Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva (FER)

Razvoj dodatka za omogućavanje statičke analize koda u programu Android Studio

Antonio Špoljar

Zagreb, lipanj 2020.

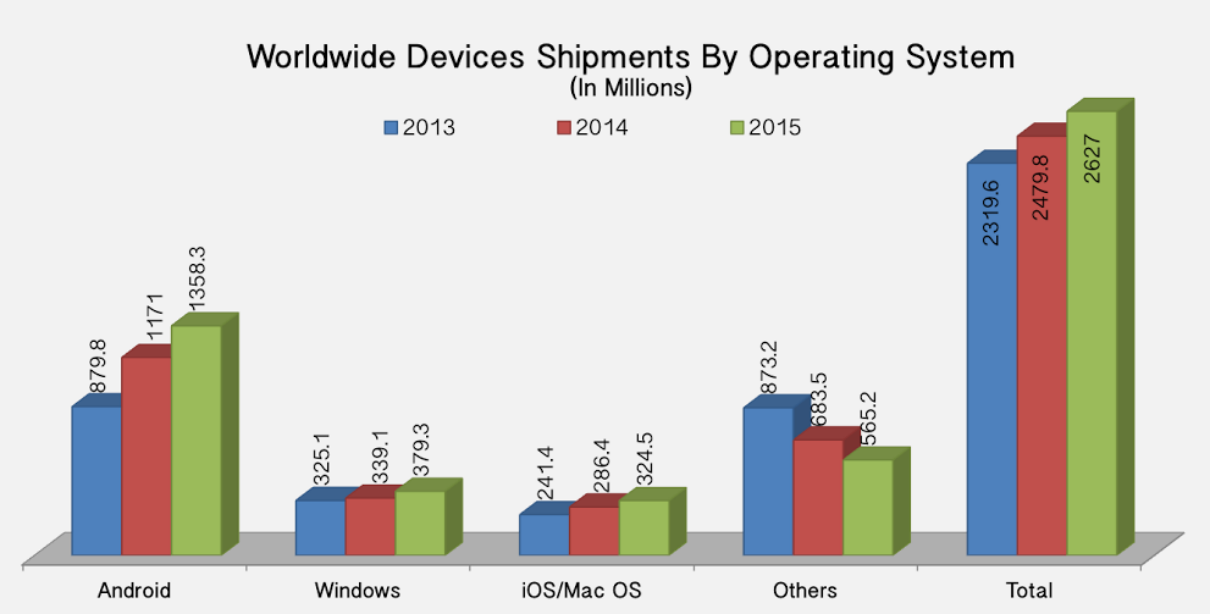
Zahvaljujem se doc. dr. sc. Grošu na ukazanom povjerenju i pomoći pri izradi rada.

Sadržaj:

1. Uvod

Mobilne aplikacije jedan su od najkorištenijih oblika softvera u svijetu, prošle godine ostvarujući rekordne brojke od 204 milijardi preuzetih primjeraka [1]. S velikim porastom korištenja pametnih telefona (engl. smartphone) i za njih predviđenih aplikacija, do velikog izražaja dolazi i sigurnost koju oni pružaju. Zlouporabom mobilnih aplikacija može nastati znatna financijska šteta, a može doći i do narušavanja privatnosti korisnika. Pri pisanju koda aplikacija, osnovna zaštita od takvih zloupotreba je uklanjanje ranjivosti koje bi napadač mogao iskoristiti u slučaju eventualnog napada. Međutim, samo pisanje koda, pa tako i njegovo provjeravanje, je dugotrajan i zahtjevan proces koji bismo htjeli što je moguće više olakšati. Osim toga, ljudi su također skloni rađenju pogrešaka pri tako opsežnim projektima. Zato se pri provjeri sigurnosti pokušava minimizirati količina grešaka korištenjem za to predviđene programske podrške. Temeljna programska podrška za tu namjenu se temelji na statičkoj analizi izvornog programskog koda. Takvi programi automatski uočavaju ranjivosti i što ranije upozoravaju programera na pogreške koje je učinio.

Android operacijski sustav je najkorišteniji operacijski sustav ne samo na mobilnim uređajima, nego i općenito [2]. Grafikon na slici 1.1 pokazuje raspodjelu korištenja operacijskih sustava na svim uređajima, u rasponu od 2013. godine do 2015. godine. Jedan od najpopularnijih programa za pisanje i uređivanje programskog koda, poznatih kao IDE (engl. Integrated Development Environment), je Android Studio. U njemu se razvija kod za Android operacijske sustave, i to pretežito u programskom jeziku Java. Alati za statičku analizu izvornog koda, te dodaci koji integriraju takve alate u Android Studio, vrlo su važna komponenta pri razvoju Android aplikacija.



Slika 1.1 Korištenje operacijskih sustava u svijetu 2014-2015

Dodaci (engl. plugins) su softverske komponente koje proširuju specifično svojstvo ili sposobnost postojećeg programa [3]. Android Studio dodatke može izraditi i objaviti bilo tko uz pomoć drugog IDE-a, IntelliJ IDEA. Naime, pisanjem Java koda i korištenjem postupaka opisanih u priručniku za izradu dodataka [4] može se napraviti dodatak za Android Studio, koji se temelji na IntelliJ IDEA-i. Takav dodatak, koji proširuje neku sposobnost IDE-a, se zatim može koristiti lokalno ili objaviti na trgovini (engl. marketplace) Android Studia. Za pisanje dodatka poželjno je poznavanje Javinog sučelja *Swing,* te osnove pisanja Gradle skripti.

Cilj ovog rada je napraviti dodatak za Android Studio koji će u njega dodati jedan prozor pomoću kojeg će se moći odabrati željene opcije i pokrenuti analiza projekta. Analizu će provoditi skripta pisana u Python, i to pomoću pokretanja unaprijed preuzetih alata za statičku analizu izvornog koda. Skripta radi na Linux uređajima, jer koristi specifičnosti operacijskog sustava. Zadatak skripte je da generira izvještaj koji ukazuje korisniku na postojeće ranjivosti u kodu, te ga obavještava o općenitom ne pridržavanju standarda kodiranja u Javi. Pri završetku analize, dodatak korisnika obavještava o uspjehu te o lokaciji generiranog izvještaja.

Rad je podijeljen na 5 glavna poglavlja. Prvo je poglavlje uvodno, nako čega se u drugom poglavlju ukratko objašnjava statička analiza izvornog koda, te su dalje opisani izrada i funkcionalnosti skripte pisane u programskom jeziku Python koja uz pomoć ulaznih parametara i nekoliko alata za analizu Java koda i Android aplikacija generira izvještaj. Treće poglavlje se koncentrira na objašnjavanje dodataka za Android Studio, te predstavlja osnovne koncepte koji se koriste u izradi istih. Opisuje se izrada jednostavnog *HelloWorld* dodatka kako bi se primjenili opisani koncepti. U četvrtom poglavlju se opisuje izrađeni dodatak za Android Studio koji integrira skriptu iz drugog poglavlja u sam Android Studio. Također se dotiče načina distribucije dodataka. Posljednje poglavlje je zaključak.

2. Statička analiza izvornog koda

U ovom poglavlju se opisuju alati koji analiziraju izvorni kod, te Python skripta koja pomoću tih alata generira izvještaj za korisnika. Objašnjeno je što je to statička analiza koda te su nabrojani alati za statičku analizu izvornog koda korišteni u radu. Opisane su mogućnosti i namjene pojedinog alata, te su prikazani rezultati pokretanja alata nad stvarnim projektima. Dalje su opisane funkcionalnosti skripte, objašnjena je njena upotreba te su nabrojane sve dodatne opcije koje je moguće uključiti. Na kraju je objašnjen najbitniji dio programskog koda skripte.

2.1 Alati za statičku analizu koda

Statička analiza izvodnog koda je metoda pronalaženja pogrešaka koja se temelji na automatskom pregledu izvornog koda [5]. Uobičajeno se kod analizira uz pomoć unaprijed pripremljenih pravila, te se prijavljuje kod koji ne slijedi dobre prakse kodiranja specificirane tim pravilima. Izrazi *statička analiza izvornog koda* (engl. static source code analysis), *statička analiza koda* (engl. static code analysis) te *analiza izvornog koda* (engl. source code analysis) se često koriste kao istovrijedni izrazi, iako postoje neke nijanse razlike među njima. Za potrebe ovog rada nije nužno ulaziti u dubinsku analizu prethodnih izraza, te će se u radu odsad sva tri izraza koristiti kao sinonimi.

Postoje mnogi alati za analizu izvornog koda, neki od njih otvorenog koda (engl. open source), a neki zatvorenog tipa. Opsežan popis se može pronaći na [6]. Od mnogih dostupnih alata s popisa, prvo su izdvojeni oni koji analiziraju Java kod i Android aplikacije. Zatim su za ovaj rad izabrani sljedeći: SpotBugs [7], Checkstyle [8], PMD i CPD [9] te Graudit [10]. Pri odabiru alata se nisu uzimali u obzir oni koji nisu bili redovito održavani, odnosno zadnja objava (engl. commit) na GitHub repozitorij im nije unutar zadnje dvije godine, iz razloga što starije verzije alata uglavnom ne mogu raditi s novim verzijama Gradlea, Jave i Android SDK-a. Također se nisu uzimali oni alati koji imaju samo grafičko sučelje, odnosno nemaju opciju pokretanja iz naredbenog retka operacijskog sustava Linux. Ukoliko alat sam ne podržava ispis u datoteku, korištenjem operatora „>“ i „>>“ u ljusci se ispis izvještaja preusmjerava u tekstualnu datoteku.

SpotBugs alat je novija, održavana verzija alata FindBugs koji se više ne koristi. Alat traži pretežito probleme vezane uz sigurnost aplikacija, te ima mnogo opcija. Vrlo je spor u ispitivanju jer mora proći kroz mnogo datoteka projekta, no pronalazi neke veće pogreške u kodu, kao što je curenje resursa (engl. resource leak). Može se koristiti samostalno, no isto tako postoje i verzije integrirane u neke druge alate kao što su Gradle, Ant i Maven. Također, postoji i verzija kompatibilna s Eclipse IDE-om. Može se koristiti i u naredbenom retku, a pri tome se na zadnjem mjestu navodi ciljani direktorij ili datoteka za analizu, a prije toga se mogu navesti dodatne opcije u obliku parametara. Preuzeto je šest verzija alata (spotbugs_3.1.0_RC7, spotbugs_3.1.12, spotbugs_4.0.0_beta1, spotbugs_4.0.0_beta2, spotbugs_4.0.0_beta3, spotbugs_4.0.0_beta4). Neki od najbitnijih parametara pri korištenju SpotBugs iz naredbenog retka su dani u tablici 2.1. Također, primjer pokretanja alata iz naredbenog retka je dan na ispisu 2.1, a nekoliko izdvojenih linija izvještaja na ispisu 2.2.

