Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 659.2:004
DOI: https://doi.org/10.24867/19OI01Nikolic

DETEKCIJA I KOREKCIJA *CODE SMELL* NEPRAVILNOSTI KOD ANDROID MOBILNIH APLIKACIJA

DETECTION AND CORRECTION *CODE SMELL* IRREGULARITIES FROM ANDROID MOBILE APPLICATIONS

Filip Nikolić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – INŽENJERSTVO INFORMACIONIH SISTEMA

Kratak sadržaj – *Kako bi se omogućilo smanjenje potrošnje energije i code smell nepravilnosti koje se nalaze u nekoj mobilnoj aplikaciji kroz ovaj rad je prikazan pristup koji počinje definisanjem problematičnih delova koda koji se pojedinačno nazivaju code smell, a zatim i potrebna detekcija tih code smell nepravilnosti putem alata kao što su aDoctor i JDeodorant, kao i česta refaktorizacija koda u novi oblik bez promene njegove prvobitne funkcionalnosti radi poboljšanja performansi mobilnih aplikacija.*

Ključne reči: *Android aplikacija, Code Smell detekcija, aDoctor, ušteda energije, JDeodorant*

Abstract – *In order to reduce energy consumption and code smell irregularities found in a mobile application, this paper presents an approach that begins by defining problematic pieces of code individually called code smell, and then the necessary detection of these code smell irregularities using tools such as are aDoctor and JDeodorant, as well as frequent refactoring of code into a new form without changing its original functionality to improve the performance of mobile applications.*

Keywords: *Android application, Code Smell Detection, aDoctor, Saving energy, JDeodorant*

1. UVOD

Mobilne aplikacije su u poslednjih par godina ostvarile veoma veliki značaj na polju razvoja softvera, polako preuzimajući primat na tržištu kao jedan od glavnih softverskih sistema koji masovno primenjuje veliki broj korisnika [1]. Aplikacije postaju složenije i razvijaju se brže kako bi se zadržalo interesovanje i opravdalo poverenje krajnjih korisnika, ali i kako bi se omogućile bolje performanse prilikom obavljanja složenih zadataka. U takvom procesu razvoja programeri su često fokusirani na vremenska ograničenja, što može da prouzrokuje izbor neoptimalnog koda koji dovodi do odlaganja čišćenja koda kako bi se ispunili zadati rokovi. Na taj način se ostvaruje uvođenje potrebnog reinženjeringa takvog koda nakon nekog određenog vremenskog perioda. Takođe, ovakve aplikacije se razlikuju od drugih tipova aplikacija po tome što imaju ograničene resurse, kao i po paradigmi programiranja koja je zasnovana na događajima.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Darko Stefanović, vanr. prof.

Problemi koji nastaju prilikom ovakvog razvoja predstavljaju loša dizajnerska rešenja, kao i slaba zadovoljenja zahteva koja mogu da prouzrokuju pojavu različitih vrsta *code smell* nepravilnosti unutar samih aplikacija. *Code smell* nepravilnosti negativno utiču na održavanje softvera i mogu da utiču na potrošnju energije mobilnih telefona na kojima su te aplikacije instalirane. Takođe, prisustvo *code smell* nepravilnosti u kodu predstavlja simptome koji ukazuju da nešto nije u redu sa aplikacijom.
Zbog sve većeg interesovanja za dužim trajanjem baterije, a samim tim i većom uštedom energije od strane mobilnih telefona, cilj ovog rada jeste da predstavi način detekcije i refaktorisanja *code smell* nepravilnosti kroz tri Android mobilne aplikacije razvijene u Java programskom jeziku koje pripadaju *Games* domenu, kako bi se ublažili problemi koji negativno utiču na te i neke druge mobilne aplikacije. U radu se koriste tehnike evolucije softvera kao što je reinženjering, koji predstavlja proces za poboljšanje kvaliteta softvera rekonstrukcijom implementirane aplikacije u novi oblik bez promene njegovih funkcionalnosti.
Zbog toga se kroz loše napisani kod koji može da prouzrokuje energetske probleme vrši proces analize kako bi se pronašli specifični tipovi *code smell* nepravilnosti i uklonili kroz čestu refaktorizaciju, odnosno transformaciju koda radi što bolje optimizacije i poboljšanja kvaliteta softvera nad tim mobilnim aplikacijama.

