zz (May, 2013).

[7] Windows Mobile OS review: goo.gl/W47wX (May, 2013).

[8] Y Media Labs-Blog: goo.gl/yDfLq (May, 2013).

[9] Robotium Tutorials: goo.gl/ZOAq2 (May, 2013).

[10] Android Apps on Google Play: https://play.google.com (May, 2013).

[11] Hierarchy Viewer|Android Developers:goo.gl/D8Sri (May, 2013).

LISAD

Lisa 1. Robotium'i rakenduse kood CD-plaadil

LIHTLITSENTS

Mina Igor Poomre
(*autori nimi*)
(sünnikuupäev: 21.01.1983)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

Android OS rakenduste automaattestid Robotium raamistikul,
(*lõputöö pealkiri*)

mille juhendaja on Marko Peterson,
(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, 06.06.2013