Parametar	**Opis funkcionalnosti**
textui	Parametar koji pokreće SpotBugs u tzv. command line mode opciji
onlyAnalyze	Parametar iza kojeg se navodi paket projekta iz kojeg se žele analizirati Java datoteke, što je nepotrebno za slučaj pokretanja nad jednom Java datotekom
workHard	Parametar koji osigurava da je analiza barem srednje (engl. medium) strogosti, odnosno garantira da se neće ignorirati veće pronađene greške
longBugCodes	Parametar za detaljniji ispis pronađenih pogrešaka
progress	Prikazuje napredak ispitivanja
sourcepath	Parametar iza kojeg se navodi putanja izvorišnog koda projekta ili datoteke koja se ispituje
home	Parametar iza kojeg se navodi putanja do direktorija SpotBugs alata na računalu
output	Parametar iza kojeg se navodi putanja na koju se sprema *.txt* datoteka s izvještajem alata, a u slučaju da parametar nije uključen izvještaj se ispisuje u naredbeni redak
auxclasspathFromFile	Parametar iza kojeg se navodi putanja do *.txt* datoteke koja sadrži sve dodatne lokacije koje koristi projekt. To je uglavnom lokacija korištenog Android SDK-a, gdje se nalaze *jar* i druge pomoćne datoteke i arhive, te lokacija Gradle direktorija. Primjer izgleda takve datoteke je na ispisu 2.2.

Tablica 2.1 Parametri alata SpotBugs

```
bash spotbugs -textui -onlyAnalyze jakhar.aseem.diva.* -workHard -
longBugCodes -progress -sourcepath /home/user/Documents/Projekt/diva-
android/app/src -home ../ -output
/home/user/Documents/Projekt/reports/SpotBugs-4.0.0-beta4_report -
auxclasspathFromFile
/home/user/Documents/Projekt/reports/Fajlovi/auxclasspathFromFile
/home/user/Documents/Projekt/diva-android/
```

Ispis 2.1 Primjer narede za pokretanje alata SpotBugs

```
...
M B NM_CLASS_NAMING_CONVENTION Nm: The class name
jakhar.aseem.diva.R$bool doesn't start with an upper case letter  At
R.java:[line 4093]

M X OBL_UNSATISFIED_OBLIGATION_EXCEPTION_EDGE OBL:
jakhar.aseem.diva.InsecureDataStorage3Activity.saveCredentials(View)
may fail to clean up java.io.Writer on checked exception  Obligation
to clean up resource created at
InsecureDataStorage3Activity.java:[line 64] is not discharged

H I DM_DEFAULT_ENCODING Dm: Found reliance on default encoding in
jakhar.aseem.diva.InsecureDataStorage3Activity.saveCredentials(View):
new java.io.FileWriter(File)  At
InsecureDataStorage3Activity.java:[line 64]

...
```

Ispis 2.2 Nekoliko linija SpotBugs izvještaja

```
/home/user/Android/sdk
/home/user/.gradle/
```

Ispis 2.3 Primjer izgleda datoteke koja se predaje uz parametar *-auxclasspathFromFile*

Checkstyle je jednostavan alat koji provjerava strukturu i kvalitetu koda, te prijavljuje dijelove koda koji ne odgovaraju standardima programiranja u Javi, koji su definirani u *Google Java Style Guide* [11]. Druga kolekcija pravila za Checkstyle koja se nije koristila u ovom radu je definirana u *Java Code Conventions* pravilniku [12]. Pravila se odnose na formatiranje Java koda, nazive datoteka, paketa i razreda, prakse u programiranju (engl. programming practices), Java dokumentaciju (engl. Javadoc) i slično. Jedini bitni parametar kod pokretanja Checkstyle alata iz naredbenog retka je *-c*. Iza njega se navodi putanja do *xml* datoteke koja sadrži pravila. Poslije zadnjeg parametra se navodi putanja do datoteke koja se analizira. Za razliku od ostalih alata, Checkstyle dolazi u obliku *jar* arhive, te je pokretanje alata malo drugačije. Primjer naredbe za poziv Checkstyle alata je dan na ispisu 2.4, dok je nekoliko linija iz izvještaja prikazano na ispisu 2.5.

```
java -jar checkstyle-8.27-all.jar -c
/home/user/Documents/Projekt/reports/Fajlovi/checkstyle.xml
/home/user/Documents/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/AccessControl3Activity.ja
va > /home/user/Documents/Projekt/reports/checkstyle_report
```

Ispis 2.4 Primjer naredbe za pokretanje alata Checkstyle

```
...
- android.support.design.widget.Snackbar. [UnusedImports]
[ERROR]                  /home/user/Documents/Materijali/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/MainActivity.java:46:5:
Class 'MainActivity' looks like designed for extension (can be
subclassed), but the method 'onCreate' does not have javadoc that
explains how to do that safely. If class is not designed for extension
consider making the class 'MainActivity' final or making the method
'onCreate' static/final/abstract/empty, or adding allowed annotation
for the method. [DesignForExtension]
[ERROR]                  /home/user/Documents/Materijali/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/MainActivity.java:47:29:
Parameter savedInstanceState should be final. [FinalParameters]
...
```

Ispis 2.5 Nekoliko linija Checkstyle izvještaja

Alat PMD analizira izvorni kod tražeći nekorištene varijable, prazne *catch* blokove, nepotrebno kreiranje objekata i mnoge druge slične greške. Može se koristiti za analizu izvornih kodova mnogih jezika: Java, JavaScript, XML, Scala, PLSQL i drugi. Ovaj alat također koristi *xml* datoteku kako bi specificirao pravila. Međutim, tu sam korisnik izrađuje pravila, pri čemu su osnove kreiranja takve datoteke dane na službenim stranicama [13]. CPD alat, koji je distribuiran zajedno s PMD-om, traži duplicirani kod, i to kroz jednu ili više datoteka izvornog koda. Parametri koji su se koristili uz PMD dani u tablici 2.2, dok je primjer naredbe za pokretanje alata na ispisu 2.6.

Parametar	**Opis funkcionalnosti**
d	Parametar iza kojeg se navodi datoteka ili vršni direktorij projekta kojeg se želi analizirati
f	Parametar iza kojeg se navodi koja vrstu datoteke se želi generirati kao izvještaj (npr. opcija *text* ako se želi generirati *.txt* izvještaj)
auxclasspath	Slično kao parametar *auxclasspathFromFile* kod SpotBugs alata. Iza parametra se, odvojene zarezom, pišu putanje do direktorija, arhiva i datoteka
R	Iza ovog parametra se navodi lokacija *xml* datoteke s pravilima

Tablica 2.2 Parametri alata PMD

```
bash run.sh pmd -d /home/user/Documents/Projekt/diva-android/app/src/
-f      text      -auxclasspath      /snap/android-studio/81/android-
studio/,/home/user/.gradle/                                          -R
/home/user/Documents/Projekt/reports/Fajlovi/rulesets.xml            >
/home/user/Documents/Projekt/reports/pmd_report
```

Ispis 2.6 Primjer naredbe za pokretanje alata PMD

CPD koristi dva bitna parametra, a to su *--minimum-tokens* i *--files*. Iza prvog se definira minimalni broj ponovljenih znakova u jednoj ili više datoteka koji se prijavljuje, dok se iza drugog navode datoteke i direktoriji koje se analizira. Primjer naredbe za pokretanje alata CPD je na ispisu 2.7.

```
bash      run.sh      cpd      --minimum-tokens      100      --files
/home/user/Documents/Projekt/diva-android/app/src/                   >
/home/user/Documents/Projekt/reports/cpd_report
```

Ispis 2.7 Primjer naredbe za pokretanje alata CPD

U nastavku je na ispisu 2.8 prikazano nekoliko linija izvještaja alata PMD, a zatim na ispisu 2.9 isto za alat CPD.

```
/home/user/Documents/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/APICreds2Activity.java:35
:     Avoid unused imports such as 'android.os.Environment'

/home/user/Documents/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/APICreds2Activity.java:44
:     Each class should declare at least one constructor

/home/user/Documents/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/APICreds2Activity.java:47
:     Avoid excessively long variable names like savedInstanceState

/home/user/Documents/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/APICreds2Activity.java:47
:     Parameter 'savedInstanceState' is not assigned and could be
declared final

...
```

Ispis 2.8 Nekoliko linija PMD izvještaja

```
Found a 15 line (110 tokens) duplication in the following files:
Starting at line 61 of /home/user/Documents/Projekt/diva-android/app/src/main/java/jakhar/aseem/diva/InsecureDataStorage3Activity.java
Starting at line 60 of /home/user/Documents/Projekt/diva-android/app/src/main/java/jakhar/aseem/diva/InsecureDataStorage4Activity.java

            File uinfo = File.createTempFile("uinfo", "tmp", ddir);
            uinfo.setReadable(true);
                                ...
```

Ispis 2.9 Nekoliko linija CPD izvještaja

Graudit je jednostavna skripta koja pomoću GNU naredbe *grep* traži rupe u sigurnosti u izvornom kodu. Parametri koji se koriste su dani u tablici 2.3. Alat dolazi uz mnogo baza podataka koje se mogu koristiti: Android, DotNet, iOS, Java, Javascript, Perl, PHP, Python, Ruby, SQL... Nedostatak ovog alata je što u izvještaju uz ispisani kod ne navodi grešku ili razlog izdvajanja tog koda kao potencijalno lošeg. Neki bitni parametri koji se mogu koristiti pri pokretanju Graudita su dani u tablici 2.3. Primjer naredbe za pokretanje je na ispisu 2.10, dok je dio izvještaja predočen ispisom 2.11.