2. PREGLED LITERATURE

Za potrebe ovog rada prikupljeno je nekoliko radova koji sadrže značajne informacije, koje su kroz neka poglavlja povezane sa načinom detekcije i korekcije *code smell* nepravilnosti unutar Android mobilnih aplikacija, kao i probleme koje mogu da prouzrokuju definisane *code smell* nepravilnosti kroz performanse i energetsku potrošnju. Ovaj odeljak takođe prikazuje i glavne koncepte bitne za ovaj rad, kao što su pojam *code smell* i alati za detekciju.

2.1. Android aplikacije

Android aplikacije predstavljaju softverske aplikacije koje su dizajnirane da rade na mobilnim uređajima, koje pokreće Android platforma. Pošto je Android platforma napravljena za mobilne uređaje, tipične Android aplikacije su najvećim delom dizajnirane za pametne telefone ili tablet računare koji rade na Android operativnom sistemu.
Android aplikacije koje su korišćene u ovom radu obično su razvijene u programskom jeziku Java koristeći alat za razvoj Android softvera odnosno SDK, koji predstavlja

kolekciju moćnih razvojnih alata i biblioteka koje se koriste za razvoj aplikacija za Android platformu. Aplikacije su organizovane kao kolekcija četiri različita tipa osnovnih komponenti, i kao takve aplikacije mogu biti sastavljene od jedne ili više od svake vrste komponenti. Osnovne komponente predstavljaju važne gradivne blokove od kojih je izgrađena svaka od aplikacija, a dostupne komponente koje se javljaju u Android aplikacijama su: *Activities*, *Services*, *Broadcast receivers* i *Content providers.* Android aplikacije se sastoje i iz dva bitna segmenta i to su: funkcionalnost koja predstavlja kod kako se aplikacije ponašaju sa zahtevima i obuhvata sve algoritme koji pokreću aplikaciju, dok drugi bitan segment predstavljaju resursi koji su predstavljeni kroz podatke kao što su tekst, slike, audio i video fajlovi, datoteke, kao i drugi podaci koje aplikacije mogu da koriste.

2.2. Code Smell

Code smell predstavlja strukture u programskom kodu koje ukazuju na dublje potencijalne probleme sa kvalitetom softvera, nad kojima je potrebno izvršiti neki vid korekcije. Pojava *code smell* nepravilnosti u mobilnim aplikacijama uzrokuje probleme sa performansama kao što su prekomerna potrošnja hardverskih resursa ili neki vidovi zastoja i otkazivanja prilikom rada same aplikacije. Postoji više različitih vrsta *code smell* nepravilnosti, a neke od njih se često pojavljuju prilikom razvoja Android aplikacija i to su [2]:

- *Complex Class* — klasa koja sadrži kompleksne metode.
- *Hashmap Usage* — upotreba heš mapa se ne preporučuje prilikom upravljanja malim setovima podataka.
- *Long Parameters List* — predstavlja dugu listu parametara neke metode.
- *Move Method* — predstavlja metodu koja se koristi u nekoliko spoljnih klasa.
- *Duplicated Code* — isti kod koji se koristi na više mesta.
- *Dead Code* — predstavlja kod koji se ne koristi, odnosno to je funkcija bez poziva ili uslov koji se nikada ne javlja.

2.3. Refaktorisanje

Pojava prethodno navedenih problema rešava se kroz refaktorisanje koje predstavlja proces restrukturiranja koda bez promene njegovog spoljnog ponašanja. To je tip korekcije koda kojim se uklanjaju postojeće *code smell* nepravilnosti i kao rezultat se dobija poboljšan dizajn softvera. Odnosno refaktorisanje predstavlja proces prerade izvornog koda sa ciljem očuvanja njegovog spoljnog ponašanja, kroz disciplinovan način čišćenja koda kojim se smanjuju šanse za uvođenje grešaka u kodu i ugrožavanje kvaliteta softvera. Proces refaktorisanja se koristi za unapređenje strukture koda u smislu njegove čitljivosti, složenosti, mogućnosti održavanja, proširivosti i ponovnoj upotrebi. Refaktorisanje malog ciklusa je nešto što bi trebalo da se primenjuje veoma često u kodu, odnosno potreba za čestim uzimanjem metoda kako bi kod postao bolji i pravljenje novih testova kako bi postao čvršći. Svaki korak refaktorisanja može biti vrlo jednostavan i može radikalno poboljšati dizajn implementiranog koda, odnosno kroz refaktorisanje nekog koda moguće je premestiti određene delove jedne klase u drugu, izvlačiti neki deo koda iz određene metode kako bi se kreirala nova metoda, kao i pomeranje nekih delova koda po hijerarhiji. Rezultirajuća interakcija vodi do programa sa dizajnom koji ostaje dobar kako se razvoj nastavlja dalje.