Parametar	**Opis funkcionalnosti**
B	*Supress banner* parametar, odnosno sprječava ispisivanje imena alata prije samog izvještaja
z	Isključivanje korištenja boja
d	Parametar za određivanje baze podataka koja se koristi pri analizi (npr. *-d android*)

Tablica 2.3 Parametri alata Graudit

```
bash graudit -B -z -d android /home/user/Documents/Projekt/diva-android/app/src > /home/user/Documents/Projekt/reports/graudit_report
```

Ispis 2.10 Primjer naredbe za pokretanje alata Graudit

```
/home/user/Documents/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/InsecureDataStorage1Activ
ity.java-33-
/home/user/Documents/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/InsecureDataStorage1Activ
ity.java:34:import android.content.SharedPreferences;

/home/user/Documents/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/InsecureDataStorage1Activ
ity.java-35-import android.preference.PreferenceManager;

################################################

/home/user/Documents/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/InsecureDataStorage1Activ
ity.java-44-    @Override

/home/user/Documents/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/InsecureDataStorage1Activ
ity.java:45:    protected void onCreate(Bundle savedInstanceState) {

/home/user/Documents/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/InsecureDataStorage1Activ
ity.java:46:        super.onCreate(savedInstanceState);

/home/user/Documents/Projekt/diva-
android/app/src/main/java/jakhar/aseem/diva/InsecureDataStorage1Activ
ity.java-47-
setContentView(R.layout.activity_insecure_data_storage1);

################################################

                                ...
```

Ispis 2.11 Nekoliko linija Graudit izvještaja

2.2 Skripta za statičku analizu koda

Nakon što su pojedini alati konfigurirani, pokrenuti i isprobani, sljedeći korak je izrada skripte. Za izradu je odabran programski jezik Python, specifično u projektu je korišten Python3 interpreter verzije 3.6. To znači da korisnik mora na računalu imati instaliran Python3, verzije veće ili jednake 3.6. Zadaća skripte je funkcionirati slično kao i alati koji su se proučavali u prethodnom poglavlju, odnosno ponuditi opcije koje se uključuju korištenjem određenih parametara pri pozivu iz naredbenog retka. Grafičko sučelje nije razmatrano. Skripta je dostupna za korištenje i može se naći na GitHubu [14].

2.2.1 Funkcionalnosti skripte

Skripta nudi nekoliko opcija pokretanja. *README.md* datoteka se može ispisati pozivanjem skripte s parametrom *-h*. Drugi bitni podatak, lista svih dostupnih alata, dobiva se pokretanjem uz parametar *-l*. Osim kod prethodno navedena dva specijalna načina pokretanja, oblik naredbe za pokretanje skripte mora biti: *python3 script.py [-t] [options]*. Tu se pretpostavlja da smo pozicionirani u direktoriju *Script* koji sadrži Python datoteku *script.py*, koja je ulazna točka programa. Nakon parametra *-t* se, odvojeni zarezom i bez razmaka, navode alati koje se želi pokrenuti. Dozvoljeni su svi alati prikazani u poglavlju 2.1, a uz to je dozvoljeno pisati ključnu riječ *spotbugs*, koja pokreće najnoviju verziju SpotBugs alata dostupnu (to je trenutno spotbugs_4.0.0_beta4). Postoje i dvije dodatne opcije. Prva se uključuje parametrom *-p* iza kojeg se navodi ime paketa kojeg se želi analizirati. Druga opcija se piše iza parametra *-c*, koji ima sličnu funkciju kao *auxClasspath* parametri kod SpotBugs i PMD alata. Iza tog parametra se navodi putanja do datoteke, koja mora izgledati kao na ispisu 2.1. Međutim, pri pokretanju nekih alata obavezno je uključiti jednu ili obje opcije. Na ispisu 2.12 je dio *README.md* datoteke koji prikazuje sve ponuđene alate, te za svaki specificira koje dodatne opcije moraju biti uključene pomoću parametara ako se želi pokrenuti alat. Također, na ispisu 2.13 su dani primjeri naredbi.

```
Available tools (run with -t), must be comma separated without any
spaces:
        spotbugs_3.1.0_RC7
        spotbugs_3.1.12
        spotbugs_4.0.0_beta1
        spotbugs_4.0.0_beta2
        spotbugs_4.0.0_beta3
        spotbugs_4.0.0_beta4
        spotbugs (runs the newest version)
        pmd
        cpd
        graudit
        checkstyle
    Following options should be provided when running specific tools
while analysing a project:
        spotbugs (all versions):
            -c <file>
            -p <package>
        pmd:
            -c <file>
        cpd: none
        graudit: none
        checkstyle:
            -p <package>
```

Ispis 2.12 Izvadak iz datoteke *README.md*

```
python3 script.py -h

python3 script.py -l

python3 script.py -t spotbugs,pmd,cpd,checkstyle,graudit -c
/home/user/Documents/auxclasspathFromFile -p my.package.*
/home/user/Documents/project/

python3 script.py -t pmd,cpd,checkstyle -c
/home/user/Documents/auxclasspathFromFile
/home/user/Documents/project/app/src/main/java/my/package/Class
ForAnalysis.java
```

Ispis 2.13 Primjeri naredbi za pokretanje skripte iz naredbenog retka

2.2.2 Programski kod skripte

U vršnom direktoriju projekta se nalazi nekoliko bitnih poddirektorija i datoteka. Tu je *README.md* datoteka, *script.py* koji je ulazna točka programa i gdje se nalazi većina koda, te *utils.py* koji sadrži potrebne konstante. Zatim direktorij *tools* gdje su spremljeni svi alati opisani u poglavlju 2.1. Direktorij *reports* je predviđen za izvještaje pojedinih alata. Tu su još i dvije *css* datoteke za oblikovanje *html* izvještaja te *xmls* direktorij u kojem su *xml* datoteke potrebne alatima. Na kraju će se u vršnom direktoriju nalaziti i datoteka imena *report.html* koja sadržava generirani konačan izvještaj skripte.

Datoteka *script.py* ulazna je točka programa, te sadrži cijeli glavni dio programskog koda. Ideja je da se učitaju i provjere parametri i putanje koje je korisnik unio u naredbeni redak, te se pokrenu svi alati koje je korisnik naveo. Izvještaji se spremaju u *reports* direktorij. Zatim se na temelju tako dobivenih izvještaja boduju alati. Naposlijetku, izrađuje se *report.html* datoteka s izvještajem svakog pokrenutog alata, te tablicom bodova.

Za učitavanje parametara dobivenih kroz naredbeni redak se koristi metoda *getopt* modula istog imena. Najprije se pomoću *argv = sys.argv[1:]* izbaci prvi predani argument, što je uvijek ime same skripte (u ovom slučaju *script.py*). Poziva se metoda *getopt* kojoj se kao prvi parametar predaje naš modificirani *argv*, a kao drugi argument *String* vrijednost u kojoj se navode svi parametri koji mogu biti legalno predani, i to na način da se one iza kojih se očekuje dodatni argument zapiše s dvotočkom (t, c, p na ispisu 2.14). Metoda vraća dva objekta, na ispisu 2.14 spremljene u varijable *opts, args. args* sadrži zadnji predani argument, putanju do datoteke ili direktorija koji se analizira. *opts* je *tuple* objekt koji je oblika *(opt, arg),* gdje je *opt* parametar (h, l, t, c, p), a *arg* argument koji je unesen neposredno iza parametra.

```
    try:
        opts, args = getopt.getopt(argv, "hlt:c:p:")
    except getopt.GetoptError:
        error_message(getopt.GetoptError.msg + "Incorrect arguments. "
                                               "\nFor help read README file
or run the script with -h option.", 1)
```

Ispis 2.14 Učitavanje argumenata, kod razreda *script.py*

U kodu se provjerava je li korisnik unio valjane argumente uz parametre, te se pamti koji argumenti su predani uz koje parametre. Uz to se odredi je li zadnji predani argument Java datoteka ili direktorij, te se sukladno tome pokreće prikladni potprogram. Nakon toga se pokreću alati, i to na način da se obilazi lista u kojoj su zapisani svi alati koje je korisnik naveo iza *-t* parametra. Provjerava se koji od alata se trenutno obrađuje, te se ulazi u potrebni *if* blok. Konstruira se *String* koji sadrži cijelu naredbu za pozivanje alata iz naredbenog retka, te se stvara datoteka u koju će biti pohranjen izvještaj alata pomoću naredbe *open* i argumenta *w+* koji Pythonu nalaže da stvori datoteku s predanom putanjom u slučaju da ona ne postoji, a da obriše sadržaj datoteke ako ona već postoji.

Potom se zove metoda *Popen* modula *subprocess*, koja prima 4 bitna argumenta. Prvi od njih je *String* s naredbom ljuske. Drugi, treći i četvrti argument upravljaju načinom izvršavanja naredbe, odnosno treba li se ona izvršiti u ljusci, te što da koristi kao standardni izlaz, a što kao izlaz za pogreške (*stdout* i *stderr*). Zatim se obavezno čeka da proces završi naredbom *process.wait()* jer, kao što je prije spomenuto, u slučaju nekih alata analiza traje dugo. U slučaju da alat sam kao jedan od argumenata dobiva put do datoteke u koju sprema izvještaj, predaje se novokreirana (ili postojeća, ali prazna) datoteka stvorena prije opisanim postupkom pomoću naredbe *open*. Međutim ako alat nema te opcije, koristi se opcija ljuske za preusmjeravanje ispisa u datoteku, odnosno operator „>“. Na ispisu 2.15 je dio koda iz petlje koji, u slučaju da je predan odgovarajući argument, pokreće SpotBugs alat opisanim postupkom.

```
for t in tools:
        if t in (SPOTBUGS_4_0_0_BETA4, SPOTBUGS_4_0_0_BETA3,
SPOTBUGS_4_0_0_BETA2, SPOTBUGS_4_0_0_BETA1,
                 SPOTBUGS_3_1_12, SPOTBUGS_3_1_0_RC7, SPOTBUGS):
            if t == SPOTBUGS:
                t = SPOTBUGS_4_0_0_BETA4
            sourcepath = "-sourcepath " + project_path_appsrc +
"main/java/"
            sourcepath += " -home ./tools/SpotBugs/"
            f1: TextIO = open("./reports/" + t, "w+")
            run = "bash ./tools/SpotBugs/" + t.replace("_", "-") +
"/bin/spotbugs -textui -onlyAnalyze " + \
                  package + " -workHard -longBugCodes " + sourcepath
+ " -output " + get_report_path(t) + \
                  " -auxclasspathFromFile " +
aux_classpath_from_file_path + " " + project_path

            process = subprocess.Popen(run, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
            process.wait()
            f1.close()
                                ...
```

Ispis 2.15 Dio petlje koji pokreće alat SpotBugs, kod razreda *script.py*

Sljedeći korak je bodovanje izvještaja koje su generirali alati, koje je različito za svaki pojedini alat. U nastavku su navedeni načini bodovanja za svaki alat.

SpotBugs: Jedini pronalazi veće sigurnosne probleme, stoga dobiva +10 bodova za svaku prijavljenu grešku. Vrlo često prijavljuje problem s imenovanjem u datoteci R.java, koju ne piše korisnik, stoga su te greške ignorirane (tzv. „false positive“).

Checkstyle: pošto pronalazi samo manje, stilističke greške, dobiva samo +1 bod za svaku prijavljenu pogrešku. Neke greške (linije duže od 80 znakova, varijable koji bi mogle biti *final* i slično) se uopće ne boduju, jer se ne smatraju dovoljno bitnima.

PMD: dobiva +1 bod za svaku prijavljenu grešku, osim nekih koje se smatraju manje bitnima (lokalne varijable koje bi mogle biti *final,* parametri koji bi mogli biti *final* i slično).

CPD: nema bodovanja.

Graudit: ne prijavljuje do koje je greške došlo nego samo ispisuje problematičan kod, stoga se svaka greška boduje s +1, a ukupni broj se podijeli s 2 i zaokruži, kao svojevrsna kazna za nepreciznost alata.

Zadnji bitni element skripte je generiranje html izvještaja. Kao pomoć pri tome se koristio modul *htmlgen* [15]. Uz pomoć tog modula, te dvije *css* datoteke, dizajniran je html izvještaj čiji je skraćeni izgled na slici 2.1.

Static source code analysis report

1. spotbugs_4.0.0_beta4
2. pmd
3. cpd
4. checkstyle
5. graudit
6. table

spotbugs_4.0.0_beta4

M B NM_CLASS_NAMING_CONVENTION Nm: The class name jakhar.aseem.diva.R$bool doesn't start with an upper case letter At R.java:[line 4093]

...

pmd

/home/antonio/Documents/Projekt/diva-android/app/src/main/java/jakhar/aseem/diva/APICreds2Activity.java:35:
Avoid unused imports such as 'android.os.Environment'

...

cpd

Found a 15 line (110 tokens) duplication in the following files:

...

checkstyle

- android.support.design.widget.Snackbar. [UnusedImports]

...

graudit

/home/antonio/Documents/Projekt/diva-android/app/src/main/java/jakhar/aseem/diva/InsecureDataStorage1Activity.java-33-

...

table

TOOL	SCORE
spotbugs_4.0.0_beta4	150
pmd	136
cpd	no score
checkstyle	109
graudit	31

Slika 2.1 Skraćeni izgled *html* izvještaja

3. Dodaci za Android Studio

Ovo poglavlje upoznaje čitatelja s osnovnim konceptima pri izradi dodataka za bilo koji IDE baziran na IntelliJ IDEA-i. Opisuje se izrađivanje dodatka uz pomoć alata Gradle, upoznaje se s elementima grafičkog sučelja koje dodatak tipično koristi ili proširuje te nekim drugim mogućnostima dodataka. Drugi dio poglavlja opisuje izradu jednostavnog dodatka za Android Studio, nazvanog *HelloWorld*.

3.1 Općenito o dodacima za IntelliJ platformu

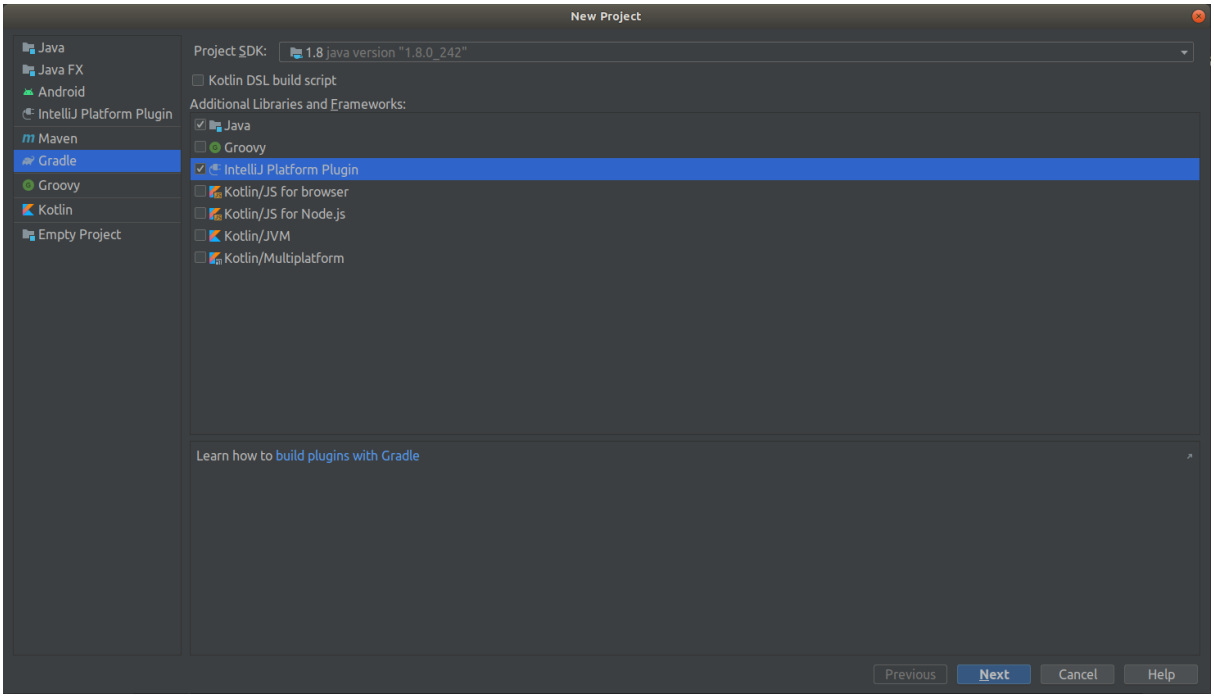
IntelliJ platformu je razvila tvrtka JetBrains. Ona je otvorenog koda, te služi za izgradnju IDE-a. Osnova IntelliJ platforme je Intellij IDEA. Android Studio dodaci, kao i dodaci za sve IntelliJ IDE-e, pišu se u IntelliJ IDEA ili IntelliJ IDEA Ultimate pomoću programskih jezika Jave i Kotlina, ili samo jednog od navedenih [4]. Među ostalim IDE-ima koji se proširuju na ovaj način su: Pycharm, AppCode, PhpStorm, IntelliJ IDEA Ultimate, Rider, RubyMine... Moguće je napraviti dodatak koji će raditi na više IntelliJ IDE-a oktuženja odjednom, međutim taj postupak nije opisan u ovom radu.

Dodaci uobičajno pripadaju jednoj od sljedećih skupina: podrška za novi programski jezik (engl. custom language support), integracija novog alata (engl. tool integration), nadogradnja za korisničko sučelje (engl. user interface add-ons) i integracija radnih okvira (engl. framework integration). Postoje dvije vrste gradnje projekata za dodatke, svaka vrsta koristi jedan pomoćni alat. Prvi alat je DevKit, koji je prije bio standardno korišten za gradnju dodataka. No danas se preporuča korištenje alata Gradlea. U slučaju da se radi dodatak za Android Studio pomoću Jave, pri izradi projekta se zbog kompatibilnosti odabire Java 1.8.

Koraci za izradu novog projekta za izgradnju dodatka su sljedeći:

- instalacija IntelliJ IDEA Community Edition
- omogućavanje dodataka Gradle i Plugin DevKit
- stvaranje novog projekta
- podešavanje IntelliJ SDK-a (engl. software development kit)

Instalacija IntelliJ IDEA Community Edition (ponekad skraćeno na IntelliJ IDEA CE) je jednostavna na svim operacijskim sustavima [16]. Omogućavanje dodataka se radi u *Plugins* odjeljku, do kojeg se dolazi preko izbornika *File ->Settings*. Zatim se sa lijeve strane odabere kartica *Plugins*, te se instaliraju i omoguće Gradle i Plugin DevKit dodaci. Potom se stvara novi projekt pomoću *File->New->Project.* Odabere se kartica *Gradle,* za Project SDK se postavi neki SDK koji koristi JDK 1.8 (engl. Java development kit). Naravno, za to je potrebno imati instaliran odgovarajući JDK na računalu. Izaberu se opcije *Java* i *IntelliJ Platform Plugin*, ide se dalje pritiskom na gumb *Next*, te zatim odabiranjem lokacije projekta i pritiskom na *Finish* završava inicijalizacija projekta. Prozor iz ovog postupka je prikazan na slici 3.1.



Slika 3.1 Prozor za izradu projekta

U projektu je sada stvoreno nekoliko poddirektorija i datoteka. U *src/main/resources/META-INF* direktoriju se nalazi konfiguracijska datoteka dodatka, *plugin.xml*. U isti direktorij se postavlja ikona dodatka, koja striktno mora biti imenovana *icon.svg*. *.svg* vektorska ikona se može generirati na mnogo načina, na primjer u programu *Adobe Illustrator*. Ikonu nije nužno postavljati, no preporučeno je. Na ispisu 3.1 je dan izgled *plugin.xml* čim se ona generira, uz pretpostavku da je ime projekta *HelloWorld*.

Glavni elementi datoteke su *id*, *name*, *vendor*, *description*, *depends*, *extensions*, *actions*. *id* je jedinstveni identifikator dodatka, dok element *name* definira ime koje će biti prikazano. *version* i *vendor* sadrže podatke o verziji i autoru dodatka. *description* definira kratki opis funkcionalnosti. Vrijednosti navedene u dosad objašnjenim elementima prikazuju se prilikom pristupanja dodatku koji je objavljen (engl. published) na JetBrains repozitoriju ili lokalno uključen u ciljani IDE [17], u ovom slučaju Android Studio. Više o načinima distribucije dodataka bit će rečeno u poglavlju 3.2. Element *depends* definira koja se skupina funkcionalnosti proširuje. Neke od mogućnosti koje su dostupne na bilo kojem IDE-u izvedenom iz IntelliJ platforme su *com.intellij.modules.platform* (proširuje funkcionalnosti poruka, izgleda, komponenti sučelja, akcija, dokumenata i datoteka, servisa...), *com.intellij.modules.lang* (proširuje funkcionalnosti referenci, leksera i parsera, automatskog nadopunjavanja koda...), *com.intellij.modules.xdebugger* (proširuje funkcionalnosti *debuggera*) i drugi. Popis svih mogućnosti je dostupan na [18]. *extensions* element definira koji se elementi IDE-a proširuju, te omogućuje referenciranje Java razreda kojima se element proširuje. Neki od osnovnih parametara koji se mogu postaviti u *extensions* elementu su *id*, *order*, *os*. *id* je identifikator, *order* određuje na kojem mjestu se pojavljuje element (može biti *first*, *last*, *before* id, *after* id). *os* određuje operacijski sustav (npr. *os="linux"*). Zadnji element je *actions*, unutar kojeg se mogu registrirati akcije koje se zatim mogu dodjeljivati izbornicima glavne alatne trake sučelja.

```
<idea-plugin>
    <id>org.example.HelloWorld</id>
    <name>Plugin display name here</name>
    <vendor email="support@yourcompany.com"
url="http://www.yourcompany.com">YourCompany</vendor>

    <description><![CDATA[
    Enter short description for your plugin here.<br>
    <em>most HTML tags may be used</em>
    ]]></description>

    <!-- please see
https://www.jetbrains.org/intellij/sdk/docs/basics/getting_started/pl
ugin_compatibility.html
         on how to target different products -->
    <depends>com.intellij.modules.platform</depends>

    <extensions defaultExtensionNs="com.intellij">
        <!-- Add your extensions here -->
    </extensions>

    <actions>
        <!-- Add your actions here -->
    </actions>
</idea-plugin>
```

Ispis 3.1 Konfiguracijska datoteka *plugin.xml*

Što se tiče programiranja dodatka, važno je shvatiti da se iz koda dodatka koji se piše ne može pristupiti dokumentima, priručnim datotekama i drugim resursima koje programer ima na svojem računalu. Sve što postoji u okolini dodatka su resursi korisnikovog računala, odnosno korisnikovog Android Studio programa. Zbog toga način rješavanja pojedinih problema vezanih uz kodiranje dodatka treba prilagoditi takvom načinu razmišljanja.