2.4. Alati za detekciju

ADoctor i *JDeodorant* predstavljaju alate za detekciju *code smell* nepravilnosti. Oba ova alata moguće je instalirati kao *plugin* unutar Android Studia i neometano ga na taj način primenjivati. Pristup se vrši prolaskom kroz navigacioni bar i odabirom polja pod nazivom *Refactor* u kome se oni nalaze nakon što su uspešno instalirani. *ADoctor* identifikuje *code smell* nepravilnosti specifične za Android, koristeći apstraktno sintaksno stablo izvornog koda i prolazeći kroz njega primenom različitih pravila otkrivanja. Na taj način omogućavajući analizu bilo koje Java klase koja je sintaksno tačna. Neke od *code smell* nepravilnosti koje podržava *aDoctor* su [3]:

- *Internal Getter and Setter* — indirektan pristup putem getera i setera.
- *Member Ignoring Method* — nestatička metoda koja ne koristi varijable instance i druge nestatičke metode.
- *Public Data* — privatni podaci se čuvaju u skladištu koje je javno dostupno u drugim aplikacijama.
- *Debuggable Release* — postavljanje atributa *debuggable* na *true* predstavlja veliku bezbednosnu pretnju, jer svaka spoljna aplikacija može imati puni pristup izvornom kodu.
- *Durable Wakelock* — metoda sa instancom *PowerManager.Wakelock* stiče *wakelock* bez postavljanja vremenskog ograničenja i naknadnog pozivanja *release* metode i druge.

JDeodorant predstavlja dodatak koji obuhvata niz tehnika za predlaganje i automatsku primenu mogućnosti korekcije nad izvornim kodom u programskom jeziku Java, kroz funkciju koja vrši statičku analizu izvornog koda. *JDeodorant* usvaja *ad-hoc* strategije za svaki *code smell* uzimajući u obzir posebne karakteristike osnovnog problema dizajna koda, za razliku od drugih pristupa koji se oslanjaju na generičke strategije koje se mogu prilagoditi različitim *code smell* nepravilnostima. Neke od *code smell* nepravilnosti koje podržava *JDeodorant* su [4]:

- *Feature Envy* — metoda u jednoj klasi šalje puno poziva ka nekoj drugoj klasi, odnosno metodu više zanima neka druga klasa nego ona u kojoj se nalazi.
- *God Class* — klasa koja sadrži veliki broj atributa i metoda.
- *Type Checking* — česta kompleksna provera tipa sa primenom *typeof* operatora.
- *State Checking* — kompleksni uslovni iskaz koji bira putanju daljeg izvršavanja na osnovu stanja objekta.
- *Long Methods* — metode koje sadrže mnogo više linija koda u odnosu na ostale metode.

3. IZBOR APLIKACIJA I DETEKCIJA PROBLEMA

Izbor Android aplikacija za ovaj rad izvršen je upotrebom *F-Droid* softverskog skladišta za Android aplikacije. Iz-

bor aplikacija podeljen je u tri različite faze. U okviru prve faze izbora aplikacija izvršena je preliminarna ručna pretraga aplikacija vodeći računa da je glavni jezik implementacije koda Java, zato što izabrani alati koji se koriste za dalju detekciju *code smell* nepravilnosti zahtevaju aplikacije razvijene u Java programskom jeziku i da te aplikacije pripadaju *Games* domenu.

Sledeća faza izbora aplikacija predstavlja filtriranje aplikacija na osnovu broja linija koje su definisane da budu u rasponu od jedne hiljade pa do deset hiljada linija koda. Konačno, u okviru treće faze izbora aplikacija izvršeno je preuzimanje prethodno filtriranih aplikacija.

Nakon uspešne primene sve tri faze prikupljene su tri aplikacije nad kojima će biti izvršena dalja analiza energetskog uticaja. Aplikacije koje su izabrane su aplikacije pod nazivom Memo Game, Crazyflie i Block Puzzle Stone Wars [5]. Detektovanje problema izvršeno je upotrebom prethodno navedenih alata za detekciju kroz Android Studio, identifikovani problemi su predstavljeni i rešeni u nastavku rada.