3.2 HelloWorld dodatak

Cilj *HelloWorld* dodatka je proširiti *ToolWindow* element i napraviti prozor na kojem je jednostavno ispisan tekst „Hello World“. Međutim prvo će se pogledati drugi česti način proširivanja, a to je registriranjem akcija putem *actions* elementa, kako bi se predstavila dva načina interakcije dodatka s korisnikom.

Proširivanjem *actions* elementa moguće je u izbornike dodati željenu stavku. Na primjer u glavni izbornik *Tools* se dodaje stavka koja kad je odabrana pokreće zadani Java razred. Primjer registriranja akcije je dan na ispisu 3.2, a na ispisu 3.3 je kostur Java razreda koji implementira akciju. Taj razred mora proširivati razred *AnAction* i implementirati metodu *public void actionPerformed(AnActionEvent event),* a može (i poželjno je) da implementira metodu *public void update(AnActionEvent e).* U *actionPerformed* se, koristeći *AnActionEvent* dobiven kao paramtear, formira akcija koja se želi napraviti. U metodi *update* se za trenutni kontekst parametra *AnActionEvent* akcija osvježuje (engl. update), na primjer omogućuje ili onemogućuje.

```
<actions>
      <group id="HelloWorld.TestMenu" text="Test
Menu"description="Test">
      <add-to-group group-id="MainMenu" anchor="first" />
      <action id="HelloWorld.Test" class="Test" text="Test"
description="Test item" />
      </group>
</actions>
```

Ispis 3.2 Registriranje akcija u *plugin.xml*

```
public class TestMenu extends AnAction {

      @Override
      public void actionPerformed(AnActionEvent event) {
      // Using the event, implement an action. For example, create
      and show // a dialog.
      }

      @Override public void update(AnActionEvent e) {
      // Using the event, evaluate the context, and enable or disable
      the // action.
}
```

Ispis 3.3 Razred *TestMenu*

Drugo vrlo često korišteno proširenje sučelja je dodavanje elementa *ToolWindow*. Ovo će se koristiti pri gradnji *HelloWorld* dodatka, kao i u poglavlju 4 pri gradnji pravog dodatka. Na ispisu 3.4 dan je primjer koda *extensions* elementa ako se želi stvoriti prozor.

```
<extensions defaultExtensionNs=”com.intellij”>
        <extension os=”linux”/>
        <extension order=”last”/>
        <toolWindow id=”SCAT” secondary=”false” anchor=”right”
factoryClass=”fer.zavrsni.src.window.ToolsWindowFactory”/>
</extensions>
```

Ispis 3.4 Kod za definiranje tvornice prozora

Prvo se definiraju prije spomenuti parametri *os* i *order*, a zatim se registrira prozor pomoću elementa *toolWindow*. Prozoru se mogu definirati parametri *id, anchor, secondary, factoryClass, icon, conditionClass.* Ovdje je definiran identifikator (*id=“SCAT“*), pozicija prozora (*anchor=“right“*) te *secondary* koji označava treba li prozor biti postavljen u donjem dijelu prozorske trake ili ne. Ovdje je *secondary* opcija stavljena na *false,* odnosno prozor se pojavljuje u gornjem dijelu trake. *factoryClass* definira koji Java razred je tvornica prozora. To je razred koji izrađuje prozor, te specificira kako će prozor izgledati i koje funkcionalnosti će imati. *ToolsWindowFactory* je ulazna točka u kodu kad korisnik pritisne ikonu prozora kako bi ga proširio. Razred koji je tvornica prozora mora proširivati *ToolWindowFactory* i implementirati tri funkcije, od kojih je jedna *deprecated* stoga se ne koristi. Najbitnija je funkcija *createToolWindowContent(@NotNull Project project, @NotNull ToolWindow toolWindow)* koja se poziva kod kreiranja prozora. U njoj se definira izgled i funkcionalnosti prozora. U ovom primjeru se stvara objekt tipa *JPanel,* na njega se dodaje labela *JLabel l* na kojoj piše željeni tekst „Hello World“. Iz *ToolWindow toolWindow* objekta koji se dobiva kao parametar uzima se *JComponent* objekt metodom *getComponent(),* te se na njega dodaje *JPanel p.* Na ispisu 3.5 je prikazan prethodni primjer.

```
public class ToolsWindowFactory implements ToolWindowFactory {

    @Override
    public void createToolWindowContent(@NotNull Project project,
@NotNull ToolWindow toolWindow) {
  JComponent parent = toolWindow.getComponent();
  JPanel p = new JPanel();
  JLabel l = new JLabel("Hello World");
         p.add(l);
  parent.add(p);
    }

    @Override
    public void init(@NotNull ToolWindow window) {

    }

    @Override
    public boolean shouldBeAvailable(@NotNull Project project) {
        return true;
    }

}
```

Ispis 3.5 Kod razreda *ToolsWindowFactory*

Sada se može isprobati funkcionalnost dodatka. Za to se moraju definirati svi elementi u *plugin.xlm* datoteci, kako je to već prije opisano. Također, u datoteci *build.gradle* se specificira kako se pokreće dodatak. Kod takve datoteke je dan na ispisu 3.6.

```
plugins {
    id 'java'
    id 'org.jetbrains.intellij' version '0.4.17'
}

group 'org.example'
version '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.12'
}

intellij {
    // Define IntelliJ Platform against which to build the plugin
project.
    version '192.7142.36'  // Same IntelliJ IDEA version as target
Android Studio
    type 'IC'              // Use IntelliJ IDEA CE because it's the
basis of the IntelliJ Platform
    // Require the Android plugin, Gradle will match the plugin
version to intellij.version
    plugins 'android'
}

runIde {
    // Absolute path to installed target Android Studio to use as IDE
Development Instance
    ideDirectory '/snap/android-studio/current/android-studio'
}

patchPluginXml {
    changeNotes """
      This is the initial test release of HelloWorld plugin."""
}
```

Ispis 3.6 Sadržaj *build.gradle* datoteke

Osim uobičajenih elemenata Gradle dokumenta [19], potrebno je imati i *intellij* te *runIde* elemente. U *intellij* elementu se definiraju specifičnosti ciljanog IDE-a. *version* parametar je broj oblika „xxx.xxxx.xx“, koji se može dobiti otvaranjem ciljanog Android Studio programa te odabirom *Help->About.* Otvara se prozor kao na slici 3.2, gdje je traženi broj označen crvenom bojom.

Slika 3.2 Android Studio *About* prozor

type parametru se vrijednost postavlja na *IC* (kratica od *IDEA Community*), a *plugins* parametru je vrijednost *android.* Na taj način će Gradle sam spojiti verzije trenutno korištenog IntelliJ IDEA i ciljanog Android Studia. *runide* element definira parametar *ideDirectory* kojemu se za vrijednost postavlja putanja do instalacije Android Studia koji se želi pokrenuti. Sada pokretanjem *Run* kreće izgradnja Gradle projekta i njegovih resursa, te naposlijetku pokretanja specificiranog Android Studia s prisutnim prozorom izgrađenim u dodatku.

4. Static Code Analysis Tools (SCAT) dodatak

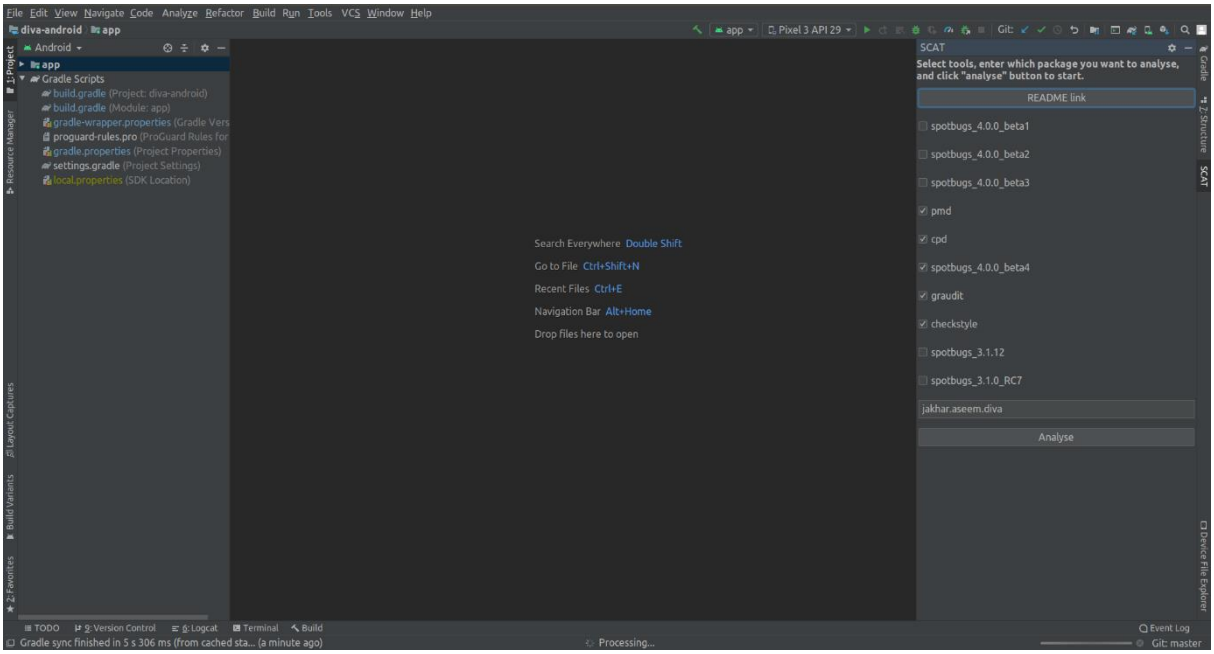
U ovom poglavlju se nadogradnjom osnovnih elemenata prezentiranih u prošlom poglavlju gradi dodatak koji analizira trenutno aktivni projekt i izrađuje izvještaj. Prvo se opisuju funkcionalnosti i uporaba dodatka, zatim se postepeno gradi prozor koji će biti dio Android Studio sučelja, te se prikazuje način na koji je rješeno pokretanje analize preko pozivanja naredbenog retka unutar dodatka i izvršavanja naredbi u njemu. Na kraju se opisuje kako se dodatak lokalno distribuira, te kako se uključuje u ciljani Android Studio.