4. ANALIZA ENERGETSKOG UTICAJA I REŠENJE PROBLEMA

Primena prethodno navedenih teorijskih osnova izvršena je praktično nad tri prethodno filtrirane i izabrane Android mobilne aplikacije. Aplikacije su namenjene ljubiteljima igara i kao takve zahtevaju što bolje dizajnerske odluke kako bi se zadržalo interesovanje i opravdalo poverenje od strane njihovih korisnika.

Zbog toga je nad njima primenjena detaljna analiza koda, radi detekcije i kasnijeg refaktorisanja problema u vidu *code smell* nepravilnosti, kako bi se omogućio bolji kvalitet i smanjila potrošnja energije na uređajima na kojima se često koriste. Na samom početku izvršena je analiza potrošnje energije od strane prethodno navedenih aplikacija pre detekcije i rešavanja problema. Analiza napajanja je izvršena nad *Huawei P Smart* mobilnim uređajem i sastojala se od sledećih koraka bitnih za sprovođenje analize potrošnje energije:

- prvi korak *predstavlja postavljanje vremena korišćenja aplikacije od približno 15 minuta,*
- *drugi korak predstavlja merenje potrošnje energije u miliamper* časovima (mAh) za određeno vreme korišćenja aplikacije koje je navedeno u prethodnom koraku.

Ovi koraci se ponavljaju iznova četiri puta i na osnovu dobijenih rezultata izvlači se srednja vrednost za količinu potrošene energije. Analizom prve aplikacije pod nazivom Memo Game, je utvrđeno da je vrednost potrošnje energije tokom prva četiri izvođenja nad prethodno navedenom aplikacijom iznosila: 21,64mAh u prvom izvođenju, 22,84mAh u drugom izvođenju, 21,95mAh u trećem izvođenju i 19,94mAh u četvrtom izvođenju. Na osnovu dobijenih rezultata izračunata je srednja vrednost potrošnje energije pre procesa detekcije i korekcije koja iznosi 21,59mAh.

Analizom druge aplikacije pod nazivom Crazyflie, je utvrđeno da je vrednost potrošnje energije tokom prva četiri izvođenja nad prethodno navedenom aplikacijom iznosila: 29,55mAh u prvom izvođenju, 28,64mAh u drugom izvođenju, 29,98mAh u trećem izvođenju i 29,51mAh u četvrtom izvođenju.

Na osnovu dobijenih rezultata izračunata je srednja vrednost potrošnje energije pre procesa detekcije i korekcije koja iznosi 29,42mAh.

Analizom treće aplikacije pod nazivom Block Puzzle Stone Wars, je utvrđeno da je vrednost potrošnje energije tokom prva četiri izvođenja nad prethodno navedenom aplikacijom iznosila: 10,29mAh u prvom izvođenju, 9,83mAh u drugom izvođenju, 10,54mAh u trećem izvođenju i 10,99mAh u četvrtom izvođenju. Na osnovu dobijenih rezultata izračunata je srednja vrednost potrošnje energije pre procesa detekcije i korekcije koja iznosi 10,41mAh. Nakon izvršene analize, sledeći korak predstavlja detekcija i kasnija korekcija otkrivenih problema koji su nastali prilikom razvoja same aplikacije. Detekcija postojećih *code smell* nepravilnosti je prvo izvršena upotrebom *aDoctor plugin*-a unutar Android Studia, a zatim nakon toga i detaljnom manuelnom analizom na osnovu liste postojećih *code smell* nepravilnosti. Analizom koja je izvršena od strane *aDoctor*-a pronađene su sledeće *smell* instance:

- *Member Ignoring Method — koja se rešava postavljanjem nestatičke metode u statičku, time omogućavajući da metoda bude povezana sa klasom, povećavajući njenu efikasnost.*
- *Internal Setter — koja se rešava direktnim dodeljivanjem promenjive instance umesto poziva metode setter, da bi se smanjio gubitak energije.*
- *Durable Wakelock — koja se rešava direktnim dodeljivanjem naredbe release na kraju bloka izvornog koda gde code smell* instanca *PowerManager.Wakelock* poziva metodu *accept*.