4.1 Funkcionalnosti dodatka

Dodatak SCAT [21] služi kao sučelje za lakšu interakciju sa skriptom opisanom u 2. poglavlju. Za rad dodatka je potrebno klonirati (engl. clone) projekt *Script* s GitHub repozitorija [14] te ga postaviti u vršni direktorij Android Studio projekta koji se želi analizirati. Ako se želi analizirati projekt imena „Application“ koji se nalazi na putanji */home/user/Documents/Application,* tada je skriptu potrebno postaviti na putanju */home/user/Documents/Application/Script.*

Osim preuzimanja skripte, još su dva preduvjeta za rad skripte. Korisnik mora na svom računalu imati instaliran i pravilno konfiguriran *Python3,* kako bi se iz naredbenog retka naredbom *python3 [arguments]* moglo pokretati alate. Također, projekt mora imati napravljene sve izvršne datoteke prije nego se pokrene analiza (tzv. *build*). Neki alati koriste izvršne *.class* datoteke za provedbu svoje analize, te će se bez provođenja postupka gradnje projekta prije analize najvjerojatnije dobiti nezadovoljavajući izvještaj. Generalno je djelovanje dodatka u takvim uvjetima nedefinirano.

Izgled cjelokupnog prozora koji je integriran u Android Studio je dan na slici 4.1.



Slika 4.1 SCAT prozor unutar Android Studia

Dodatak omogućuje odabir alata koji se žele pokrenuti označavanjem kućica (engl. check box) uz pripadajuću oznaku. Dostupni alati su sljedeći:

- spotbugs_3.1.0_RC7
- spotbugs_3.1.12
- spotbugs_4.0.0_beta1
- spotbugs_4.0.0_beta2
- spotbugs_4.0.0_beta3
- spotbugs_4.0.0_beta4
- pmd
- cpd
- graudit
- checkstyle

U tekstovno polje (Java element *JTextField*) se upisuje paket koji se želi analizirati. U slučaju pogrešnog unosa imena paketa se ne obavještava korisnika o greški, pošto će se analiza uredno izvršiti i s pogrešnim *package* argumentom. No u tom slučaju je izgledno da će neki alati producirati lošije izvještaje. Međutim sami alati neće prijaviti grešku ili prekinuti rad, stoga je odlučeno da to neće raditi niti skripta niti dodatak.

Pritiskom na gumb *Analyse* se pokreće analiza projekta. Na traci iznad statusne trake se tada pojavljuje poruka „Processing...“, koja tamo ostaje sve dok dodatak ne izvrši analizu do kraja. Prikaz te trake je na slici 4.2. Također, pored natpisa je i ikona pomoću koje se može prekinuti postupak. Kad je analiza završena, otvara se pomoćni prozor (Java element *JOptionPane*) s porukom o uspješnom završetku analize i putanjom do izvještaja, koji se nalazi u vršnom direktoriju projekta *Script* i zove *report.html*. Također, u slučaju greške pri unosu argumenata ili kakve druge pogreške, u sličnom pomoćnom prozoru se dobiva prikladna poruka o greški te se analiza prekida.

Slika 4.2 Traka za vrijeme trajanja analize projekta

Nije preporučeno pokretati analizu koja uključuje više od jedne verzije alata SpotBugs. Taj alat je vrlo spor, te se već samostalnim pokretanjem samo jedne verzije tog alata na izvještaj može čekati više od pet minuta.

Na vrh prozora, odmah ispod naslova su kratke upute za korištenje. Ispod toga je postavljen *README* gumb koji na pritisak otvara internetski preglednik i u njemu GitHub stranicu dodatka [21].

4.2 Tvornica prozora

U poglavlju 3.2 objašnjeno je stvaranje prozora pomoću nasljeđivanja razreda tvornice prozora, *ToolWindowFactory*. No često se u većim projektima stvaraju dva razreda: jedan osnovni koji proširuje *ToolWindowFactory*, te drugi koji se poziva iz konstruktora prvoga. Drugi razred je pritom onaj koji izrađuje sve potrebne komponente i funkcionalnosti. U takvom pristupu bi kod osnovnog razreda, nazvanog *ToolsWindowFactory*, izgledao kao na ispisu 4.1.

```
public class ToolsWindowFactory implements ToolWindowFactory {

    @Override
    public void createToolWindowContent(@NotNull Project project,
@NotNull ToolWindow toolWindow) {
        ToolsWindow toolsWindow = new ToolsWindow(project,
toolWindow);
        ContentFactory contentFactory =
ContentFactory.SERVICE.getInstance();
        Content content =
contentFactory.createContent(toolsWindow.getContent(), "", false);
        toolWindow.getContentManager().addContent(content);
    }

    @Override
    public void init(@NotNull ToolWindow window) {

    }

    @Override
    public boolean shouldBeAvailable(@NotNull Project project) {
        return true;
    }

}
```

Ispis 4.1 Kod razreda *ToolsWindowFactory*

Prvo se izrađuje instanca razreda u kojem se obavlja „posao“ (razred *ToolsWindow*), te se zatim iz njega metodom *getContent()* dohvaća vršna komponenta, Java element *JPanel*. *ToolWindow* komponenta dobivena kao argument funkcije *createToolWindowContent* ima svoj *ContentManager* objekt, pomoću čije se metode *add()* dodaje *JPanel* element. Kostur razreda *ToolsWindow* dan je na ispisu 4.2.

```
public class ToolsWindow {
            private JPanel toolsWindowContent;
       ...

            public ToolsWindow(@NotNull Project project, @NotNull
ToolWindow toolWindow) {
              createComponents();
            }

            private void createComponents() {
              toolsWindowContent = new JPanel();
         ...
            }

            public JPanel getContent() {
              return toolsWindowContent;
            }
}
```

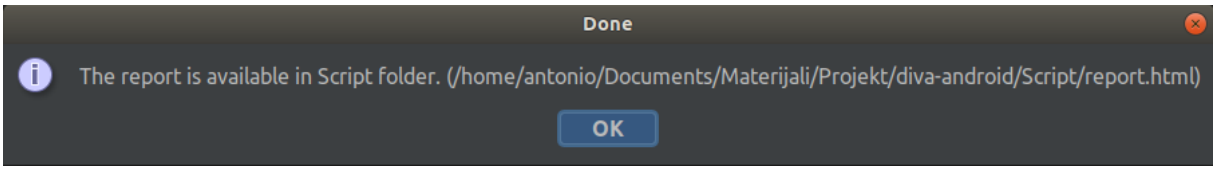
Ispis 4.2 Kostur razreda *ToolsWindow*

U metodi *createComponents()* se gradi sam prozor. Dodaje se *JLabel* element imena *header* koji će sadržavati kratke upute za korištenje. Taj element se podebljava i centrira pomoću *HTML* oblikovanja, kao što je prikazano na ispisu 4.3.

```
header = new JLabel("<HTML><strong>" + Util.SHORT_README +
"</strong></HTML>", JLabel.CENTER);
```

Ispis 4.3 Dodavanje *README* elementa

Zatim se dodaje gumb koji na pritisak otvara internetski preglednik i GitHub stranicu dodatka. Ta se funkcionalnost može postaviti pomoću *addButtonListener* metode, unutar koje se stvara anonimni razred koji pomoću *browse(URI uri)* metode razreda *BrowserUtil* ostvaruje željenu funkcionalnost. U slučaju neuspjeha, hvata se iznimka te se korisniku javlja pogreška pomoću *JOptionPane.showMessageDialogue* metode, koja stvara pomoćni prozor za dijalog, prikazan na slici 4.3. Ovakav način interakcije s korisnikom općenito će se koristiti za prijavljivanje grešaka. Prethodni postupak je predočen na ispisu 4.4.



Slika 4.3 Primjer elementa *JOptionPane*, prikazanog u slučaju uspješnog završetka analize

```
        readmeLink = new JButton();
        readmeLink.setText("README link");
        readmeLink.addActionListener(e -> {
            try {
                BrowserUtil.browse(new URI("https://github.com/Spooky-
M/SCAT"));
            } catch (URISyntaxException uriSyntaxException) {
                JOptionPane.showMessageDialog(null,
                        "Something went wrong while accessing URI. " +
                                "Please contact developers for further
information and questions.",
                        "Error", JOptionPane.INFORMATION_MESSAGE);
                return;
            }
        });
        toolsWindowContent.add(readmeLink);
```

Ispis 4.4 Kod razreda *ToolsWindow*, kreiranje *README* gumba

Pomoću unaprijed pripremljene liste *String* reprezentacija svih alata (konstanta smještena u *Util* razredu), na osnovni *JPanel* se dodaje onoliko *JCheckBox* elemenata koliko ima alata. Svakom se elementu pridjeljuje mnemonička vrijednost, i to po indeksu koji je imao u listi, kao što je prikazano na ispisu 4.5.

```
        boxes = new ArrayList<>();
        for(String tool : tools.keySet()) {
            JCheckBox toolCheckBox = new JCheckBox(tool, false);
            boxes.add(toolCheckBox);
            toolCheckBox.setMnemonic(tools.get(tool));
            toolsWindowContent.add(toolCheckBox);
        }
```

Ispis 4.5 Kod razreda *ToolsWindow*, kreiranje *JCheckBox* elemenata

Dalje se dodaje element *JTextField* koji korisniku služi za upisivanje paketa koji se želi analizirati. Sad se izvršava glavni dio, odnosno dodaje se gumb za započinjanje analize. Njemu se također pridjeljuje *ButtonListener*, na sličan način kao i *README* gumbu, što je prikazano na ispisu 4.6.

```
        analyseButton = new JButton("Analyse");
        analyseButton.addActionListener(e -> {
            analyseButton.setEnabled(false);
            List<String> selectedBoxes = new ArrayList<>();
            for(JCheckBox checkbox : boxes) {
                if(checkbox.isSelected()) {
                    selectedBoxes.add(checkbox.getText());
                }
            }

            ProjectAnalysis pa;
            try {
                pa = new ProjectAnalysis(project, selectedBoxes,
packageChooser.getText());
                pa.executeAnalysis();
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null,
                        "Something went wrong while running script. " +
                                "Please contact developers for further
information and questions.",
                        "Error", JOptionPane.INFORMATION_MESSAGE);
                return;
            }

            analyseButton.setEnabled(true);
        });
        toolsWindowContent.add(analyseButton);
```

Ispis 4.6 Kod razreda *ToolsWindow*, kreiranje *Analyse* gumba

Na pritisak *Analyse* elementa se prvo onemogući ponovni pritisak gumba, a zatim se u petlji provjerava koji od *JCheckBox* elemenata su označeni (metoda *isSelected()*), te se svi oni stavljaju u novu listu. Potom se stvara instanca razreda *ProjectAnalysis* i zove metoda *executeAnalysis()*, koji će biti objašnjena u sljedećem potpoglavlju. Konstruktoru *ProjectAnalysis* predajemo tri argumenta: trenutni projekt, list označenih alata i upisani paket. U slučaju pogreške se otvara *JOptionPane* dijalog s odgovarajućom porukom. Na kraju se pritisak na gumb ponovno omogućuje, te se element postavlja na vršni *JPanel*.

4.3 Analiza projekta

Cijela analiza se provodi u razredu *ProjectAnalysis*, točnije njegovoj metodi *executeAnalysis()* koja se poziva na svaki pritisak gumba *Analyse*. Jedini konstruktor, čiji je kostur prikazan na ispisu 4.7, prima tri argumenta. Prvi je argument isti onaj koji se prima u konstruktorima *ToolsWindowFactory*, odnosno *ToolsWindow* razreda. Njega se ne kreira, nego inherentno dobiva, i on označava trenutno aktivni projekt u korisnikovom sučelju. *tools* argument je lista označenih alata, dok je *packageName* ime paketa koje je korisnik unio u prije spomenuti *JTextField*.

```
public ProjectAnalysis(@NotNull Project project, List<String> tools,
String packageName) throws IOException {
  ...
}
```

Ispis 4.7 Izgled konstruktora razreda *ProjectAnalysis*

U konstruktoru se najprije formira *String* reprezentacija apsolutne putanje do projekta na računalu. Za to služi metoda *getBasePath(),* koja se poziva nad primljenim *Project* objektom. Zatim se radi provjere ovaj *String* pretvara u *Path* objekt, te se poziva metoda *Files.exists* kako bi se provjerila valjanost putanje. U slučaju greške, otvara se *JOptionPane* prozor s prikladnom porukom. Za dohvaćanje *Script* direktorija, koji mora biti smješten u vršnom direktoriju projekta, se koriste mogućnosti razreda *LocalFileSystem.* Instancu objekta se dobiva pozivom *getInstance(),* a zatim se od instance traži da pronađe direktorij prema predanoj putanji. Putanja se konstruira od prije dohvaćene putanje projekta i sufiksa „Script“. Ponovno se provjerava valjanost putanje i prijavljuje moguća greška. Sama ulazna točka je skripta *script.py,* koja se nalazi unutar direktorija *Script.* Pomoću putanje do direktorija *Script* i sufiksa „script.py“ se dohvaća i zatim provjerava putanja do skripte. Zadnji korak u konstruktoru je stvaranje datoteke *auxClasspathFromFile* pomoću *Files.createFile* kojoj se kao argument predaje putanja do skripte.

Metoda *executeAnalysis* izvršava cijelu analizu. Kad se pokrene, prvo se iz dobivenog *tools* argumenta konstruira jedan *String* objekt koji sadrži sve označene alate razdvojene zarezom, bez razmaka. SDK trenutnog projekta se može dobiti pomoću *ProjectRootManager* razreda. Nakon dohvaćanja njegove instance, poziva se metoda *getProjectSdk. String* oblik putanje, koji se zapravo koristi u daljnjem kodu, se dobiva metodom *getHomePath().* Sličnim se nadovezivanjem direktorija projekta i sufiksa „gradle“ dobiva putanja do direktorija koji kreira Gradle Android Studio projekta. Ove dvije putanje se koriste u sljedećem koraku, dodavanju stavki u *auxClasspathFromFile* datoteku.

```
    private void writeToAuxClasspathFile(String gradlePath, String sdkPath)
throws IOException {
        BufferedWriter bw = Files.newBufferedWriter(auxClasspathFromFile);
        bw.write(gradlePath + "\n" + sdkPath + "\n");
        bw.flush();
        bw.close();
    }
```

Ispis 4.8 Kod *writeToAuxClasspathFile* metode

U ispisu 4.8 je metoda *writeToAuxClasspathFile* metoda, koja pomoću argumenata (*String* reprezentacije putanja do direktorija *gradle* i trenutnog SDK direktorija) u prije stvorenu datoteku zapisuje u dva reda putanje. U poglavlju 2 je objašnjeno zašto je ta datoteka potrebna alatima za analizu izvornog koda.

Sada su dostupni svi potrebni podaci za pokretanje skripte. Zato se gradi naredba kojom će se pozvati skripta iz naredbenog retka. Pojedini dijelovi naredbe se spremaju u listu *String* objekata, kao na ispisu 4.9.

```
List<String> cmds = new ArrayList<>();
cmds.add("python3");
cmds.add(scriptPath.toAbsolutePath().toString());
cmds.add("-t");
cmds.add(toolsArg.toString());
cmds.add("-c");
cmds.add(auxClasspathFromFile.toAbsolutePath().toString());
cmds.add("-p");
cmds.add(packageName);
cmds.add(projectRootPath.toAbsolutePath().toString());
```

Ispis 4.9 Kod *ProjectAnalysis* razreda, izgradnja naredbe za poziv skripte

Pokretanje skripte se mora napraviti u posebnom procesu, jer se želi omogućiti korisniku da radi druge stvari dok se odvija analiza. Pokreanjem iz istog procesa bi se zablokirao cijeli Android Studio sve dok analiza ne završi, a pošto to može trajati i nekoliko minuta, to bi bilo vrlo loše rješenje. Posebni proces se u kodu može pokrenuti pomoću *run* metode *ProgressManager* razreda, kojoj se predaje *Task.Backgroundable* objekt. Pokretanje jednog takvog postupka pomoću anonimnog razreda kojemu je implementirana metoda *run* je prikazano na ispisu 4.10.

```
        ProgressManager.getInstance().run(new Task.Backgroundable(project,
"Analysis") {
            @Override
            public void run(@NotNull ProgressIndicator progressIndicator) {
                ...
            }
        });
```

Ispis 4.10 Kod *ProjectAnalysis* razreda, pokretanje pomoćnog procesa

Napredak procesa koji se odvija u pozadini se vidi na traci prikazanoj na slici 4.2. U kodu se konkretna poruka te napredak na pomičnoj traci mogu mijenjati korištenjem metoda *setText(String text)* i *setProgress(int progress)*, pri čemu *progress* mora biti postotak, odnosno vrijednost između nula i jedan. Inicijalno je vrijednost postavljena na nula. Indikator napretka se koristi kad postoji nekoliko koraka koji se moraju izvršiti, pa se napredak osvježuje nakon svakog izvršenog koraka. U ovom dodatku se izvodi samo jedan korak, stoga se ne koristi indikator napretka.

U metodi *run* anonimnog razreda implementira se pozivanje naredbenog retka i skripte. Naredbenom retku se iz Java koda može pristupiti, te iz njega izvoditi naredbe, na nekoliko načina. Ovdje je izabrana kombinacija razreda *GeneralCommandLine* i *OSProcessHandler*. Prvi razred predstavlja naredbeni redak. Koristi se njegov konstruktor koji prima listu *String* objekata, koji predstavljaju dijelove naredbe koju je potrebno izvesti. Tom konstruktoru se predaje prije stvorena lista *cmds* (prikazana na ispisu 4.9). Potom se postavlja način kodiranja metodom *setCharset*, na konstantu *StandardCharsets.UTF_8*. Potrebno je i postaviti putanju direktorija iz kojeg se izvode naredbe, što se može postići metodom *setWorkDirectory*. Metodi se predaje *String* reprezentacija unaprijed pripremljene *scriptFolder* varijable, koja je *Path* objekt s putanjom do direktorija *Script*. *OSProcessHandler* objekt se kreira pomoću konstruktora koji prima baš objekt tipa *GeneralCommandLine*. Metodom *startNotify* se pokreće proces, odnosno izvršavanjem naredbe u naredbenom retku se proslijeđuje izvršavanje same analize skripti. *waitFor* metodom se postavlja i vremenska granica nakon koje se proces prekida, a kao argument je metodi predana konstanta iz razreda *Util*. I u slučaju uspjeha, i u slučaju neuspjeha se otvara pomoćni *JOptionPane* element s odgovarajućom porukom. U slučaju uspjeha se navodi i putanja do generiranog izvještaja, koji se nalazi u direktoriju *Script* kako je prije opisano. Na ispisu 4.11 je dan kod koji izvodi prethodni postupak.