Nakon detekcije izvršeno je rešavanje otkrivenih problema primenom pravila koje predlaže i sam *aDoctor*. Ovaj alat za svaku *smell* instancu prikazuje sa jedne strane deo koda koji je potrebno zameniti, dok se na drugoj strani nalazi novi kod u koji je potrebno pretvoriti taj stari kod kako bi taj problem bio rešen. Nakon promene svih detektovanih problema od strane *aDoctor*-a izvršena je ponovna analiza sa ciljem utvrđivanja čistog koda, odnosno potvrde da su sve *code smell* nepravilnosti koje detektuje *aDoctor* alat uspešno rešene. Sledeći korak predstavlja pronalaženje *code smell* nepravilnosti koje *aDoctor* nije automatski uspeo da detektuje. Detekcija novih *code smell* nepravilnosti se vrši ručno na osnovu ranijih teorijskih znanja. Problemi koji su identifikovani kroz ovakav način pretrage su:

- *Duplicated Code* — koji se rešava zamenom neke duplicirane metode i delegiranjem uobičajenog ponašanja.
- *Comments* — koji se rešava uklanjanjem nebitnih komentara.
- *Long Parameters List* — koji se rešava uvođenjem parametarskih objekata koji obuhvataju kontekst.
- *Move Method* — koji se rešava premeštanjem određene metode u klasu koja je najviše koristi.
- *Hashmap Usage* — koja se rešava sa zamenom *HashMap* strukture sa nekom efikasnijom strukturom podataka.

Nakon rešavanja prethodno navedenih problema koji su ručno identifikovani kroz proces pregleda koda koji je odrađen od strane detaljne korisničke pretrage prolaženjem kroz kompletnu strukturu projekta, bez upotrebe bilo kog alata koji omogućava automatsku detekciju.

Prilikom pretrage korišćena je referentna lista u okviru koje su dokumentovani svi tipovi *code smell* nepravilnosti koje su navedene kroz prethodne segmente rada i koje su u toku tog procesa pronađene uz dodatno usklađivanje vremenskih oznaka, izvršena je ponovna analiza koda upotrebom *JDeodorant* alata i pronađeni su sledeći tipovi *code smell* nepravilnosti: *Feature Envy*, *Long Method*, *God Class* i *Type-State checking* koji su rešeni korekcijom uz pomoć prethodno navedenog alata koji može da izvrši primenu tehnike refaktorisanja automatski na prethodno navedene četiri vrste *code smell* nepravilnosti, uz prikaz originalnog koda sa jedne strane i refaktorisanog koda sa druge strane. Konačno, nakon što je izvršena kompletna analiza kroz detekciju i korekciju identifikovanih problema preko dva prethodno navedena alata, kao i kroz manuelnu ručnu pretragu, otklonjene su sve neophodne *code smell* nepravilnosti i izvršena je ponovna detaljna analiza potrošnje energije od strane aplikacija. Kombinacija alata i ručne pretrage koja se zasniva na prethodnim teorijskim znanjima kroz detekciju i kasniju korekciju je izvršena kako bi se ostvarila manja potrošnja energije tokom izvođenja neke od testiranih aplikacija.

Analizom prve aplikacije pod nazivom Memo Game, je utvrđeno da je vrednost potrošnje energije tokom nova četiri izvođenja iznosila: 17,84mAh u prvom, 17,41mAh u drugom, 15,48mAh u trećem i 15,59mAh u četvrtom izvođenju. Na osnovu novih rezultata koji su dobijeni za prethodno navedenu aplikaciju izračunata je srednja vrednost potrošnje energije nakon procesa detekcije i korekcije koja iznosi 16,58mAh. Na osnovu prethodnih i novo dobijenih rezultata u vidu srednje vrednosti može se utvrditi da je refaktorisanje odnosno smanjenje određenih *code smell* nepravilnosti detektovanih uz kombinaciju alata i ručnom pretragom unutar prethodno navedene aplikacije doprinelo smanjenju potrošnje energije.

Potrošnja energije se smanjila za oko 5mAh u proseku u odnosu na slučaj kada se u aplikaciji nalazio početni izvorni kod. Analizom druge aplikacije pod nazivom Crazyflie, je utvrđeno da je vrednost potrošnje energije tokom nova četiri izvođenja iznosila: 29,29mAh u prvom izvođenju, 29,33mAh u drugom izvođenju, 28,59mAh u trećem izvođenju i 29,69mAh u četvrtom izvođenju.