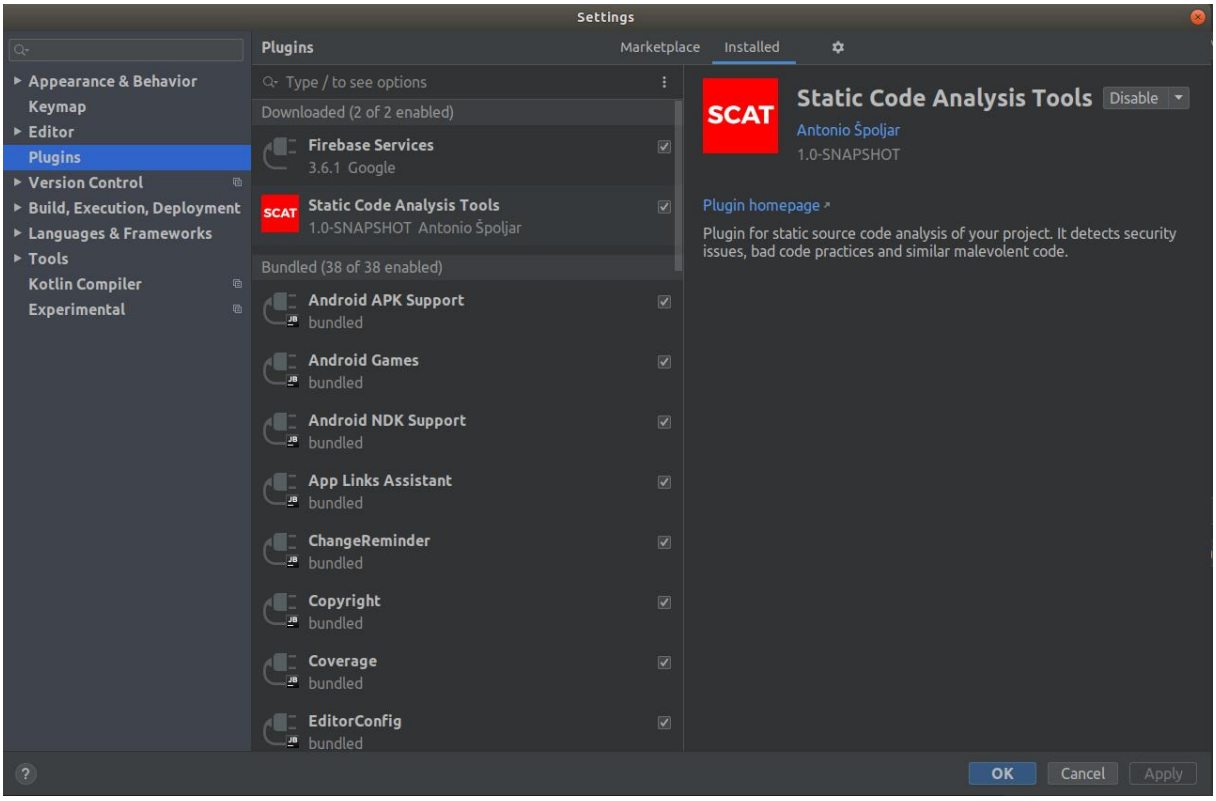
```
                progressIndicator.setText("Processing...");
                try {
                    GeneralCommandLine generalCommandLine = new
GeneralCommandLine(cmds);
                    generalCommandLine.setCharset(StandardCharsets.UTF_8);

generalCommandLine.setWorkDirectory(scriptFolder.getPath());
                    OSProcessHandler processHandler = new
OSProcessHandler(generalCommandLine);
                    processHandler.startNotify();
                    processHandler.waitFor(Util.TIMEOUT);

                    JOptionPane.showMessageDialog(null,
                            "The report is available in Script folder. ("
                                    +
projectRootPath.toAbsolutePath().toString() + "/Script/report.html)",
                            "Done", JOptionPane.INFORMATION_MESSAGE);
                } catch (ExecutionException e) {
                    e.printStackTrace();
                    JOptionPane.showMessageDialog(null,
                            "Something went wrong while running script: " +
e.getMessage(),
                            "Error", JOptionPane.INFORMATION_MESSAGE);
                }
```

Ispis 4.11 Kod razreda *ProjectAnalysis* koji se nalazi unutar *run* metode s ispisa 4.10

4.4 Lokalno distribuiranje dodatka

Nakon završetka izrade i testiranja dodatka, potrebno je isti uključiti u Android Studio. Moguće je objaviti dodatak na IntelliJ trgovini (engl. marketplace), a druga opcija je lokalno pokrenuti dodatak u željenom Android Studiu. Za lokalnu distribuciju se koristi Gradle. Iz vršnog direktorija projekta se pokrene naredbeni redak (može se koristiti i integrirani IntelliJ IDEA naredbeni redak), i izvrši se naredba *gradle buildPlugin.* Alternativa ovoj naredbi je *./gradlew buildPlugin,* obje naredbe izvršavaju istovjetni zadatak. Nakon završetka izgradnje, u poddirektoriju *build/distributions/* se nalazi *.zip* ili *.jar* arhiva.

Dodatak je potrebno uključiti u Android Studio. Pali se željena verzija Android Studia, otvara se *File->Settings->Plugins* kartica, te se u gornjem izborniku izabere *Settings* ikona pored kartica *Marketplace* i *Instelled.* U njoj se pritisne opcija *Install Plugin From Disk*. Prozor izgleda kao na slici 4.4. Izabere se prije stvorena arhiva, te Android Studio zatim učita dodatak. Pri ponovnom pokretanju Android Studia, vidljive su sve komponente kojeg definira dodatak (u ovom slučaju prozor, kao na slici 4.1).



Slika 4.4 Uključivanje dodatka SCAT u Android Studiu

5. Zaključak

U ovom radu je ukratko objašnjena statička analiza izvornog koda, te su nabrojani neki alati koji se mogu koristiti pri analizi sigurnosti i kvalitete koda projekata rađenih u Android Studiu, ali i samostalnih Java datoteka. Konstruirana je skripta koja pokreće nekoliko takvih alata te kombinira njihove izvještaj u jedan konačni, kojeg korisnik čita. Opisane su glavne komponente dodatka koji se izrađuje u IntelliJ IDEA-i, te su predstavljena dva elementa kojima se može proširiti Android Studio. Napravljen je i objašnjen *HelloWorld* dodatak, te je pomoću koncepata korištenih u njemu ostvaren i SCAT dodatak koji u sučelje Android Studia dodaje jedan prozor (element *ToolWindow*). U prozoru su dostupne opcije i gumb za pokretanje. Nakon pokretanja, dodatak korištenjem skripte analizira trenutno aktivni projekt i producira izvještaj.

Statistička analiza koda je aktualna u računalnom inženjerstvu, no to je vrlo zahtjevno područje. Postoji mnogo različitih alata u tu svrhu koji imaju raznolike pristupe, zato je ideja kombiniranja nekoliko takvih alata zajedno zanimljiva. U radu se promatra samo mali dio alata koji su bili prikladni, no postoji mogućnost proširivanja SCAT dodatka novim alatima i mogućnostima analize. Na primjer, moglo bi se omogućiti analizu samo jednog označenog Java razreda. Isto tako, dodatak bi mogao proširiti sučelje novim elementima, recimo opcijom u izborniku kojom bi se otvorile postavke dodatka i korištenih alata. U postavkama bi se mogle pružiti bogatije mogućnosti konfiguriranja.

6. Literatura

[1] Statista , J. Clement „Mobile app usage“, 1.8.2019. https://www.statista.com/topics/1002/mobile-app-usage/, pristupljeno 20.5.2020.

[2] Dazeinfo, „1 Billion Android-Powered Devices In 2014: One Out Of Every Two Devices Will Run On Android By 2015 [REPORT]“. https://dazeinfo.com/2014/04/04/android-devices-shipments-worldwide-2014-2015/, pristupljeno 21.5.2020.

[3] Wordnik, internetski rječnik engleskih riječi, „Plugin“. https://www.wordnik.com/words/plug-in, pristupljeno 21.5.2020.

[4] JetBrains, SDK dokumentacija. https://www.jetbrains.org/intellij/sdk/docs/intro/welcome.html, pristupljeno 23.5.2020.

[5] Overops blog, blog Overops kompanije, „Static Vs. Dynamic Code Analysis And How To Choose Between Them“. https://blog.overops.com/static-vs-dynamic-code-analysis-how-to-choose-between-them/, pristupljeno 21.5.2020.

[6] Wikipedia, „List Of Tools For Static Code Analysis“ https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis, pristupljeno 21.5.2020.

[7] Spotbugs, alat za analizu programskog koda. https://github.com/spotbugs/spotbugs, pristupljeno 21.5.2020.

[8] Checkstyle, alat za analizu programskog koda. https://github.com/checkstyle/checkstyle, pristupljeno 21.5.2020.

[9] PMD i CPD, alati za analizu programskog koda. https://github.com/pmd/pmd, pristupljeno 21.5.2020.

[10] Graudit, alat za analizu programskog koda. https://github.com/wireghoul/graudit/, pristupljeno 21.5.2020.

[11] Google GitHub, „Google Java Style Guide“. https://google.github.io/styleguide/javaguide.html, pristupljeno 21.5.2020.

[12] Sun Microsystems Inc. (u vlasništvu Oracle korporacije), „Java Coding Conventions“, 12.9.1997. https://www.oracle.com/technetwork/java/codeconventions-150003.pdf, pristupljeno 21.5.2020.

[13] PMD dokumentacija, „Making rulesets“. https://pmd.github.io/latest/pmd_userdocs_making_rulesets.html, pristupljeno 21.5.2020.

[14] GitHub, „Spooky-M/Script“. https://github.com/Spooky-M/Script, pristupljeno 22.5.2020.

[15] pypi.org, Python modul za generiranje HTML5 koda. https://pypi.org/project/htmlgen/, pristupljeno: 22.5.2020.

[16] JetBrains, instalacija IntelliJ IDEA razvojnog okruženja. https://www.jetbrains.com/idea/download/#section=linux, pristupljeno 23.5.2020.

[17] JetBrains, objavljivanje dodataka građenih Gradle alatom. https://www.jetbrains.org/intellij/sdk/docs/tutorials/build_system/deployment.html?search=publi, pristupljeno 23.5.2020.

[18] JetBrains, kompatibilnost dodataka s IntelliJ platformom https://www.jetbrains.org/intellij/sdk/docs/basics/getting_started/plugin_compatibility.html, pristupljeno 23.5.2020.

[19] Gradle, „Gradle User Manual 6.4.1“. https://docs.gradle.org/current/userguide/userguide.html, pristupljeno 24.5.2020.

[20] JetBrains, „Android Studio Plugin Development“. https://www.jetbrains.org/intellij/sdk/docs/products/android_studio.html, pristupljeno 24.5.2020.

[21] GitHub, „Spooky-M/SCAT“. https://github.com/Spooky-M/SCAT, pristupljeno 22.5.2020.

Razvoj dodatka za omogućavanje statičke analize koda u programu Android Studio

Sažetak

Android aplikacije su danas vrlo raširene, a velik je naglasak stavljen na njihovu sigurnost. Alati koji programeru pomažu pri automatizaciji procesa provjere sigurnosti su alati za statičku analizu izvornog koda. Ovaj rad opisuje izradu dodatka SCAT (Source Code Analysis Tools) za Android Studio koji koristi nekoliko takvih alata. Prvo je ukratko objašnjeno što je to statička analiza izvornog koda, te je predstavljeno nekoliko alata za statičku analizu korištenih u radu. Zatim je prezentirana skripta koja se pokreće u naredbenom retku operacijskog sustava Linux i korištenjem spomenutih alata analizira Android Studio projekt ili Java datoteku. Objašnjeni su osnovni koncepti pri izrađivanju dodataka za Android Studio, te je konstruiran *HelloWorld* dodatak kao primjer. Proširivanjem koncepata korištenih u *HelloWorld* dodatku se postepeno gradi dodatak SCAT, koji je zapravo sučelje korisnika prema skripti. Na kraju je opisano kako se dodatak može uključiti u Android Studio.