Na osnovu novih rezultata koji su dobijeni za prethodno navedenu aplikaciju izračunata je srednja vrednost potrošnje energije nakon procesa detekcije i korekcije koja iznosi 29,22mAh. Na osnovu prethodnih i novo dobijenih rezultata u vidu srednje vrednosti može se utvrditi da je refaktorisanje odnosno smanjenje određenih *code smell* nepravilnosti detektovanih uz kombinaciju alata i ručnom pretragom unutar prethodno navedene aplikacije doprinelo smanjenju potrošnje energije. Potrošnja energije se smanjila minimalno za oko 0,2mAh u proseku u odnosu na slučaj kada se u aplikaciji nalazio izvorni kod.

Analizom treće aplikacije pod nazivom Block Puzzle Stone Wars, je utvrđeno da je vrednost potrošnje energije tokom nova četiri izvođenja iznosila: 9,56mAh u prvom izvođenju, 9,65mAh u drugom izvođenju, 9,33mAh u trećem izvođenju i 9,45mAh u četvrtom izvođenju. Na osnovu novih rezultata koji su dobijeni za prethodno navedenu aplikaciju izračunata je srednja vrednost potrošnje energije nakon procesa detekcije i korekcije koja iznosi 9,49mAh. Na osnovu prethodnih i novo dobijenih rezultata u vidu srednje vrednosti može se utvrditi da je refaktorisanje odnosno smanjenje određenih *code smell* nepravilnosti detektovanih uz kombinaciju alata i ručnom pretragom unutar prethodno navedene aplikacije doprinelo smanjenju potrošnje energije. Potrošnja energije se smanjila minimalno za oko 1mAh u proseku u odnosu na slučaj kada se u aplikaciji nalazio izvorni kod.

5. ZAKLJUČAK

Prilikom izrade ovog rada prikazan je prolaz kroz sve faze detekcije i korekcije Android *code smell* nepravilnosti primenom alata *aDoctor*, ručnom pretragom i primenom alata *JDeodorant*. Nakon kombinacije svake od korekcija izvršene su analize prikazane kroz rezultate srednjih vrednosti, koje su iznosile: 21,59mAh za izvorni kod prve aplikacije, 16,58mAh nakon prolaska kroz sve faze korekcije prve aplikacije, 29,42mAh za izvorni kod druge aplikacije, 29,22mAh nakon prolaska kroz sve faze korekcije druge aplikacije, 10,41mAh za izvorni kod treće aplikacije i 9,49mAh nakon prolaska kroz sve faze korekcije treće aplikacije. Na osnovu svega navedenog utvrđeno je da proces detekcije koji se sastoji od kombinacije *aDoctor* alata, ručne pretrage i *JDeodorant* alata, kao i korekcije problema nakon svake od detekcija koje su izvršene primenom prethodno navedenih alata i ručnom pretragom, pomaže aplikacijama u vidu smanjenja energetske potrošnje koja igra jednu od važnih uloga vezanih za izbor ovakvog tipa aplikacija.

6. LITERATURA

[1] https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009 (pristupljeno u decembru 2021.)

[2] H. Anwar, D. Pfahl, and S. N. Srirama, “Evaluating the impact of code smell refactoring on the energy consumption of android applications.” in 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, 2019, pp. 82–86.

[3] E. Iannone, F. Pecorelli, D. Di Nucci, F. Palomba, and A. De Lucia, “Refactoring android-specific energy smells: A plugin for android studio.” in Proceedings of the 28th International Conference on Program Comprehension, 2020, pp. 451–455.

[4] J. Oliveira, M. Viggiato, M. F. Santos, E. Figueiredo, and H. MarquesNeto, “An empirical study on the impact of android code smells on resource usage.” in SEKE, 2018, pp. 314–313. A.

[5] https://f-droid.org/ (pristupljeno u decembru 2021.)

Kratka biografija:

Filip Nikolić rođen je 23.08.1997. godine u Novom Sadu. Nakon završene srednje škole 2016. godine, upisuje osnovne akademske studije na Fakultetu tehničkih nauka, smer Inženjerstvo informacionih sistema. Diplomski rad je odbranio u septembru 2020. godine i iste godine upisuje master studije.

Kontakt: filipnikolic@uns.ac.rs